# MacPython でのOpen Scripting 拡張の利用について
# Using Open Scripting Extension from Python*

Jack Jansen
jack@cwi.nl http://www.cwi.nl/ jack
(日本語訳：山本　昇†)

Original: 27-Apr-98
日本語訳：January 17, 2000

## 1 Python

MacPython での OSA(Open Scripting Architecture) サポートはまだ完全にはほど遠い状態である。また、サポートされている機能についてもそう遠くない将来に変更される可能性がある。それにもかかわらず、あなたの Python プログラムから他のプログラムにちょっとした事をさせるのに十分なものが備わっている。

OSA support in Python is still far from complete, and what support there is is likely to change in the forseeable future. Still, there is already enough in place to allow you to do some nifty things to other programs from your python program.

実際の所、この文書で*"AppleScript"*と我々が言った場合それは、*"OSA"*を意味しているのである。*MacPython* の実装には *AppleScript* 特有のことは全く含まれていない。

*Actually, when we say "AppleScript" in this document we actually mean "the Open Scripting Architecture", there is nothing AppleScript-specific in the Python implementation.*

この例題では、私たちは、スクリプタブルなアプリケーションを見いだし、その AppleScript 辞書を抜き出してそこから Python のインタフェースモジュールを作成し、その Python モジュールを利用して、アプリケーションを制御するであろう。OSA の側面に集中するために Python アプリケーションのユーザー・インターフェースにはあまりかかわらないでおく。

In this example, we will look at a scriptable application, extract its "AppleScript Dictionary" and generate a Python interface module from that and use that module to control the application. Because we want to concentrate on the OSA details we don't bother with a real user-interface for our application.

これから私たちがスクリプトを使って制御しようとしているアプリケーションは、Eudora Light である。Eudora Light は QualComm が出しているフリーのメールプログラムである。このプログラムは、多機能なメールリーダで、QualComm は Eudora Light で間に合わない様な要求を満たすために対応する商用版のメールリーダプログラムを販売している。例題として作成するプログラムはこの Eudora に、待機中のメールの送信、サーバからのメールの受信あるいはプログラムの停止などを制御命令を送ることができる。

The application we are going to script is Eudora Light, a free mail program from QualComm. This is a very versatile mail-reader, and QualComm has an accompanying commercial version once your needs outgrow Eudora Light. Our program will tell Eudora to send queued mail, retrieve mail or quit.

*この文書は MacPython の配付ファイルに含まれる、applescript.html および cgi.html の日本語訳である。
†文部省高エネルギー加速器研究機構、加速器研究施設

## 2 Pythonインタフェース モジュールの作成 Creating the Python interface module

MacPythonと共に配付される標準ツールの中に、指定されたファイルの"AETE"あるいは"AEUT"リソース を調べ上げるツールが含まれている。AETE"あるいは"AEUT"リソースはAppleScript辞書の内部的な表現である。このツールはgensuitemodule.pyと名付けられて降り、MacPythonのMac:scriptsフォルダ中に置かれている。このツールを起動すると、入力ファイルを訪ねてくるので、私たちは入力ファイルとしてEudora Lightの実行プログラムを指定する。ツールgensuitemodule.pyは"AETE"リソースの解析を開始し、辞書中で見つけ出したそれぞれのAppleEventスイートについて出力となるPythonモジュールのファイル名を訪ねてくる。最初のモジュールについてはフォルダ名を変更することを忘れてはいけない。私たちはEudoraのフォルダの中をこれから作成するPythonインターフェースファイルでゴチャマゼにはしたくないだろうからである。もし、AppleScriptスイートを飛ばしたいならキャンセルボタンを押しなさい、処理は次のスイートの処理へと移っていく。Eudoraの場合、Required SuiteとStandard Suiteを作成する必要がない。なぜならこれらのスイートはあらかじめ用意されている標準のものと同一であるからである（しかもEudoraの実行プログラム中では空白となっている）。AppleScriptはこれらの空白のSuiteを「標準のスイートで置き換えなさい」と解釈する。gensuitemoduleはいまのところこの空白を理解しない。空のRequired_Suite.pyを作成することはその名前の正しいモジュールをこれから作ろうとしているPythonアプリケーションから隠してしまう。

There is a tool in the standard distribution that looks through a file for an 'AETE' or 'AEUT' resource, the internal representation of the AppleScript dictionary. This tool is called gensuitemodule.py, and lives in Mac:scripts. When we start it, it asks us for an input file and we point it to the Eudora Light executable. It starts parsing the AETE resource, and for each AppleEvent suite it finds it prompts us for the filename of the resulting python module. Remember to change folders for the first module, you don't want to clutter up the Eudora folder with your python interfaces. If you want to skip a suite you press cancel and the process continues with the next suite. In the case of Eudora, you do not want to generate the Required and Standard suites, because they are identical to the standard ones which are pregenerated (and empty in the eudora binary). AppleScript understands that an empty suite means "incorporate the whole standard suite by this name", gensuitemodule does not currently understand this. Creating the empty Required_Suite.py would hide the correct module of that name from our application.

gensuitemoduleがときおり"Where is enum 'xyz ' declared?"のようなメッセージとファイル選択ウィンドウを表示することがある。最初のときにはこのenum, クラスあるいはpropの名前を控えたうえで、ウィンドウをキャンセルしてしまいなさい。すべてのスイートのインタフェースモジュールを作成してしまってから、これらのモジュールあるいは標準のインタフェースモジュールの中で、このenum, クラスあるいはpropの名前を探しだして下さい。その名前を発見することが出来れば、もう一度gensuitemoduleを実行し、問題のウィンドウが表示されたところで、正しいファイルを指定して下さい。gensuitemoduleは参照を解決するためのimport文を作成します。

Gensuitemodule may ask you questions like "Where is enum 'xyz ' declared?". For the first time, cancel out of this dialog after taking down the enum (or class or prop) name. After you've created all the suites look for these codes, in the suites generated here and in the standard suites. If you've found them all run gensuitemodule again and point it to the right file for each declaration. Gensuitemodule will generate the imports to make the reference work.

> もし、Required_Suite.pyあるいはその他の標準モジュールの一つを再生成使用とするのなら、System Folder:Extensions:Scripting Additions:Dialects:English Dialectをさがしてみなければなりません。そこが、コアとなるAppleEvent辞書の収められているところだからです。また、Finder_Suiteインタフェイスを探しているなら、Finderを見に行ってもむだである（それはふるいSystem 7.0 scriptingスイートをもっている）。Finder Scripting Extensiosをさがしてご覧なさい。
>
> Time for a sidebar. If you want to re-create Required_Suite.py or one of the other standard modules you should look in System Folder:Extensions:Scripting Additions:Dialects:English Dialect, that is where the core AppleEvent dictionaries live. Also, if you are looking for the

Finder_Suite interface: don't look in the finder (it has an old System 7.0 scripting suite), look at the extension Finder Scripting Extension.

作成したばかりの、Eudora_Suite.py を見てみよう。AppleScript Editor も同時に開き、gensuitemodule がどのように辞書を解釈したかをみて見てもよいだろう。Eudora_Suite.py はいくつかのテンプレートで始まっている。その後にそれぞれの AppleScript 動詞に対応するメソッドをもつ大きなクラスが定義されている。さらに、小さなくらすと辞書構造体の初期化データが続いている。

Let's glance at the Eudora_Suite.py just created. You may want to open Script Editor alongside, and have a look at how it interprets the dictionary. EudoraSuite.py starts with some boilerplate, then a big class definition with methods for each AppleScript Verb, then some small class definitions and then some dictionary initializations.

Eudora_Suite class は生成されたコードの塊である。一つ一つの AppleScript の動詞に対応したメソッドが存在する。それぞれのメソッドはどのような引数を動詞が要求しているかを知っているし、Python1.3 で導入されたキーワード式の引数の指定を使い、Python プログラマに取り扱いやすい API を提供している。あなたは、それぞれのメソッドが aetools からのいくつかのルーチンを読んでいることに気がつくだろう。aetools は Lib:toolbox にある支援モジュールである。このモジュールは幾つかの AppleEvent ツールも含んでいる。後で、除いてみるといい、(もちろんのこと) このモジュールについての説明書は存在しない。

The Eudora_Suite class is the bulk of the code generated. For each verb it contains a method. Each method knows what arguments the verb expects, and it makes handy use of the keyword argument scheme introduced in Python 1.3 to present a palatable interface to the python programmer. You will see that each method calls some routines from aetools, an auxiliary module living in Lib:toolbox which contains some other nifty AppleEvent tools as well. Have a look at it sometime, there is (of course) no documentation yet.

その他にそれぞれのメソッドは self.send メソッドをよびだしているが、そのようなメソッドは定義されていないことに気づくだろう。跡で見るように、あなたはこのメソッドをサブクラスあるいは複数の継承を、用いて用意しなくてはならない。

The other thing you notice is that each method calls self.send, but no such method is defined. You will have to provide it by subclassing or multiple inheritance, as we shall see later.

大きなクラスの後に、幾つかの小さなクラス定義がついづいている。これらのクラス宣言は (appleevent) クラスと属性のためのものである。これらのクラスをつかってオブジェクト ID をつくることが出来る。この ID は動詞に引数として渡すことが出来る。たとえば、受信メールボックスの最初のメッセージの送り手の名前を得るためには、mailbox("inbox").message(1).sender を使えばよい。これを sender(message(1, mailbox("inbox"))) として指定することも可能である。これらのクラスがベースクラスから正しく継承されていないために、この形が必要となる場合がある。であるから別のスイートからのクラスあるいは属性を使う必要があるかもしれない。

After the big class we get a number of little class declarations. These declarations are for the (appleevent) classes and properties in the suite. They allow you to create object IDs, which can then be passed to the verbs. For instance, to get the name of the sender of the first message in mailbox inbox you would use mailbox("inbox").message(1).sender. It is also possible to specify this as sender(message(1, mailbox("inbox"))), which is sometimes needed because these classes don't inherit correctly from baseclasses, so you may have to use a class or property from another suite.

aetools の標準オブジェクトについてのふるいオブジェクト指定がある。これらのオブジェクト指定には aetools.Word(10, aetools.Document(1)) の形式を使う。この指定に対応する AppleScript の用語は、"word 10 of the first document" となる。標準のオブジェクト指定子以外のものを作り出す必要が生じた場合には、ここで述べた二つのモジュールをあなたのスイートモジュールの最後につけられたコメントとともに比較為てみなさい。

There are also some older object specifiers for standard objects in aetools. You use these in the form aetools.Word(10, aetools.Document(1)) where the corresponding AppleScript terminology would be word 10 of the first document. Examine the two modules mentioned above along with the comments at the end of your suite module if you need to create more than the standard object specifiers.

次に、enumeration 辞書を得た。この辞書によって動詞の引数に英語の名前を使うことができる。これによって 4 文字のタイプコードの煩わされることが無くなる。あなたは次のように書くことが出来る。
Next we get the enumeration dictionaries, which allow you to pass english names as arguments to verbs, so you don't have to bother with the 4-letter type code. So, you can say

```
eudora.notice(occurrence="mail_arrives")
```

以下のより意味の分かりにくい形式を使わなくともよい。
instead of the rather more cryptic

```
eudora.notice(occurrence="wArv")
```

最後に私たちはモジュールの"table of contents"を得た。すべてのクラスとコードをリストしたものである。これらは gensuitemodule 自身で使用されている。
Finally, we get the "table of contents" of the module, listing all classes and such by code, which is used by gensuitemodule.

# 3 Python スイートモジュールの使い方 Using a Python suite module

此処までで、私たちは私たちのアプリケーションで使うことができるスイートモジュールを手にした。このアプリケーションは Eudora_Suite と aetools モジュールから TalkTo クラスを継承することでつくる。TalkTo クラスは基本的にスーツクラスのメソッドで使われる send メソッドのためのコンテナである。
Now that we have created the suite module we can use it in an application. We do this by creating a class that inherits Eudora_Suite and the TalkTo class from aetools. The TalkTo class is basically a container for the send method used by the methods from the suite classes.

実際には、我々のクラスは Required_Suite も継承している。このクラスの提供するメソッド, quit() を必要とするからである。Gensuitemodule は必要なすべての情報にアクセス可能なのであるから、この継承されたクラスを作り上げることも出来たはずである。しかしながら、現状では Gensuitemodule はそうは為てくれない。結局のところ、我々のプログラムの基本部分は次の様なものになる。
Actually, our class will also inherit Required_Suite, because we also need functionality from that suite: the quit command. Gensuitemodule could have created this completely derived class for us, since it has access to all information needed to build the class but unfortunately it does not do so at the moment. All in all, the heart of our program looks like this:

```
import Eudora_Suite, Required_Suite, aetools

class Eudora(Eudora_Suite.Eudora_Suite, Required_Suite.Required_Suite, \
                        aetools.TalkTo):
        pass
```

そう、我々のクラスの本体は pass である。すべての機能はベースクラスに既に用意されている、必要なことはすべてを正しくつなぎあわせる事だけである。
Yes, our class body is pass, all functionality is already provided by the base classes, the only thing we have to do is glue it together in the right way.

アプリーケーションのソースファイル testeudora.py を見てみると、ファイルは幾つかの import 文で始まっている。その後には我々の中心となるクラスの定義と Eudora のシグネチャを与える定数が含まれている。
Looking at the sourcefile testeudora.py we see that it starts with some imports. Then we get the class definition for our main object and a constant giving the signature of Eudora.

これについては少し説明が必要である。どのプログラムにたいして AppleScript を送ろうと為ているかを指定するには多くの方法がある。しかしもっとも簡単な (少なくとも Python からは) 方法はクリエータシグネチャである。アプリケーション名が使えればもっとよい、しかし Python は現在の所ファインダーのデータベース (このデータベースでアプリケーション名からシグネチャへの変換をおこなう) から情報を引きだすためのモジュールを持っていない。その他の方法として考えられる program-to-program ツールボックスの ChooseApplication はやはり Python ではまだサポートされていない。

This, again, needs a little explanation. There are various ways to describe to AppleScript which program we want to talk to, but the easiest one to use (from Python, at least) is creator signature. Application name would be much nicer, but Python currently does not have a module that interfaces to the Finder database (which would allow us to map names to signatures). The other alternative, ChooseApplication from the program-to-program toolbox, is also not available from Python at the moment.

もしアプリケーションのクリエータを指定すれば、オプションの start パラメタをさらに指定することが出来る。このパラメタでアプリケーションがまだ起動されていなければ、これを自動的に起動させることが出来る。

If you specify the application by creator you can specify an optional start parameter, which will cause the application to be started if it is not running.

メインプログラムは驚くほど単純である。(シグネチャを引数としてあたえて) Eudora と交信するためのオブジェクトを作成し、ユーザに何をすべきかを尋ね、そして適当な Eudora オブジェクトのメソッドを呼び出す。AppleScript で使われるのと同じ名前を キーワード引数を使うこともプログラムを簡単に為ている。

The main program itself is a wonder of simplicity. We create the object that talks to Eudora (passing the signature as argument), ask the user what she wants and call the appropriate method of the talker object. The use of keyword arguments with the same names as used by AppleScript make passing the parameters a breeze.

例外処理については幾つかのコメントが必要である。AppleScript は基本的に接続なしの RPC プロトコルであるから、Eudora オブジェクトを作成為たときには何も起こりません。従って、アプリケーションが既に起動しているのでは無い場合、最初のコマンドを送りだすまでそのことは Python プログラムからはわからない。その他にも AppleScript から返されるエラーについて注意が必要である。 OSErr-type のエラーに対しては、MacOS.Error を返す、サーバが生成するエラーについては aetools.Error が生成される。

The exception handling does need a few comments, though. Since AppleScript is basically a connectionless RPC protocol nothing happens when we create to talker object. Hence, if the destination application is not running we will not notice until we send our first command. There is another thing to note about errors returned by AppleScript calls: MacOS.Error is raised for all of the errors that are known to be OSErr-type errors, server generated errors raise aetools.Error.

# 4 スクリプティング機能拡張 Scripting Additions

もし通常 OSAXen と呼ばれている スクリプティング機能拡張モジュールを Python プログラムから使おうとするなら、アプリケーションと同じ方法によってこれらのモジュールを使うことが出来る。つまり gensuitemodule を OSAX(通常 System Folder:Extensions:Scripting Additions あるいは類似のフォルダにおかれている) に対して動作させ、gensuitemodule によって生成されたクラスと aetools.TalkTo を継承したクラスを定義し、その実体を作り出す。実体を作り出すさいに使われるアプリケーションシグネチャは'MACS' である。

If you want to use any of the scripting additions (or OSAXen, in everyday speech) from a Python program you can use the same method as for applications, i.e. run gensuitemodule on the OSAX (commonly found in System Folder:Extensions:Scripting Additions or something similar), define a class which inherits the generated class and aetools.TalkTo and instantiate it. The application signature to use is 'MACS'.

gensuitemodule を OSAX に適用する際に注意すべき事が二点ある。すべての OSAX は System_Object_Suite を定義使用としているようである。そして、多くの拡張は複数の方言に対するコマンドセットを持っている。この結果として gensuitemodule で作成された Python モジュール中での名前の衝

突に気をつけねばならない。そして適切な方言を選択されたことを確認する必要がある（英語ではない方言のなかには genstuitemodule をして不正な Python コードを生成させてしまうものがある)。

There are two minor points to watch out for when using gensuitemodule on OSAXen: they appear all to define the class System_Object_Suite, and a lot of them have the command set in multiple dialects. You have to watch out for name conflicts, so, and make sure you select a reasonable dialect (some of the non-english dialects cause gensuitemodule to generate incorrect Python code).

これで私たちの簡単な例題はおしまいです。最後に再び現状の Python でのスクリプティングサポートは不完全なものであることまた、そしてまた AppleEvent を使う方法の詳細は遠くない将来に変更されるであろうこと 、を強調しておきたい。これはこの文書に現れたすべての ideosyncracies を修正するだけでなく、既存のプログラムを実行不能なものにする。なぜなら現在のスイートの構成は幾つかの問題を解決するために変更されねばならないからである。もしあなたがいま AppleEvents をためしてみたいなら。やるべし！

That concludes our simple example. Again, let me emphasize that scripting support in Python is not very complete at the moment, and the details of how to use AppleEvents will definitely change in the near future. This will not only fix all the ideosyncracies noted in this document but also break existing programs, since the current suite organization will have to change to fix some of the problems. Still, if you want to experiment with AppleEvents right now: go ahead!

# A 付録：Python による CGI の作成 Using python to create CGI scripts

ここでは、N etPresenz そしておそらくは他の Mac 用 HTTP サーバでもはたらく Python CGI スクリプトの作成法を説明する。Mac においては CGI スクリプトは AppleEvent サーバであるので、一般的な AppleEvent サーバープログラムおよびアプレットのデバグについても必要最小限ではあるが学んでいく。

In this document we will (eventually) explain how to create Python CGI scripts for use with NetPresenz and probably other Mac-based HTTP servers too. Since CGI scripts are AppleEvent servers on the mac we will also learn a little about general AppleEvent server programming and about applet debugging.

現在の設定は極めて原始的なものであることに注意して欲しい。だからこの文書の情報をあなたの戦略的な製品の基礎におくことは賢明なこととは言えない。それよりも此処のコードで遊んでみて、Pythonmac-sig に参加して欲しい。この SIG では現実的な MacCGI のフレームワークについて議論をしたいと考えている。できることなら CGI スクリプトを Unix や他のプラットフォームとのあいだでポータブルにするようなフレームワークを作り上げたい。

Note that the current setup is very preliminary, and hence itis probably not wise to base your strategic products on the information in this document:-) In stead, play with the code here and join the pythonmac-sig, where we I would like to have a discussion on a real design for a Mac CGI framework (preferrably something that will make CGI scripts portable to unix and other platforms).

## A.1 AppleEvent サーバ AppleEvent servers

AppleEvent クライアントはサーバに比べて、書きやすく理解しやすいので、まずは「Open Scripting clients in Python 」を読むほうがよいだろう。

Since AppleEvent clients are easier to write and understand than servers you should probably read the section on Open Scripting clients in Python first.

次に、AE サーバ　フレームワーク、MiniAEFrame.py を見てみよう。このファイルは二つのクラス、MiniApplication および AEServer 、を含んでいる。MiniApplication は ウィンドウ等を必要としないアプリケーションに向いた、FrameWork.Application の簡略版である。AEServer は AppleEvent のコード化の一部を行うグルールーチンである。installeaehandler に AppleEvent のクラスと 四文字の event ID およびそのエベントを処理するコールバックルーチンを指定して呼びエベントにたいするコールバックを定義する。AppleEvent が生起された時、このコールバックが正しい引数とともに呼びだされる。いまのところ、引数

の名前は Open Scripting で内部的に使われている四文字の値である。将来的には gensuitemodule で生成されたスイートモジュールとど同様に変換の機構が用意されるだろう。

Next, let us have a look at the AE Server framework, MiniAEFrame.py. This file contains two classes, MiniApplication and AEServer. MiniApplication is a tiny replacement for FrameWork.Application, suitable if your application does not need windows and such. AEServer is a bit of glue that does part of the appleevent decoding for you. You call installaehandler passing it the class and id (4-char strings) of the event you have a handler for and the handler callback routine. When the appleevent occurs your callback is called with the right arguments. For now, your argument names are the 4-char values used internally by Open Scripting, eventually there will be a translation similar to what the generated OSA client suites provide.

AEServer はそのファイルをダブル・クリックすることで起動出来る。起動された AEServer は標準の run/open/print/quit の OSA コマンドに反応する。もし AEServer が通常の Python スクリプトとして動作しているさいにあなたがファイルを Python インタプリタにドラッグするとスクリプトは受け取った Apple Event を表示する。

You can test AEServer by double-clicking it. It will react to the standard run/open/print/quit OSA commands. If it is running as a normal python script and you drag a file onto the interpreter the script will tell you what event it got.

## A.2 最小の CGI スクリプト<br>A Minimal CGI script

CGI スクリプトをためすにはまず http サーバが必要である。私 (Jack Jansen) は Peter Lewis のシェアウェア、NetPresenz を使っている (試用以上のことをしようとするなら、料金を払うのを忘れないで！)。あなたの http サーバをインストールし、そのサーバが静的なテキストベースのドキュメントをそのサーバがとりあつかうことが出来ることを確認しよう[1]。

To try a CGI script you will first need a http server. I have used the shareware NetPresenz by Peter Lewis (don't forget to pay if you give it more than a test run!). Install your http server, and make sure that it can serve textual documents.

つぎに我々の CGI スクリプトの例を見てみよう。CGI スクリプトはアプリケーションであるひつようがある。そのため exapmle2 で説明されるように、Applet を作りだす必要がある。我々のアプレットコード、cgitest.cgi.py は execfile からなる必要最小限の小さなものである。これは実際のコードは realcgitest.py である。コードを変えるたびに mkapplet を操作する必要をなくすためである。コードが満足にいくものであれば realcgitest.py を cgitest.cgi.py に名付けなおしておく。

Next, let us have a look at our example CGI scripts. CGI scripts have to be applications, so we will have to make an applet as explained in example 2. Our applet code, cgitest.cgi.py is a rather minimal execfile statement. The reason for this is debugging: the real code is in realcgitest.py, and this way you do not have to run mkapplet again every time you change the code. Rename realcgitest.py to cgitest.cgi.py once you are satisfied that it works.

リソースファイルは、一つの例外を除いては、それほど特別な事はない。我々は AppleEvent の処理を自分で行おうとしているのであるから、Python の初期化コードが argc と argv を作り出すことは望ましくない、なぜならこの処理が我々が処理しようとしている AppleEvent まで処理してしまうかもしれないからである。このために我々は argv の初期化を禁止する Popt リソースを付け加えた。このリソースを作りあげる簡単な方法は, .rsrc ファイル (あるいは完成した Applet) を EditPythonPrefs アプリケーションにドロップし、"no argv processing" オプションを設定するだけである。

The resource file is not very special, with one exception: since we want to do our own appleevent handling we don't want the Python initialization code to create argc and argv for use, since this might gobble up any appleevents we are interested in. For this reason we have included a 'Popt' resource that disables the argv initialization. An easy way to create this resource is to drop the .rsrc file (or the finished applet, if you like) onto EditPythonPrefs and set the "no argv processing" option.

[1]訳者注:訳者は StarNine 社の WebStar を使っている。このサンプル CGI は WebStar でも同じく動作する。

コードそれ自身も実際そう複雑なものではない。我々は,"open applicatio"および"quit"そして、CGI実行時に送られる "WWW\275"/"sdoc" エベントに対する AppleEvent の処理ルーチンを実装した (quit エベントの処理コードは MiniAEFrame 中のテストコードから拝借した) 。cgi 処理ルーチンは HTML 中の CGI 引数を清書して出力する。じっさいの引数に就いては、次の URL で詳しく説明されている。

`http://www.biap.com/datapig/mrwheat/cgi\_params.html.`

The code itself is actually not too complicated either. We install handlers for "open application" and "quit" (stolen from the test code in MiniAEFrame) and the "WWW\275"/"sdoc" event, the event sent on CGI execution. The cgi handler pretty-prints the CGI arguments in HTML and returns the whole string that is to be passed to the client. The actual parameters passed are explained in

`http://www.biap.com/datapig/mrwheat/cgi\_params.html.`

スクリプトをテストするために cgitest.cgi.py を mkapplet の上にドロップする。でき上がった cgitest.cgi を NetPresenz が見つけ出すことの出来る所にコピーし、WebBrowser からこのファイルを指すようにする。

To test the script drop cgitest.cgi.py onto mkapplet, move the resulting cgitest.cgi to somewhere where it is reachable by NetPresenz, and point your web browser towards it.