HOLON

ホロン株式会社

HCGI&HACGI
CGI 開発支援解説

本マニュアル、およびマニュアルに記載されているソフトウェア（コンピュータプログラム）は、ホロン株式会社にすべての権利が帰属します。

ソフトウェアの正常な使用またはバックアップコピーを作成する場合を除き、ホロン株式会社の書面による同意なしには、本マニュアルやプログラムの一部または全部を問わず、複製を禁じています。
本ソフトウェアの仕様、およびマニュアルに記載されている事項は、将来予告なしに変更することがあります。
本マニュアルには、正確な情報を記載するように努めましたが、誤植や制作上の誤記がないことを保証するものではありません。
また、本ソフトウェアおよびマニュアルを運用した結果の影響については、一切責任を負いかねますのでご了承ください。

Appleの名称、ロゴマークは、米国その他の国で登録された米国アップルコンピュータ社の登録商標です。
その他、本マニュアルに記載されたソフトウェア、ハードウェア製品の名称およびロゴマークは、すべて開発および著作・販売会社等、関係各社の商標または登録商標であることを明記し、本文中での表記を省略させていただきます。

本マニュアルに記載された人物、団体名は、全て架空であり、実在いたしません。
類似の人物、団体名が存在する場合は、全くの偶然であり、本マニュアルには一切関係がありません。
また、本マニュアルに記載された他社商品名は、参考を目的としたものであり、それら製品の使用を強制または推奨するものではありません。

目次

I. 本文の内容

ホロン株式会社では、HiBase V4の複合サーバをリリースするに当たり、「HBCGI&HBACGI」を用意しました。「HBCGI&HBACGI」は、C/C++を利用する高速なCGIの開発手段で、「シンクロナスCGI(.CGI)」と「アシンクロナスCGI（.ACGI)」の両方の開発に利用することができます。本文はこの「HBCGI&HBACGI」の使い方を解説するものです。

※本文の内容を理解するには、以下の知識を必要とします。

・AppleEvent

・MacintoshでのCGI とAppleEventの関係

※参考文献

・Interapplication communication / InsideMacintosh: Addison Wesley

・WEB SITES by Jon Wiederspan & Chuck Shotton: Addison Wesley

II. Macintosh 環境での CGI

CGI（Common Gateway Interface）は、HTTPサーバの拡張をおこなう枯れた技術として広く普及しています。現在インタネットに公開されているホームページをもつサイトで、CGIのメカニズムを使っていないところはないでしょう。そしてCGIは普通なんらかの開発行為を必要とするため、CGI開発の成否がホームページ開発のキーとなっているのが現実の姿です。

Macintosh環境では、CGI の開発にAppleScriptを利用するのが一般的です。AppleScriptの習得の容易さや、汎用言語（C/C++やPascal等）を使ったAppleEvent対応のアプリケーションプログラム（スクリプタブルなアプリケーションプログラム）の開発の難しさがCGIの開発をAppleScriptに向かわせるわけですが、この場合よく問題になるのはCGIのパーフォーマンスです。AppleScriptは、プログラミングの熟練者でなくてもAppleEventに対応したプログラムを簡単に大量に作ることができる点で大いに評価できますが、スクリプト言語にありがちなパフォーマンスの悪さも特徴の一つです。Macintoshのネットワーク環境での処理能力の低さがよく話題になります。これを詳細に検討する時、MacOSのマルチタスク能力よりも、AppleScriptの利用頻度の高さがボトルネックになっているケースを多く見受けます。つまり、「スクリプタブルなアプリケーションプログラムは AppleScriptで」という安易さが、Macintoshのネットワーク環境での処理能力の低さを招いているケースです。ネットワークに関連するプログラムは例外無くパフォーマンスを常に要求されるわけですから、コンパイラ言語で実行コードを作り出すプログラムだけで構成するのが望ましく、CGIも例外ではありません。とくに、アクセス頻度の高いホームページをかかえるWEB SiteではAppleScriptでの安易なCGI の開発は論外と考えます。

III. Macintosh 用 HTTP サーバの一般知識の解説

AppleEvent と MacHTTP

一般的に、CGI の実装は、環境変数と標準入出力を利用してHTTP サーバと CGI の間のコミュニケーションを行うというUNIX環境での方法に従います。しかし、Macintoshはその生い立ちにおいて既にGUIであったため、環境変数と標準入出力といういわば原始的なユーザインタフェースであるTTYから生まれた概念がなく、UNIX環境でのようなCGI の実装方法が利用できません。

そのため、MacintoshでのCGIの実装は、MacOS7で導入されたユニークで強力なアプリケーション間通信のメカニズムであるAppleEventを利用することになります。

この「CGIの実装にAppleEventを用いるという方法」は、Macintosh環境で最初に稼動したHTTPサーバであるMacHTTPで採用されました。その後、WebSTARを代表とするMacintosh用のいろんなHTTPサーバで同じメカニズムが使われたことによりいまやMacintoshのデファクトスタンダードといえる方法になっています。

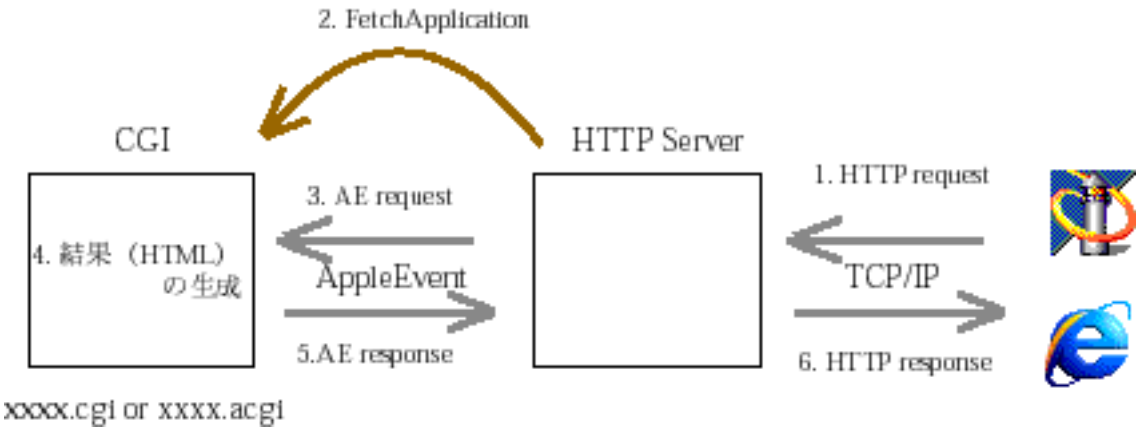
以上の経緯から、MacintoshでのCGIの開発は、AppleEvent対応のアプリケーションプログラムを開発することになります。これには、「AppleScriptで開発する」、「HyperCardで開発する」、「C/C++で開発する」の方法があります。これらのうち、最も一般的に使われるのは「AppleScriptで開発する」一番簡単な方法ですが、本文で話題にするのは最後の「C/C++で開発する」方法です。結論から言って一番厄介でもあります。その厄介さは、AppleEventの理解にはInsideMacintoshを始めとする膨大な解説書（千ページを超えると思われます）を読破しなければならず、そのAppleEventの理解無しに、「C/C++でAppleEvent対応のアプリケーションプログラムを開発する」方法が選択できないという点です。「AppleScriptで開発する」、「HyperCardで開発する」の両者は比較的簡単にAppleEvent対応のプログラムを開発できます。しかし、できあっがてくるアプリケーションプログラムを比較する時、

「AppleScriptで開発する」と「HyperCardで開発する」の両者は大きくパーフォーマンスで劣ることになり、やはり実用性を考えるならば「C/C++ で開発する」方法を選択するしかないと考えます。

処理の概略

ここで、WEB Browser、HTTP サーバ、CGI の三者の働きを、概観しておきましょう。

下の図をみてください。



WEB Browserから送られてきた要求が、HTTPサーバを中継して、CGIに伝わり、同じルートを逆にたどって結果が返されてきます。

WEB BrowserとHTTPサーバはTCP/IPのHTTP（プロトコル）でコミュニケーションが確立されており、HTTPサーバとCGI はAppleEvent（プロトコル）になります。

1.HTTP request

WEB Browser（Internet ExplorerやNetscapeCommunicatorなど）から、CGI関連の要求がHTTPサーバに到着します。例えば、FORM での入力データの送信などが代表的な例です。

2.FetchApplication

HTTPサーバはWEB Browserの要求を解析し、URLの中にxxxx.cgiもしくはxxxx.acgiのサフィックスを持つファイル名称を発見するとこれをCGI要求と判断します。そして目的のCGIが稼動しているかどうかを調べ、稼動していないなら目的のCGI を起動します。

3.AE request

次にHTTPサーバはAppleEventを組み立てます。この時、1 で送られてきたHTTP requestの全ての内容をAppleEventの中に梱包します。

少しAppleEventを解説しておきます。AppleEventというのはMacintoshのプログラム間通信のメカニズムに付けられた名前ですが、狭い意味ではプログラム間で送受信するパケットを指す言葉として用いられます。上の「AppleEventの中に梱包」という表現はまさに「プログラム間で送受信するパケット」の意味で使っています。

さて、このパケットですが非常に強力な梱包能力を持っています。例えばあらゆるデータ型を、また短いデーでも長いデータでも領域を無駄にすること無く梱包することができます。

HTTPサーバはこの強力な梱包機能を利用して、HTTP requestで要求されてきたあらゆる情報（付録-iに一覧を示します）をAppleEventに梱包しそれをAppleEvent Managerの提供するAESendを用いてCGI に送ります。

ここでもうひとつAppleEventで説明することがあります。AppleEventはCGIだけでなくいろんな用途に使われるプログラム間通信機能でもあります。そのため厳密な用途に関する規格が存在します。詳

しくは、AppleEvent Registoryというかなりなボリュームの文書に詳しくまとめられていますから興味のある方はそちらを参照してください。

この用途に関する規格ですが、イベントクラスとイベントコードによって識別されています。例えば、イベントクラス：'aevt'、イベントコード：'odoc' は文書ファイルをオープンする際に送られてくるAppleEventを示しています。

CGI に送るAppleEventもイベントクラス：'WWWｽ'、イベントコード：'sdoc'と決められています。この識別コードをAppleEventにさらに梱包しCGI に送るAppleEventが完成します。

4.CGI の仕事

HTTP サーバから送られてきた要求は、CGI が定義した AppleEventHandler に伝えられます。AppleEventHandlerは識別子（イベントクラスとイベントコード）ごとに定義され、プログラムの始めに AppleEventManager に登録しておきます。AppleEventManager はこの登録されているAppleEventHandlerに送られてきたAppleEvent を振り分けているのです。

HTTP サーバから送られてきたイベントクラス：'WWWｽ'、イベントコード：'sdoc'のAppleEventは、CGI が定義しているAppleEventHandlerに到着します。

CGI は、まず到着したAppleEventから付録-iに示しているパラメータを取り出します。そしてパラメータに応じて仕事をし（例えば、FORM での入力データが送られてきたならば、入力データをCGIの管理するファイルに書き込む)、結果をAppleEventに梱包しHTTPサーバに送り返します。

5.AE response

ここでいう結果の送り返しは全てHTMLページの形式でおこなわれます。つまり、CGIは仕事の終了報告（正常終了の報告だけでなくエラーの時も報告します）をHTMLで表現しそのページデータを再びAppleEventに梱包し（付録-2：レスポンスAppleEventの内容を参照)、それを送り返すことになります。

6.HTTP response

CGIから結果を返されたHTTPサーバは、返されてきたAppleEventからCGIの終了報告であるHTML ページを取り出し、何の変更も加えずWEB Browserに返します。

以上が、Macintosh環境でのWEB Siteを中心としたメカニズムです。

CGI .vs. ACGI

AppleEventの他にもうひとつ厄介な問題があります。それは、「CGI の実行をマルチスレッドでおこなうかどうか」の問題です。

一般的に、個々の要求に着目するならば、マルチスレッドで実行しないほうが（つまりシングルスレッド）CGIの処理能力は向上します。しかし、アクセス頻度の高いホームページを運用しているWEB Siteの場合、シングルスレッドのCGIは一つの要求を完了するまで次の要求を受け付けることができず、結果として何時アクセスしても要求を拒否されるホームページを抱えることになってしまいます。一方、マルチスレッドで実行するCGI は個々の処理にスレッドの切り替えが発生するため処理能力では劣るものの、CGI本来の処理を実行しながら要求の受付も同時平行でおこなうため、ホームページの訪問者を拒否することはありません。つまり、処理能力とターンアラウンド時間のトレードオフがシングルスレッドかマルチスレッドかの検討項目になります。

MacintoshのCGIにおいて、このシングルスレッドで要求を処理しているCGIを「シンクロナスCGI」、マルチスレッドで要求を処理しているCGI を「アシンクロナスCGI」と呼ぶことになっており、それぞれアプリケーションプログラム名称に「.CGI」「.ACGI」のサフィックスを付けて区別しています。勿論、「アシンクロナスCGI」の開発のほうがスレッドマネージャと言う難解な物をハンドリングする必要があるぶん難しくなります。

まとめ

以上、大雑把にMacintoshのHTTP関連の働きを眺めてきました。まとめると、MacintoshでのCGIの開発はAppleEvent対応のアプリケーションプログラムを開発することです。この開発をAppleScriptでおこなっているのが現在一般的です。しかし、実用性という点からはC/C++言語で開発するのが望ましく、その時の障害となるのが「AppleEventとスレッド処理にたいする理解」ということです。

ホロン株式会社の提供する「HBCGI&HBACGI」は、膨大な文書を必要とするAppleEventのハンドリングをC++のクラスライブラリにまとめ、CGIの開発者にとっていわばブラックボックスにしました。つまり、AppleEventの知識を必要とする部分をひとつにまとめ、それを開発者が意識しないでもAppleEvent対応のアプリケーションプログラムであるCGIの開発を可能にしています。また、「シンクロナスCGI(.CGI)」「アシンクロナスCGI（.ACGI）」の両方の開発環境が用意されており、AppleEventと同様、スレッドマネージャと言う難解な物も開発者から隠しています。

IV. HBCGI と HBACGI

「HBCGI&HBACGI」の利用方法

「HBCGI&HBACGI」には、CWP3で作ったプロジェクトを二つ用意しています。それぞれ、HBCGI.mcpは「シンクロナスCGI(.CGI)」を、HBACGI.mcpは「アシンクロナスCGI（.ACGI)」を開発するためのプロジェクトです。

両者は、以下の様なサイトの特徴に応じて使い分けることができよう設計しています。

HBCGI.mcp：

「シンクロナスCGI(.CGI)」の開発プロジェクトです。訪問者（クライアント）が重複しない、つまり、アクセス頻度の低いサイトのCGI を開発する場合に利用してください。

HBACGI.mcp：

「アシンクロナスCGI（.ACGI)」の開発環境です。訪問者（クライアント）が重複する、つまり、アクセス頻度の高いサイトのCGI を開発する場合に利用してください。

「シンクロナスCGI(.CGI)」、「アシンクロナスCGI（.ACGI)」を決定したならば：

1. 新しいフォルダを作り、そのフォルダにプロジェクトファイル、プロジェクトデータ、ソースコードをコピーしてください。

2. コピーしたプロジェクトファイルを開き、プロジェクトに含まれるHCGIUser.cppを目的の応じて変更してください。

3. HCGIUser.cppの変更の後、コンパイル／リンクしてCGI の完成です。

4. 完成したCGI をホームページライブラリの正しいフォルダにコピーしてください。

以上が、「HBCGI&HBACGI」を利用した CGI の開発スタイルです。

モジュール

「HBCGI&HBACGI」は、メインモジュール（CGIMain / ACGIMain）、HCGI モジュール（HCGI.h、HCGI.cp）、HCGIUserモジュール（HCGIUser.h、HCGIUser.cp）の三つからなっています。

メインモジュールは、AppleEventやマルチスレッドのMacintosh特有の処理を行う部分です。

HCGモジュールは、重要なクラスを二つ含んでいます。それぞれ、HCGIクラス、HCGIDataクラスです。

HCGIDataクラスはサーバから送られてきたAppleEventのキーワードパラメータを貯えるためのクラスです。HCGIクラスは、CGIの処理を行うためのアブストラクションクラスでHCGIDataを利用してCGIの目的を果たすためのフレームワークを提供します。

HCGIUserモジュールは、開発者（つまりあなたです）が変更するモジュールで、HCGIUserというクラスがHCGI を継承する形で定義されています。

HCGIUser の変更

早い話が、「HBCGI&HBACGI」環境では、HCGIUserモジュールを目的に応じて修正することがCGIの開発です。このHCGIUserモジュールを詳細に解説しておきます。

グローバル関数：

HCGI*　　MakeCGIObject();

解説：

CGIMain/ACGIMain から最初に呼び出されます。これはHCGIオブジェクト（HCGIの派生クラスから作り出すオブジェクトで、デフォルトは HCGIUser クラスとなっています）を生成して返します。HCGIUserクラスの名称を変更する場合、正しいクラス名称でオブジェクトを生成する修正をしてください。

HCGIUser クラスのメンバ関数：

HCGIUser();

解説：

コンストラクタ

HCGIUserの親クラスであるHCGIクラスは、CGI処理のフレームワークを提供していますが、その一つにログ機能があります。CGIはHTMLページを経てしかユーザとのコミュニケーションができません。そのため開発のデバッグが面倒になるのが普通です。ログ機能はこのデバッグを支援する機能としてHCGIのフレームワークに用意されており、適当なファイルを作り出してそこにログを書き込んでゆきます。

HCGIUserのコンストラクタでは、親クラスのコンストラクタ：HCGI(JBool iON = false, const JByte* iName = NULL); を呼び出す際に、ログ機能の有効（iON:true）／無効（iON:false)、ログファイル名称（iName）を指定します。

~ HCGIUser();

解説：

デストラクタ

ECode　　CGI_Init();

解説：

CGI の開始宣言。

CGI がメモリにロードされた時、最初に一度だけ呼ばれます。

例えば、CGI の参照するマスタファイル等のオープンをここでおこないます。

ECode　　CGI_Quit();

解説：

CGIの終了宣言。

CGIが終了する時、一度だけ呼ばれます。

例えば、CGI_Initでオープンしたファイルのクロースをここでおこないます。

ECode　　CGI_Proc(HCGIData& ioData);

解説：

CGIの処理要求。

HTTPサーバにクライアント（WEB Browser）からCGI要求が到着する度に呼び出されます。

HCGIDataから要求を読み出して、目的の処理を実行し、結果（HTMLで表現されたページデータ）を

HCGIDataに書き出してください。

HCGIDataに貯えられているデータの読み出し、および、HCGIDataへの結果の書き出しには「付録-iii：

HCGIDataクラスのメンバ関数」を利用してください。

ECode　　CGI_Periodic();

解説：

MacintoshのWaitNextEventでidle-eventが返される度に呼び出されます。

CGIにおいて、このCGI_Periodicを利用することは非常に特殊な場合と考えます。

void　CGI_Name(JByte* oName) const;

解説：

CGIがメモリにロードされた時、最初に一度だけ呼ばれます。HCGIUserの名称を返します。

V. 今後の予定

ホロン株式会社では、「HBCGI&HBACGI」をさらに機能強化することで、最終的にはHiBase複合サーバエクステンション（HBExtension）として位置付ける予定です。

HBExtensionは、HiBase 複合サーバのWEB ServerとDBServer の連結をおこなうCGI を開発することで実現することになります。HiBase複合サーバのWEB ServerとDBServerは同一タスク内で別スレッドとして実装されているため、この両者の連結は、Macintoshの弱点であるタスクのスイッチを行わずにスレッドの切り替えのみで WEB ServerとDB Serverが交互に稼動することになり、結果として大きなパフォーマンスの向上がはかれます。

HBExtension のリリース時期および提供に関するお知らせは、ホロン株式会社の WEB Site（www.hln.co.jp）で順次おこってゆきます。

「HBCGI&HBACGI」は、CWP3（CodeWarrior Professional rel-3）に対応した環境設定になっています。CWP3 以外の環境に対するCGI 環境も順次用意してゆきますので、詳しくはホロン株式会社のWEB Site（www.hln.co.jp）を参照してください。

付録 -i：リクエスト AppleEvent の内容

イベントクラス： WWWｽ

イベントコード： sdoc

キーワードパラメータ：

'----': パスアーギュメント（URLの'$'に続くパラメータ）

'kfor': サーチアーギュメント（URLの'?' に続くパラメータ）

'post': ポストアーギュメント（POSTで送られてくるコンテント）

'user': ユーザ名称（認証が必要な時）

'pass': パスワード（認証が必要な時）

'frmu': 現在未使用

'addr': クライアントのドメイン名称

'svnm': サーバのドメイン名称

'svpt': サーバのポート番号

'scnm': スクリプト名称（実行CGI へのパス）

'ctyp': コンテントタイプ（ポストアーギュメントのMIME タイプ）

'refr': 実行CGI の参照するURLページ

'Agnt': クライアントプログラム（WEB Browser）の名称

'Kact': アクション（常に"ACGI"）

'Kapt': アクションパス（実行CGI へのパス）

'meth': メソッド（HTTP要求のあったメソッドで、"GET"もしくは"POST"）

'Kcip': クライアントの TCP/IP アドレス

'Kfrq': リクエスト全体のコピー（HTTPサーバに送られてきたオリジナルデータ）

'Kcid': コネクション識別子（CGI に送る要求を識別するためのユニークな4 バイト整数）

付録 -ii：レスポンス AppleEvent の内容

キーワードパラメータ：

'----': HTML で表現されたデータ

付録 -iii：HCGIData クラスのメンバ関数

データの読み出し関数

以下のFetchXXX関数は、HCGIDataにあるそれぞれのデータへのポインタを戻り値として、また有効なデータの長さをlDataに返します。利用者（開発者）は返されてきたポインタを利用してデータをアクセスできますが、データを変更しないでください。

JByte*　　FetchPathArg　(JLong& lData) const;

：パスアーギュメントを返します。

JByte*　　FetchSrchArg　(JLong& lData) const;

：サーチアーギュメントを返します。

JByte*　　FetchPostArg　(JLong& lData) const;

：ポストアーギュメントを返します。

JByte*　　FetchUserName (JLong& lData) const;

：ユーザ名称を返します。

JByte*　　FetchPassWard (JLong& lData) const;

：パスワードを返します。

JByte*　　FetchCAddr　(JLong& lData) const;

：クライアントのドメイン名称を返します。

JByte*　　FetchSAddr　(JLong& lData) const;

：サーバのドメイン名称を返します。

JByte*　　FetchSPort　(JLong& lData) const;

：サーバのポート番号を返します。

JByte*　　FetchScript　(JLong& lData) const;

：スクリプト名称（実行CGI へのパス）を返します。

JByte*　　FetchCType　(JLong& lData) const;

：コンテントタイプ（ポストアーギュメントのMIME タイプ）を返します。

JByte*　　FetchRefer　(JLong& lData) const;

：実行CGI の参照するURLページを返します。

JByte*　　FetchAgent　(JLong& lData) const;

：クライアントプログラム（WEB Browser）の名称を返します。

JByte*　　FetchAct　　(JLong& lData) const;

：アクション（常に"ACGI"）を返します。

JByte*　　FetchActPath　(JLong& lData) const;

：アクションパス（実行CGI へのパス）を返します。

JByte*　　FetchMethod　(JLong& lData) const;

：メソッド（HTTP要求のあったメソッドで、"GET"もしくは"POST"）を返します。

JByte*　　FetchClientIP (JLong& lData) const;

：クライアントの TCP/IPアドレスを返します。

JByte*　　FetchFullReq（JLong& lData) const;

：リクエスト全体のコピー（HTTPサーバに送られてきたオリジナルデータ）を返します。

JLong　　FetchConnectID() const;

：コネクション識別子（4バイト整数）を返します。

データ（HTMLで作るページデータ）の設定関数

CGIは終了を報告するレポートをHTMLページの形で返す必要があります。レポートはHTTPヘダーとHTTP コンテンツから構成されますが、このうちHTTP コンテンツを作る際に利用できるのがAppendContensです。また、HTTPヘダーの生成にはMakeHeaderを利用します。

void　AppendContents(HFile* iFile);

：iFileで示すファイルをHTTPコンテンツに追加します。

void　AppendContents(HStream* iStream);

：iStreamで示すストリームをHTTPコンテンツに追加します。

void　AppendContents(const void* p, JLong l);

：pで示すメモリの内容をlバイト分、HTTP コンテンツに追加します。

void　AppendContents(const JByte* p);

：pで示す文字列（C型文字列）を、HTTPコンテンツに追加します。

void　AppendContents(JByte ch);

：chで示す一文字をHTTPコンテンツに追加します。

---

void MakeHeader(JWord err = 200, const JByte* pMsg = "OK");

：HTTPヘダーを生成します。