# Web サーバーに会議室を作る

## ～AppleScript の CGI プログラミング～

田中求之(mact@antares.ecn.fpu.ac.jp)

### PDF版について（1997年10月3日）

**この文書は、技術評論社の雑誌「The BASIC」（通称ざべ）１９９６年１２月号のために執筆した（執筆自体は９６年９月）記事の原稿を PDF にしたものです。Macintosh の Web サーバーにおける CGI 作成の解説ということで、私が作成し公開している EasyBBS という会議室（掲示板） CGI の原理を解説したものになっています。**

**The BASIC という雑誌は、もっぱら DOS, DOS/V, WIN のユーザーを対象とした雑誌でしたので（日本のパソコン・ジャーナリズムの中で特異な位置を占めていた雑誌でした）、Mac ユーザーで読んだ方は少ないと思います。そこで、編集部（元編集部）の許可を得て、PDF として公開することにしました。PDF 版作成にあたっては、図版を削りましたが、文章には手を入れていません。**

**なお、スクリプトなどがなるべく読みやすいように、改ページを多めに入れています。このため、ページの空白が目立つ部分がありますが、この点はご了承ください。**

**転載・再配布はご遠慮ください。**

## §０　はじめに

最近ではパソコンで運用されている Web サーバーの数も増えてきました。筆者は２年ほど前から Macintosh LC475 で Web サーバーを運用してきましたが、少し前までは Mac でサーバーをやっていること自体が珍しがられていたのですが、今や Mac でどのようなサーバーを運用しているのかを問われるようになってきたと感じています。ちょうど、ホームページを持つこと自体に意義がある段階が終わって、その中身が問われるようになってきたのと同じですね。そうした状況において、自分のパソコンでサーバーを運用することのメリットの一つとして、Web サーバーの「機能」を自分の手によって自在に広げていくことができるという点が挙げられると思います。Web サーバーでは、CGI と呼ばれる Web サーバーを補助するプログラムを作っていくことで、サーバーの機能を拡張できるようになっているのですが、この CGI を自分の思うように作っていけることが、自分のパソコンでサーバーを運用する醍醐味の一つでしょう。

そこで、この記事では、MacOS の Web サーバーにおける CGI の作り方を紹介します。AppleScript で CGI を作ることで、Web サーバーに会議室システム（メッセージボードあるいはＢＢＳとも呼ばれる、ユーザーが自由に意見を書き込んだりコメントを付けたりできるもののことです）を構築する例を紹介します。

なお、予めお断りしておきますが、CGI の具体的な作成方法に話を絞るために、MacOS の Web サーバーに関することや、AppleScript などについての詳しい説明は省かせていただきます。す

でに Mac で Web サーバーを運用しており、AppleScript に関しても多少の経験があることを前提に話を進めます。また、AppleScript のスクリプティング表現形式は英語を用います。ただし、どの言語でプログラミングを行おうと CGI 作りの点では共通する問題も少なくありませんので、この点では他の言語やあるいは他の機種で CGI を作ろうとされている方にも参考になるのではないかと思います。

## §１　基礎知識

Macintosh の専門誌ではない本誌で、いきなり AppleScript のプログラミングの話に入っていくのはさすがに気が引けますので、まず最初に CGI あるいは MacOS 上の CGI の特徴に関して、最低限のことをまとめておきます。

### ●CGI について

CGI (Common Gateway Interface) は、端的に言うならば、web サーバーの機能を補助するプログラムのことです。もともとは UNIX の web サーバーで導入されたものです。ブラウザーからのアクセスによって CGI が実行され、その結果が Web サーバーを介してユーザーへと送られるという仕組みになっており、UNIX においては環境変数や標準入出力を用いてサーバーと CGI との間のコミュニケーションが行われるようになっています。アクセスの都度プログラムが実行されるわけですから、ユーザーから送られてきた情報や実行される状況に応じて動的に変化するページを作ることが可能になります。ページカウンターやクリッカブルマップなどは、CGI の用いられている良く知られた例でしょう。その他にも、FORM によってユーザーからメッセージを受け取れるようする、データベースと Web サーバーを統合する、あるいはメーリングリストなどの他のサービスと Web を有機的に連携させることも可能です。いささか大げさに言えば、Web サーバーを、単なるホームページのファイルサーバーから Interactive さらには Intercreative なコミュニケーションを提供するサーバーへ展開することを可能にするのが CGI なのです。

### ●MacOS の CGI の特徴

CGI が具体的にどのように実装されるかは、OSによって異なっています。MacOS では以下のようになっています。

まず、MacOS では AppleEvent を使ってサーバーと CGI の連携が行われるようになっています。先ほど述べたように、CGI は Web サーバーとの間でデータをやり取りしながら実行されるプログラムです。UNIX では環境変数や標準入出力を利用してサーバーと CGI のコミュニケーションが行われるようになっているのですが、MacOS にはこうした仕組みはありません。そのため、MacOS では、System 7 で導入されたアプリケーション間通信の仕組みである AppleEvent が使われます。MacOS で走る Web サーバーも最近では数が増えてきましたが、CGI をサポートしているものはすべて共通の規格になっています。もともとは MacHTTP というシェアウェアの Web

サーバーが実装したものが、その後、事実上の標準規格として用いられるようになりました。MacHTTP で動く CGI であれば、他の CGI をサポートしている Web サーバーでも動くようになっています。

また、MacOS の CGI はアプリケーションとして作られます。MacOS にはシェルスクリプトやバッチファイルに相当するものはありません。強いて言うなら、AppleScript のスクリプトのファイルが相当しますが、CGI には使用しません。AppleScript で CGI を作成する場合も、アプリケーション形式にしておきます。

### ●MacOS での CGI の仕組み

MacOS でどのように CGI が実行されるのかを具体的に説明しましょう。

サーバーは、CGI に処理を任せるべきリクエストを受け取ると、まず CGI アプリケーションが起動しているかどうかを調べ、もし起動していない場合には、まず CGI アプリケーションを立ち上げます。それから、CGI に対して具体的な処理に必要なデータを AppleEvent によって送ります。

サーバーから CGI へは class=WWWΩ ID=sdoc の AppleEvent が送られます。ユーザーからのデータ（FORM に入力されたもの等）やブラウザなどに関する情報などは、すべてこの AppleEvent のパラメーターとして送られます（Appendix にまとめておきましたのでそれをご覧ください）。

サーバーから AppleEvent を受け取った CGI アプリケーションは、パラメーターで渡されたデータをもとに処理を行って、処理が終了した時点でサーバーに対して処理結果を reply します。これがサーバーから CGI へ送られた AppleEvent の返値としてサーバーに戻ってくることになります。サーバーは CGI からの reply を受け取ると、それをクライアントにむけて送り出します。

以上のような仕組みになっています。

### ●MacOS での CGI の作り方

MacOS で CGI を作る場合は、サーバーから送られてくる AppleEvent を受け取って処理を行い reply するというアプリケーションを作ることになります。ＣあるいはＣ＋＋でソースを書いて、コンパイラでアプリケーションに仕上げるという真っ当な方法以外に、AppleEvent が扱えるスクリプト言語、たとえば AppleScript、Frontier（Userland がフリーで配布しているスクリプティング環境です）あるいは HyperCard などを使って CGI を作成するという方法があります。UNIX から CGI を移植する場合などは MacPerl を用いれば Perl で CGI を作ることができます。これらのスクリプト言語を用いて CGI を作成した場合、処理速度の点ではコンパイラで作成したアプリケーションにはかないませんが、何よりも初心者にもハードルが比較的低く取り組みやすいということと、変更などのカスタマイズが簡単に行えるというのがメリットです。中でも AppleScript は、System 7.5（漢字Talk7.5）以降のシステムには標準で付いてきますし、AppleScriptに対応しているアプリケーションであればさまざまなコントロールが可能ですので、

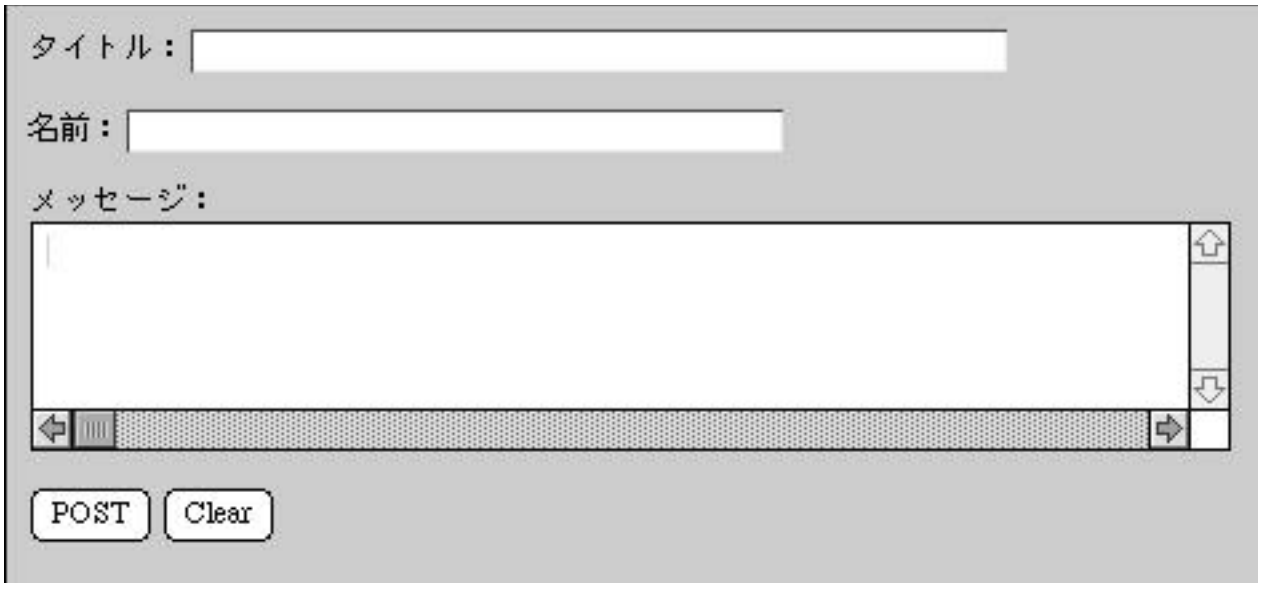
多くの Web サーバーにおいて CGI の作成に用いられています。

## §２　AppleScript による CGI 作成

具体的に AppleScript で CGI を作っていくときの方法について述べます。例として、以下のような FORM からのメッセージを受け取ってそれをファイルに記録しておくという、FORM の処理の基本的なもので説明していきます。

```
<FORM METHOD="POST" ACTION="form.acgi">
タイトル：<INPUT TYPE="TEXT" Name="Title" SIZE=50><P>
名前：<INPUT TYPE="TEXT" NAME="Name" SIZE=40><P>
メッセージ：<BR>
<TEXTAREA NAME="MSG" COLS=70 ROWS=5></TEXTAREA><P>
<INPUT TYPE="SUBMIT" VALUE="POST">
<INPUT TYPE="Reset" VALUE="Clear">
</FORM>
```

この FORM をブラウザで表示すると図１のようになります。




FORM タグの ACTION の部分で指定されている form.acgi というのが、この FORM のデータを扱う CGI アプリケーションの名前です。

拡張子が .acgi になっているのは、非同期で処理を行わせるアプリケーションであることを意味しています。Mac の CGI アプリケーションは、拡張子として.cgi と .acgi の２通りが選べます。どちらもプログラム自体には違いが無いのですが、この CGI を呼び出したサーバー側の動作が異なります。.cgi の場合には、サーバーは CGI アプリケーションに AppleEvent で処理を

指示すると、CGI から reply があるまで他の処理を中断して待ちます。一方、.acgi の場合には、サーバーは CGI に AppleEvent を送ると、他の処理を続行し、CGI 側からの reply が送られてきた時点で再び CGI に関する処理を再開するようになっています。ですから、基本的には CGI アプリケーションは拡張子 .acgi にしておけばよいのですが、サーバーの処理と CGI の処理が平行して行われることになりますので、CGI の処理の速度は低下します（MacOS は疑似マルチタスク環境です）。このため、CGI の処理速度を優先させたい場合や、処理の遅いマシンなどの場合に .cgi の拡張子を用いることもあります。ここでは .acgi を用いることにします。

さて、ページにアクセスしてきたユーザーがフィールドにデータを書き込み、POST ボタンをクリックすると、それを受けて、Web サーバーは form.acgi に対して AppleEvent で処理を指示するメッセージを送ります。ユーザーがページに書き込んだメッセージは、AppleEvent の class=post のパラメーターに収められて送られます。ですから、form.acgi の基本的なスクリプトは以下のようになります。

```
on «event WWWΩsdoc» given «class post»:post_args
    ここに具体的な処理が入る
    return （ページのデータ）
end «event WWWΩsdoc»

* « は option + \ で入力する文字です。< を２つ重ねたものではありません
* » は option + shift + \ で入力する文字です。> を２つ重ねたものではありません
* Ω （オメガ）は option + z で入力します
```

AppleEvent (class= WWWΩ, ID=sdoc) によってメッセージが送られてくるわけですから、これを受けとめて処理を行うハンドラーは上のような形になります。

AppleScript においては、AppleEvent を受け取るハンドラーを書く場合は、上のように «event XXXXyyyy» (XXXX = Event Class, yyyy = Event ID) というメッセージを受けとめる形のハンドラーを書きます。また、AppleEvent には対応していても AppleScript に対応していないソフトに対して «event XXXXyyyy» というメッセージで AppleEvent を送ることが可能です。

また、処理が終わったら、かならずサーバーに対してページのデータを return しなければなりません。この場合、「ページのデータ」というのは、HTML で書かれたページの内容に HTTP のヘッダーをくわえたものにしなければなりません。HTTP のヘッダーというのは、サーバーとブラウザの間でページに関する情報を交換するために HTTP のプロトコルとして決められているものです。ブラウザでは通常は表示されませんが、サーバーから送られてくるページのデータには、必ずヘッダーが先頭に付いているのです。ファイルとして書いたページなどの場合は、サーバーが自動的にヘッダーを生成してからデータをブラウザにむけて送り出すのですが、CGI の処理の場合は、このヘッダーを作るのも CGI アプリケーションが行わなければなりません。サーバーは CGI から返されたデータを、そのまま何も手を加えずに直接ブラウザへと送るようになっているのです。

具体的には以下のようなヘッダーを追加した上で、ページのデータを組み立てます。

```
HTTP/1.0 200 OK
Server: MacHTTP
MIME-Version: 1.0
Content-type: text/html
```

最後に必ず空行が入ります。また、ヘッダー部分は改行は return+lineFeed (CR + LF) を用います。スクリプトの冒頭でこのヘッダーを property として定義しておいてそれを使用するのが便利でしょう。form.acgi では、メッセージを受け取ったら、Thank you という感謝の言葉を表示するようにしましょう。この場合は、以下のようになります。

```
property crlf : (ASCII character 13) & (ASCII character 10)
property http_10_header : "HTTP/1.0 200 OK" & crlf ¬
    & "Server: MacHTTP" & crlf ¬
    & "MIME-Version: 1.0" & crlf ¬
    & "Content-type: text/html" & crlf ¬
    & crlf

on «event WWWΩsdoc» given «class post»:post_args
    ここに具体的な処理が入る
    return http_10_header & "<TITLE>Thank You</TITLE>" & crlf ¬
          & "<h3>メッセージをありがとうございました！</h3>"
end «event WWWΩsdoc»
```

続いて、送られてきたメッセージを処理してファイルに書き込む処理の説明に移りましょう。class = post のパラメータの内容は post_args という変数に収められますから、これを処理することになります。この場合、単純に post_args をファイルに書き込めばよいというものではないのです。Web(HTTP) では、FORM のデータは特別な形にまとめられて送られてくるようになっていますので、これを解きほぐして、書き込まれたメッセージを取り出す必要があります。

Form ページにユーザーが書き込んだ情報は、書き込んだデータが URL形式に変換された上、フィールド名とデータが = をデリミタにして結合され、さらにすべてのフィールドのデータが & をデリミタにして一つにまとめられて送られてくるようになっています。たとえば、先ほどの FORM において

**タイトル：Web に会議室を**
**名前：田中求之**
**メッセージ：CGI で会議室を作る**

という書き込みをして、POST ボタンをクリックしたとします。すると、最終的に form.acgi が post_args として受け取るのは、以下のようなデータになっているのです。

```
Title=Web%20%82%C9%89%EF%8Bc%8E%BA%82%F0&Name=%93c%92%86%8B%81%94V&MSG
=CGI%20%82%C5%89%EF%8Bc%8E%BA%82%F0%8D%EC%82%E9
```

メッセージの内容は、アルファベットと一部の記号を除いて、すべて %XX (XX は１バイト毎の文字の ASCII コードを HEX で記したもの）という表記に変換（ URL エンコードと言います）されています。これをもとに戻す（デコード）という作業を行うことになります。この処理の手順は以下のようになります。

１：& をデリミタにして、フィールド毎に分解する

```
Title=Web%20%82%C9%89%EF%8Bc%8E%BA%82%F0
Name=%93c%92%86%8B%81%94V
MSG=CGI%20%82%C5%89%EF%8Bc%8E%BA%82%F0%8D%EC%82%E9
```

２：それぞれのフィールドのデータを、= をデリミタにしてフィールド名とメッセージに分離し、メッセージ部分を取り出す

```
Web%20%82%C9%89%EF%8Bc%8E%BA%82%F0
%93c%92%86%8B%81%94V
CGI%20%82%C5%89%EF%8Bc%8E%BA%82%F0%8D%EC%82%E9
```

３：%XX 表記をデコードして、もとの文字列にもどす

```
Web に会議室を
田中求之
CGI で会議室を作る
```

このような処理を行って、やっとファイルに書き込むデータ（ユーザーが書き込んだ通りのデータ）が取り出せます…　といいたいところなのですが、実はもう一つやっかいな問題が残っています。それは漢字コードの問題です。ユーザーから送られてきたメッセージの漢字コードを判別した上で、適切な変換を行う必要があるのです。

Mac あるいは WINDOWS などのパソコンでは、漢字コードとして SHIFT-JIS (SJIS) というコードが使われていますが、UNIX などでは EUC というコードが使われています。また、インターネットのメールやニュースでは JIS という漢字コードが用いられます。このように、インターネット上で用いられる可能性がある漢字コードは３種類あるわけです。この３つのコードに互換性はありません。そして、さまざまな機種・環境の人が皆繋がるのがインターネットなんですから、書き込む際に用いられる漢字コードとしてどれが用いられるかは、ユーザー次第ということになります。Mac においてファイルに書き込む場合は、漢字コードは SJIS を用いるのが普通ですから、FORM のメッセージを処理する CGI においては、送られてきたメッセージの漢字コードを判定した上で、EUC や JIS が用いられていた場合には、それを SJIS に変換するという処理が必要になるのです。この処理を忘れると、判別不明のメッセージが記録されてしまうという問題が起きます。ですから、CGI における post_args の処理として、最後に漢字コードの判別と変換が必要になります。

このように、post_args の処理は、なかなか面倒ですし、漢字コードの処理のように

AppleScript だけでは処理できないものも含まれます。そこで、ここでは osax を用いることにします。

osax というのは、AppleScript の機能を拡張するモジュールのようなものです。システムフォルダーの「機能拡張」フォルダーの中に「スクリプティング機能拡張」（英語版の AppleScript をインストールした場合には「Scripting Additions」）という名前のフォルダーがあることと思います。この中に入っているのが osax です。端的に言うならば AppleScript に新しいコマンドを追加するためのファイルだと言ってもよいでしょう。

CGI のスクリプトを書く上で役に立つコマンドを集めた osax がフリーウェアでありますので、それを使います。Tanaka's osax という名前のフリーウェアがインターネットで公開されています。名前で気がつかれたかも知れませんが、筆者が作成し公開しているものです。筆者は、これまでさまざまな CGI を作ってきたのですが、その中で、AppleScript のスクリプトでは処理が遅いものや、漢字コードの処理のように AppleScript では手に余るものを osax にしてきました（ちなみに、osax はコンパイラを使ってコードリソースとして作成するのですが、筆者の場合は HyperTalk でソースを書いて、それを CompileIt! という HyperTalk コンパイラでコンパイルして作成しています）。以下の説明では、この osax を使ってスクリプトを書いていくことにします。Tanaka's osax は以下の URL のページから手に入ります。最新版は 1.0b10 です。

- Tanaka's Tool
  <http://mtlab.ecn.fpu.ac.jp/Tanaka_tool.html>

Tanaka's osax をインストールした場合、先ほど説明した post_args からメッセージを取り出す処理は、以下のように一つのコマンドを呼ぶだけで済みます。

```
set myList to DecodeJArgs post_args
```

これで myList = {"Web に会議室を", "田中求之","CGI で会議室を作る"} となりますので（漢字コードの判別と SJIS への変換も行います）、あとはこれをファイルに記録するスクリプトを書けばよいわけです。

完成した form.acgi のスクリプトは以下のようになります。messages というフォルダーの中に、MSG_<タイムスタンプ> というファイル名で保存していくようになっています。タイムスタンプは、処理した日時の年・月・日・時・分・秒を繋げた文字列です。これを用いることで、ファイル名の重複を回避できますし、ファイル名でソートしたときにメッセージが古い順番で並ぶことが保証されます。TimeStamp を得るのと、データをファイルに書き込むのにも Tanaka's osax のコマンドを使っています。また、メッセージの改行コードを Mac で用いられる return に統一するのにも osax の xReturner というコマンドを用いています（漢字コードが機種によって異なるのと同様に、改行コードも異なっているため、送られてくる改行コードが return とは限らないのです）。

```
property crlf : (ASCII character 13) & (ASCII character 10)
property http_10_header : "HTTP/1.0 200 OK" & crlf ¬
    & "Server: MacHTTP" & crlf ¬
    & "MIME-Version: 1.0" & crlf ¬
    & "Content-type: text/html" & crlf ¬
    & crlf

property msgFldr : "Macintosh HD:myServer:messages:"

on «event WWWΩsdoc» given «class post»:post_args

    set myList to DecodeJArgs post_args

    set myData to "タイトル:" & (item 1 of myList) & return ¬
          & "名前：" & (item 2 of myList) & return ¬
          & "メッセージ：" & return & (xReturner (item 3 of myList))

    set myFile to msgFldr & "MSG_" & (TimeStamp)

    writeToFile myData to file myFile

    return http_10_header & "<TITLE>Thank You</TITLE>" & crlf ¬
          & "<h3>メッセージをありがとうございました！</h3>"
end «event WWWΩsdoc»
```

* msgFldr はメッセージを保存しておくフォルダーのパスです

このスクリプトをアプリケーション（アプレット）としてセーブすれば CGI の出来上がりです。なお、アプリケーションとしてセーブする際には、「実行後、終了しない（Stay Open)」と、「初期画面を表示しない (Never Show Startup Screen)」の２つのオプションを共に有効にする（チェックする）ことを忘れないでください。特に最初のオプションをチェックするのを忘れると、CGI アプリケーションとしては機能しないものになりますので注意してください。


この form.acgi が、Mac の CGI アプリケーションの基本形といってもよいものになります（ CGI は必ずしも FORM の処理ばかりに使うものではありませんが）。以下では、これをもとにして、会議室システムを作っていくことにします。

## §３　会議室システムを作る

### ●会議室システムの考え方

ここでいう会議室システムというのは、以下のような仕組みのことです。

**１：ユーザーの書き込みがページとして即座に登録される**
**２：登録されているメッセージが簡単にブラウズできる**
**３：メッセージに対して他の人がコメントを付けられる**

これ以外にも、メッセージの検索機能などつけ加えるべきものはいくつもありますが、ここでは、上の基本的な機能を作り上げていくことにします。それぞれの機能は、次のような仕組みを CGI 側で用意することで実現できます。

１の書き込みの登録については、先ほどの form.acgi と同じような処理を行い、ファイルに書き込む際に、HTML のタグを付けてホームページの形式に仕立てておくようにすればよいわけです。つまり、受け取ったメッセージを HTML のファイルとして保存するわけです。そして、書き込みが終了したら、出来上がったページを表示するようにします。

２については、メッセージを保存するフォルダーのインデックス（ファイルのリスト）を CGI が生成すればよいわけですが、ファイル名を表示したのでは中身が分かりませんので、ページのタイトルを抜き出して表示するようにします。また、発言あるいはコメントが付けられた日時が分かりやすいように、ファイルの最終更新日を表示するようにします。

３についてどのような方法を取るのかによって CGI のスクリプトが全く異なったものになります。ある意味で、会議室（メッセージボード）を作るときには、この３をどのように考えるかが決めてであると言えるでしょう。そこで、少し詳しく説明することにします。

たとえば、インターネットで会議室というと NetNews を思い浮かべる方も多いと思いますが、NetNews にみられるように、コメントも独立した発言としてそれぞれ別のページ（ファイル）としておき、ブラウズの際にリンクによってスレッドを辿っていくという構成にすることもできます。しかし、この方法は、発言－コメントのリンクを管理するのが面倒ですし、アクセスしてブラウズする際もコメントを辿っていくのが面倒です。そこで、一つのページに、最初の発言とそれに対するコメントがすべて書き込まれるようにし、さらにページの最後の部分にコメント書き込み用のフィールドも付いているということにしましょう。図が出来上がった会議室へアクセスしたときの様子です。


この方法の特徴は、最初の発言から後に続くコメントがページの上から下へと並んでいき、最後に自分のコメントを書き込む欄がある、という自然な流れになっていることです。最新のコメントだけを読みたいときなどにはページをスクロールしていかなければならないのが面倒ではありますが、スレッド全体が１つのページになっていますので、途中から参加した人でも話の流れが追いやすということもメリットとして挙げられるでしょう。

この１スレッド＝１ページ方式を実現する場合のポイントは、コメントをページに追記していく

ようにするということと、コメントを書き込む FORM が必ずページの最後に表示されるようにするということです。このためには、発言およびコメントのメッセージはファイルに記録するようにし、FORM によるコメント欄は、ページを表示する際に CGI によって追加を行うようにします。こうすることで、コメントをファイルに追記する処理も簡単になりますし、また FORM タグを CGI が動的に作り出すことで、コメントを書き加えるべきファイルの情報なども的確に指示できるようになります。

**●必要となる処理（ハンドラー）**

では具体的にスクリプトを書いていくことにします。bbs.acgi という CGI アプリケーションを作ることにします。

会議室システムの場合、一つの CGI で以下の処理を行わせることになります。

**１：新規発言のページ化と保存**
**２：ページの表示**
**３：コメントの追加**
**４：発言一覧の表示**

一つの CGI でこのような複数の処理を行わせるわけですから、サーバーから CGI が呼ばれる際に、どの処理を行うべきなのかを判別できるようにしなければなりません。つまり、CGI の呼び出しの際に処理すべき内容を指示したパラメーターを渡す必要があります。これには search_args と呼ばれるものを用いることにします。これは CGI を呼び出す URL の中で ? の後ろに書いておくもので、検索エンジンの検索語句や、クリッカブルマップでのクリックした座標の位置などをサーバーに知らせるのに使われるものです。たとえば

```
http://your.host/bbs.acgi?hello
```

という URL でサーバーにアクセスすると、? の後ろに書かれた hello が search_args として bbs.acgi という CGI アプリケーションに渡されるようになっているのです。Mac の場合はサーバーからは class=kfor のパラメーターでこの内容が CGI に伝えられます。

そこで、CGI は search_args と、メッセージの内容が入っている post_args をチェックすることで、処理の内容を判別するようにします。

search_args **と** post_args **が共に空のとき**
**→登録されている発言の一覧表示（会議室のトップページの表示）**
search_args **は空で、**post_args **にメッセージが入っているとき**
**→新規発言の登録**
search_args **にページ名（**MSG_XXXXX **）が入っていて、**post_args **は空**
**→指定されたページの表示**
search_args **に** ADD_ + **ページ名で、**post_args **にメッセージ**
**→指定されたページにコメントの追加**

そして、それぞれの処理はハンドラーに分けてスクリプトを書き、メインのハンドラーから必要に応じて呼び出すというスタイルにします。エラー処理もつけ加えた bbs.acgi のメインのハンドラーは以下のようになります。

```
property crlf : (ASCII character 13) & (ASCII character 10)
property http_10_header : "HTTP/1.0 200 OK" & crlf ¬
    & "Server: MacHTTP" & crlf ¬
    & "MIME-Version: 1.0" & crlf ¬
    & "Content-type: text/html" & crlf ¬
    & crlf
property redirect_header : "HTTP/1.0 302 Found" & crlf ¬
    & "Server: WebSTAR/1.0 ID/ACGI" & crlf ¬
    & "MIME-Version: 1.0" & crlf ¬
    & "Location: "


property msgFldr : "Macintosh HD:myServer:messages:"

on «event WWWΩsdoc» given «class kfor»:search_args, «class
post»:post_args
    try
          if search_args = "" then
                if post_args = "" then
                      会議室のトップページ
                      return topPage()
                else
                      新規発言登録
                      return makeNew(post_args)
                end if
          else if "ADD_" is in search_args then
                コメント追加
                return addComment(search_args, post_args)
          else if "MSG_" is in search_args then
                ページ表示
                return showPage(search_args)
          else
                error "CGI の呼び出し方が間違っています"
          end if
    on error errMsg
          return http_10_header & "<TITLE>Error</TITLE>" & crlf ¬
                & "<h3>CGIの実行中にエラーが生じました</H3>" & crlf ¬
                & "<B>" & errMsg & "</B>"
    end try
end «event WWWΩsdoc»
```

* redirect_header という property については、後で説明します。

続いて、個々の処理（ハンドラー）について説明していきます。

## １：新規発言のページ化と保存（ makeNew ハンドラー）

これは§２で作成した form.acgi のスクリプトを改良したものになります。post.html という発言投稿用のページを用意しておき、そのページの FORM に書き込まれたメッセージに HTML のタグを付けて、発言保存用のフォルダーの中に書き込むという処理をおこないます。

ただし、form.acgi とは違って、新しいページを作成した後には、書き込んだページを表示するようにします。このため、ハンドラーの最後に return する内容が form.acgi とは異なります。§２の form.acgi ではヘッダーにページのデータを付けたものを返していましたが、今度はブラウザに対して新しいページへのアクセスの指示する redirect ヘッダーと呼ばれるものを返すようにします。

redirect ヘッダーというのは、以下のようなヘッダーです。

```
HTTP/1.0 302 Found
Server: MacHTTP
MIME-Version: 1.0
Location: <URL>
```

（最後に空行。改行コードは CR+LF。<URL> で新しいアクセス先を指示）

このヘッダーが CGI からサーバー経由でブラウザに送られると、受け取ったブラウザは、指示された URL のページへ自動的にアクセスし直すようになっています。この場合は、ヘッダーを送り返すだけでよく、ページのデータ等は必要ありません。

そこで、新しいページを書き込んだ際には、この redirect ヘッダーによって、書き込んだページにアクセスを行うように指示を出すことにします。redirect ヘッダーはスクリプトの冒頭で property として定義しておくことにいます。そして、CGI によってページを表示するための URL をブラウザに指示します。

ファイルに書き込んだページを表示するのに、わざわざ redirect によって改めて表示用の URL でアクセスし直させるのは、ユーザーがブラウザで画面の reload を行ったときに２重の書き込みが行われるのを防ぐためです。

makeNew ハンドラーは以下のようになります。

```
on makeNew(post_args)

    post_args をデコードする
    set myList to DecodeJArgs post_args

    set myTitle to item 1 of myList
    set myName to item 2 of myList
    set myMsg to item 3 of myList

    メッセージの改行コードを return に統一
    set myMsg to xReturner myMsg

    メッセージ中の < と > を表示用の表記に変換
    これにより、メッセージ中のタグはすべて無効になる
    set myMsg to xReplace myMsg search "<" replace "&lt;"
    set myMsg to xReplace myMsg search ">" replace "&gt;"

    HTML のページのデータに仕立てる
    メッセージは <PRE> によって表示する
    set myPage to "<TITLE>" & myTitle & "</TITLE>" & return ¬
          & "<H2>" & myTitle & "</H2>" & return ¬
          & "<H3>発言者：" & myName & "</H3>" & return ¬
          & "<TT>(Date: " & ((current date) as string) &
")</TT><P><HR><P>" & return ¬
          & "<PRE>" & myMsg & "</PRE><P><HR size=4><P>" & return

    ファイル名の決定
    set myFile to "MSG_" & (TimeStamp)

    ファイルに書き込み
    writeToFile myPage to file (msgFldr & myFile)

    書き込まれたページを表示するように redirect の指示を返す
    URL のあとに crlf が２つ必要
    return redirect_header & "bbs.acgi?" & myFile & crlf & crlf

end makeNew
```

* xReplace は置換を行う Tanaka's osax のコマンドです

## ２：ページの表示（ showPage ハンドラー）

search_args で指示されたページを読み込み、それにコメント欄の FORM を追加した上で return します。処理としては非常に簡単ですが、FORM の部分を書くときには、HTML のタグで用いる " を \ でエスケープするのをわすれないでください。また、ACTION の部分にコメント追加用のコマンドを埋め込むことと、会議室のトップページへのリンクを付けておくことを忘れないでください。

```
on showPage(search_args)

    set myPage to readFromFile file (msgFldr & search_args)

    return http_10_header & myPage & return ¬
          & "<FORM METHOD=\"POST\" ACTION=\"bbs.acgi?ADD_" & search_args
& "\">" & return ¬
          & "<H4>コメント</H4><P>" & return ¬
          & "発言者:<INPUT TYPE=text NAME=name SIZE=40><p>" & return ¬
          & "<TEXTAREA NAME=comt ROWS=5 COLS=70></TEXTAREA><P>" & return
¬
          & "<INPUT TYPE=submit VALUE=\"Post Comment\"><INPUT TYPE=reset
VALUE=\"Reset\">" & return ¬
          & "</FORM><P><HR><P><h4><UL>" & return ¬
          & "<LI><A HREF=\"bbs.acgi\">Top Page</A></UL></H4><HR><P>"

end showPage
```

* readFromFile はファイルのデータを読み込む Tanaka's osax のコマンドです

## ３：コメントの追加（ addComment ハンドラー）

search_args の中に ADD_ + ページ名の形でコメントを付けるべきページが指示されていますので、ADD_ を削ってページ名を取り出した上で、そのページに post_args をデコードして取り出したメッセージを追記していきます。ページにコメントをくわえた後、先ほどの新規発言登録の場合と同じように redirect ヘッダーを使って、コメントが加わったページにアクセスし直す指示を返すようにします。

なお、コメント追加を指示する search_args にわざわざ ADD_ をつけ加えたのは、これによってページ表示の場合とはアクセスの URL が異なることになりますので、コメントが追加されたページを表示する場合（ブラウザ側からは同じページにアクセスし直すことになる）、確実に更新されたページが表示されるようになるからです。

```
on addComment(search_args, post_args)

    コメントを追加するページの確定
    set AppleScript's text item delimiters to {""}
    set tgPage to (characters 5 through -1 of search_args) as string

    post_args のデコード
    set myList to DecodeJArgs post_args
    set myName to item 1 of myList
    set myMsg to item 2 of myList

    メッセージの改行コードを return に統一
    set myMsg to xReturner myMsg

    メッセージ中の < と > を表示用の表記に変換
    set myMsg to xReplace myMsg search "<" replace "&lt;"
    set myMsg to xReplace myMsg search ">" replace "&gt;"

    ページに追加するデータを整える
    set myData to "<B>" & myName & "</B> さんからのコメント<BR>" & return ¬
          & "<TT>( Date: " & ((current date) as string) & ")</TT><p>" &
return ¬
          & "<PRE>" & myMsg & "</PRE><HR><P>" & return

    ファイルへの追加
    appendToFile myData to file (msgFldr & tgPage)

    コメント済みのページへのリダイレクト
    return redirect_header & "bbs.acgi?" & tgPage & crlf & crlf

end addComment

* apendToFile はファイルにデータを追記する Tanaka's osax のコマンドです
```

## ４：発言一覧の表示（topPage ハンドラー）

会議室のトップページの表示を行うためのハンドラーです。このページで最低限表示すべき内容は、すでに登録されている発言の一覧と、発言投稿用のページへのリンクです。しかし、なんと言っても会議室の入り口のページでもありますから、それだけでは味気ないでしょう。画像を表示したり、参加者へのメッセージを書いておいたり、あるいは自分の他のページへのリンクを埋め込んだりと、Web ならではの色々な華を添えられるようにしておきましょう。そこで、ページを「ヘッダー＋発言リスト＋フッター」の３つの部分に分けて、発言リストは CGI が生成し、ヘッダーとフッターは予めファイルに書いておいたものを読み込み、それらを合わせたものを表示するようにします。もちろん、CGI のスクリプトの中にヘッダーとフッターの部分を書いておくことも可能ですが、こうすると、画像一つ取り替えるためにわざわざ会議室の運用を止めて CGI のスクリプトを書き直さなければならなくなり、面倒です。ファイルにしておけば、いつでも気軽に、変更することが可能になります。ヘッダーファイル（BBS_Header）とフッターファイル（BBS_Footer）はスクリプト中にパスを書いておくことにします。

発言リストの作成は、次のような手順で行います。まず発言が保存されているフォルダーの中のファイルのリストを得ます。これには Tanaka's osax の dirLister というコマンドを用います。ファイルが存在していた場合には、それぞれのファイルからタイトル（<TITLE> タグで囲まれた部分）を抜き出し、これとファイルの最終更新日とをリストに追加していきます。タイトルの抜き出しには Tanaka's osax の getPageTitle というコマンドを用います。また、ファイルのタイトルの表示には、そのファイルの表示用のリンクを張っておきます。

なお、発言リストの生成にあたっては、最初にリスト型の変数 myList を作っておき、これに各ファイルの情報をリストのアイテムとして追加していき、最後に AppleScript の text item delimiter を return にしておいてから myList をリスト型からテキスト型へ型変換を行うという処理を行います。ページとして表示するデータを生成していくわけですから、最初からテキスト型の変数に追加していくのが自然なのですが、処理速度を上げるためにはこちらの方がよいのです。たとえば、

１行目
〜
１０００行目

という単純なテキストデータを作成する場合でも、最初からテキスト型の変数に追加していく

```
set myList to ""
repeat with x from 1 to 1000
    set myList to myList & (x as string) & " 行目" & return
end repeat
```

というスクリプトですと、筆者の Quadra 840AV では約２０秒かかりますが、リスト型で生成していき最後にテキストに変換する

```
set myList to {}
repeat with x from 1 to 1000
    set end of myList to (x as string) & " 行目"
end repeat
set AppleScript's text item delimiters to {return}
set myList to myList as string
```

```
set AppleScript's text item delimiters to {""}
```

というスクリプトは、スクリプト自体の行数は長いのですが、実行時間は約７秒と、先ほどの３分の１ほどの時間で処理できてしまいます。このように、次々と項目を追加していくような処理の場合は、たとえ最終的に必要なのがテキスト型のデータであっても、リスト型の変数を使ってリストとして処理していき、最後にテキストに変換した方が早いのです（変換の際に text item delimiter を return に設定するのを忘れないこと）。AppleScript はテキスト処理（文字列の処理）が早くないのは確かですが、リストと text item delimiter をうまく使いこなすことで、速度をかせぐことができます。筆者はリストをうまく使うのが AppleScript の秘訣だと思っています。

すこし横道にそれてしまいましたが、topPage ハンドラーのスクリプトは以下のようになります。

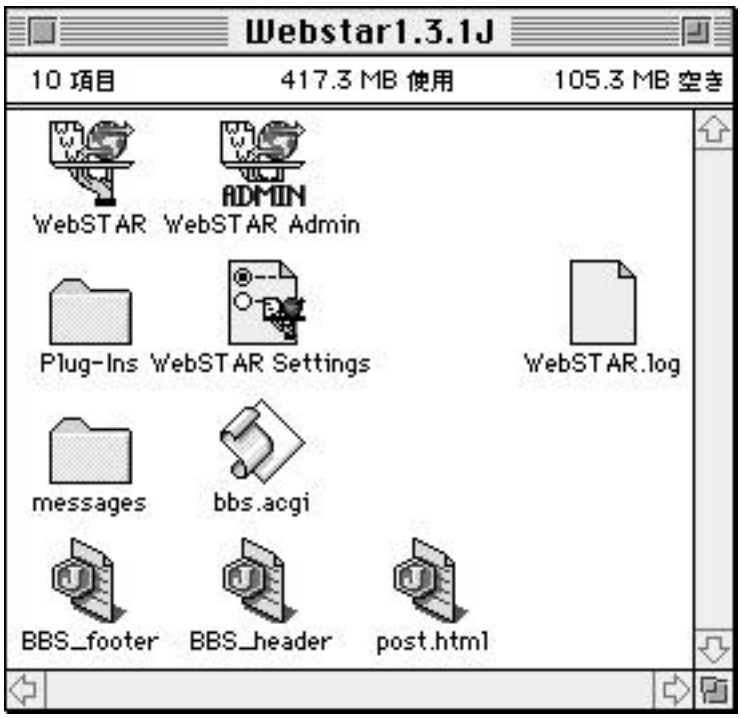
```
on topPage()
    フォルダーの中の発言ファイルの名前のリストを得る
    try
          set myList to dirLister alias msgFldr ofName "MSG_"
    on error
          set myList to {}
    end try

    登録されている発言の数を調べる
    set myMsgNm to count items of myList

    ページに表示するメッセージを作成する
    set myData to {}
    set end of myData to ((current date) as string)
    set end of myData to "<P><H4>現在 " & (myMsgNm as string) & "  個のメッセージが登録されています</H4>"

    if myMsgNm ≠ 0 then

          発言の数が０でなかったら、発言タイトル一覧の作成を行う
          ファイル毎に発言のタイトルと最終更新日を抜き出してリストにくわえていく
          set end of myData to "発言タイトル一覧：<UL>"
          repeat with myF in myList
                set tgF to alias (msgFldr & myF)

                タイトルを抜き出す
                set myTitle to getPageTitle tgF

                表示用のリンクを埋め込みながらリストに追加する
                set end of myData to "<LI><B><A HREF=\"bbs.acgi?" & myF & "\">" & myTitle & "</B></A><BR>"

                ファイルの最終更新日も追加
                set end of myData to "<TT>( " & ((modification date of (info for tgF)) as string) & " )</TT><P>"
          end repeat

          set end of myData to "</UL><P>"
```

```
    end if

    myData をリスト型からテキスト型へ型変換を行う
    set AppleScript's text item delimiters to {return}
    set myData to myData as string
    set AppleScript's text item delimiters to {""}

    ヘッダーとフッターを読み込む
    set myHeader to readFromFile file "Macintosh HD:myServer:BBS_header"
    set myFooter to readFromFile file "Macintosh HD:myServer:BBS_footer"

    ヘッダー&リスト&フッターをページとして return する
    return http_10_header & myHeader & myData & myFooter

end topPage
```

* 発言投稿用ページ（post.html）へのリンクは BBS_footer ファイルに書いておくことにしてあります。

これで bbs.acgi のスクリプトは完成です。

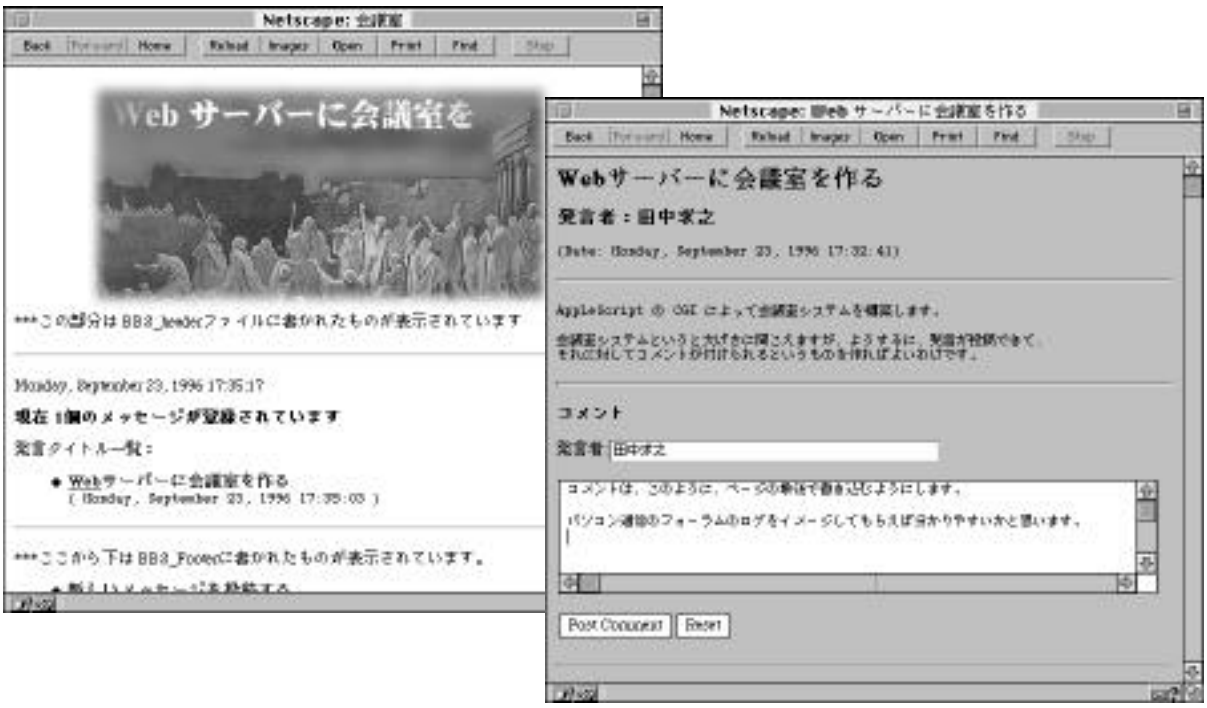
さっそくたち上げてみましょう。スクリプトの構文確認を行ってから bbs.acgi という名前のアプリケーションとしてセーブします。bbs.acgi、BBS_header、BBS_footer、post.html それと発言保存用のフォルダーを Web サーバーと同じフォルダーに収めて、bbs.acgi をたちあげます。




そして、ブラウザで

http://your.host/bbs.acgi
（your.host の部分は、あなたの Web サーバーのホスト名あるいは IP アドレス）

にアクセスしてみてください。会議室のトップページが表示されます。新規発言、あるいはコメントの書き込みなどを行って動作を確かめてみてください。




## §４　最後に

発言をポストする、発言をブラウズする、発言にコメントを付ける、という会議室の基本的な機能はこれで完成です。このままでも十分にコミュニケーションを行うためのツールとして活用できると思います。最近流行のグループウェアには比べようはありませんが、何よりも手軽に運用できるのがメリットです。また、スクリプトを改良していくことで、さまざまな機能を追加していくこともできます。メッセージの検索や複数の会議室の開催など、アイデア次第でいくらでも拡張していくことが可能ですので、皆さんも自分なりに改造してみてください。

なお、私のサーバーで実際に会議室を運用しています。ここで紹介したものよりも機能は増やしてありますが、基本的には同じ原理のものです。AppleScript で作った CGI がどの程度使いものになるのか気になる方は、覗いてみてください。

- Web Scripter's Meeting
  <http://mtlab.ecn.fpu.ac.jp/webcon.mtxt>

MacOS での Web サーバーの運用と CGI について話し合うための会議室になっていますので、この記事に関係する質問などがあれば、遠慮なく投稿してください。

また、この記事で紹介した会議室システムを私なりに改良したものを EasyBBS という名前で私のサーバーに登録してあります。もちろんフリーウェアです。興味のある方はお試しください。

- EasyBBS
  <http://mtlab.ecn.fpu.ac.jp/easyBBS/>

からダウンロードできるようになっています。先ほど紹介した Web Scripter's Meeting では Tanaka's osax および EasyBBS のオンラインサポートも行っています。

参考文献

・『Macintosh インターネットサーバー構築術』 Cyber Barbarians 編（オーム社）

・"Planning and Managing WEB SITES on the Macintosh" Jon Wiederspan & Chuck Shotton 著（Addison Wesley Developer Press）

Appendix : CGI に送られる AppleEvent のパラメーター一覧表

（）の中がパラメーターのclass。

・path_args ('----')　URLで$の後に追加されるユーザーからの情報

・search_args ('kfor')　URLで?の後ろに追加されるユーザーからの情報

・post_args ('post')　METHOD=POSTの場合にFORMに書き込まれたデータが収められている

・method ('meth')　ユーザーが情報を送るのに用いた方法（GETかPOSTのどちらを用いたか）

・client_address ('addr')　ユーザーの IP アドレスまたはホスト名。どちらの型で渡されるかはWeb サーバーの設定による。

・username ('user')　REALMによるユーザー確認を行った場合のユーザー名。ユーザー確認を行っていないときには空。

・ password ('pass') ---　REALMによるユーザー確認を行った場合のパスワード。ユーザー確認を行っていないときには空。

・from_user ('frmu')　ユーザーのE-mailアドレス。ただし、ブラウザがこの情報をサーバーに知らせるようになっていなければ空。現時点でこの情報をサポートしているソフトは少ない

・server_name ('svnm')　Web サーバーのIPアドレスまたはホスト名。どちらの型で渡されるかはWeb サーバーの設定による。

・server_port ('svpt')　Web サーバーが使用しているポート番号（通常は８０）

・script_name ('scnm')　CGIアプリケーションのURL（相対パス）

・content_type ('ctyp')　CGIアプリケーションの呼び出しを行ったリクエストの

MIMEタイプ

・referer ('refr')　CGIアプリケーションを呼び出すURLが書いてあったページのURL。つまり、どのページからこのCGIアプリケーションが呼ばれているか。

・user_agent ('Agnt')　ユーザーが使用しているブラウザの名前

＊以下のパラメーターは WebSTAR で追加されたもの

・action ('Kact')　CGIアプリケーションを呼び出したActionの名前

・action_path ('Kapt')　CGIアプリケーションの相対パス

・client_ip ('Kcip')　ユーザーのIPアドレス

・full_request ('Kfrq')　ユーザーのクライアントソフトからWebSTARへ送られてきたアクセス要求がヘッダーも含めてそのまま収められてる