upgrade your music!
Premium Powered Bookshelf Speakers • Starting at $349 per pair • ☎ 877.853.4447
"There are no other speakers in this price range that come close" -Mac Observer
audioengine 5
www.audioengineusa.com

CGIs in Lisp

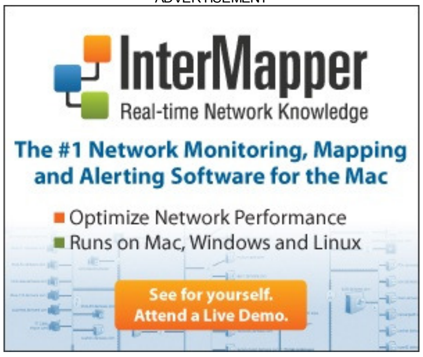
Volume Number: 12
Issue Number: 7
Column Tag: Webtech™

CGIs in Lisp

A Web dictionary server

By Mathieu Lafourcade and Gilles Sérasset

Note: Source code files accompanying article are located on MacTech CD-ROM or source code disks.


InterMapper
Real-time Network Knowledge
The #1 Network Monitoring, Mapping and Alerting Software for the Mac
Optimize Network Performance
Runs on Mac, Windows and Linux
See for yourself. Attend a Live Demo.

upgrade your music!
audioengine 2
www.audioengineusa.com
Premium Powered Desktop Speakers
$199 per pair
"These are the best speakers for your desktop, computer, or media player"
-Connect Reviews

Introduction

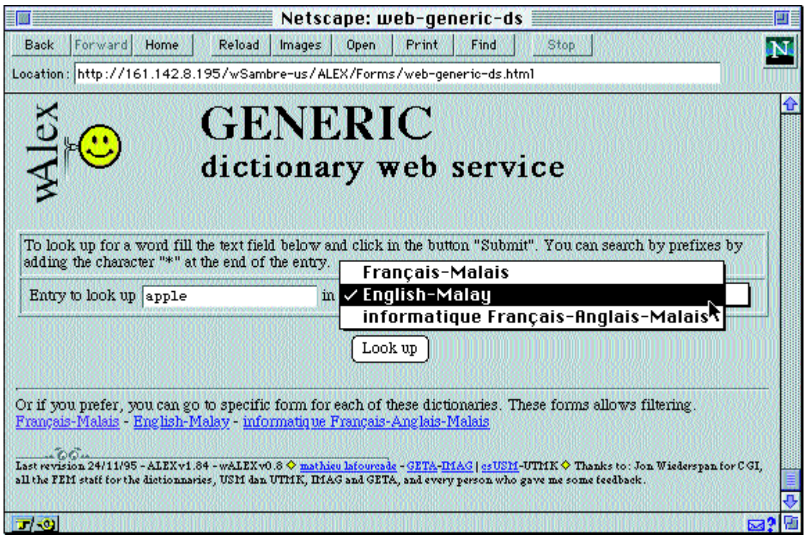
A paper dictionary is fine for reading comfort, versatility, etc., but updating of the data is not possible (apart from buying a new dictionary!), and the amount of data may be smaller than an electronic version. A resident electronic dictionary allows cut and paste, and updates are generally possible; but, like the paper dictionary, you must carry it along (in your hard disk). The Web-based dictionary doesn't take any space on your hard disk; the memory footprint is that of your Web browser. On the other hand you do rely on the availability of the network and/or server.

This article describes how a Web server offering multilingual dictionary services was set up by integrating three components: MacHTTP, AppleScript, and Macintosh Common Lisp (MCL). The dictionary server is constructed from a dictionary application, ALEX, written with MCL. This application can be remotely (and facelessly if needed) controlled with AppleScript. Given this design, the connection with MacHTTP is painless. The result is a very cost-effective way of providing a large linguistic resource on the Web.

Our developmental approach was to come up with something functional as fast as possible and to enhance the solution afterward. We had to compromise between functionality delivered to the user, performance, and the technical implications for the developer.

We first describe a typical user session; you can try this out yourself by pointing to the URL from your own browser (you can see the URL in Figure 1). The general architecture of the server is then presented, followed by some technical details about the three components.

Note: The included sources have often been trimmed to keep this article as short as possible - for instance, the error handling is generally not included.



Location: http://161.142.8.195/wSambre-us/ALEX/Forms/web-generic-ds.html

GENERIC
dictionary web service

To look up for a word fill the text field below and click in the button "Submit". You can search by prefixes by adding the character "*" at the end of the entry.

Entry to look up: apple in [Français-Malais / ✓ English-Malay / informatique Français-Anglais-Malais]

Look up

Or if you prefer, you can go to specific form for each of these dictionaries. These forms allows filtering.
Français-Malais - English-Malay - informatique Français-Anglais-Malais

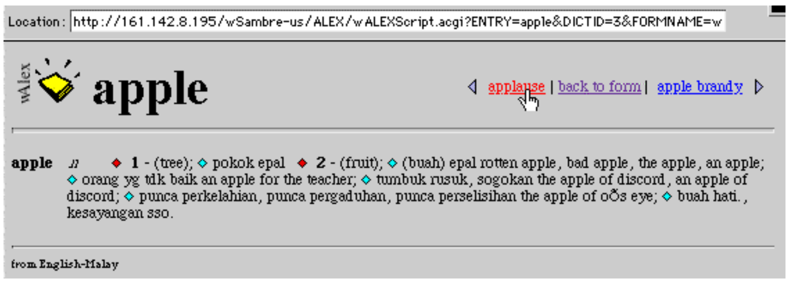
Last revision 24/11/95 - ALEXv1.84 - wALEXv0.8 ◆ mathieu lafourcade - GETA-IMAG | csUSM-UTMK ◆ Thanks to: Jon Wiederspan for CGI, all the FEM staff for the dictionnaries, USM dan UTMK, IMAG and GETA, and every person who gave me some feedback.

Figure 1. Initial Web page of the server

A Typical User Session

The Web page displays a form (Figure 1). The user can choose a dictionary, enter a word to look up, and click on the "Look up" button. It is possible to terminate the candidate word with an asterisk; in this case the first entry which starts with the text entered is returned.

A new page appears displaying the information associated with the word entered (Figure 2). If the entry is not found, a (friendly) error message is given. Beside the title (corresponding to the word looked up), the user has access to links to the previous word (if any), to the form (just like pressing the "Back" button of the Web client), or to the next word (if any). This way, the user can walk through the entire dictionary.



Location: http://161.142.8.195/wSambre-us/ALEX/wALEXScript.acgi?ENTRY=apple&DICTID=3&FORMNAME=w

wAlex apple

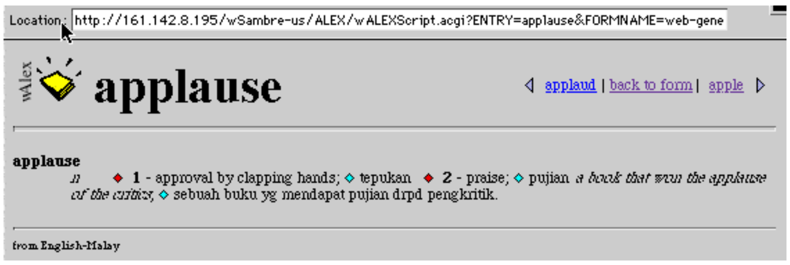
◁ applause | back to form | apple brandy ▷

apple *n* ◆ **1** - (tree); ◇ pokok epal ◆ **2** - (fruit); ◇ (buah) epal rotten apple, bad apple, the apple, an apple; ◇ orang yg tdk baik an apple for the teacher; ◇ tumbuk rusuk, sogokan the apple of discord, an apple of discord; ◇ punca perkelahian, punca pergaduhan, punca perselisihan the apple of oÕs eye; ◇ buah hati., kesayangan sso.

from English-Malay

Figure 2. The data displayed for the entry "apple";
the source of this page is given in Listing 10

For example, the user can click on the previous entry "applause" and obtain the corresponding page (Figure 3). The user can continue to scan the dictionary or can return to the form and choose an other entry or dictionary.



Location: http://161.142.8.195/wSambre-us/ALEX/wALEXScript.acgi?ENTRY=applause&FORMNAME=web-gene

wAlex applause

◁ applaud | back to form | apple ▷

applause
n ◆ **1** - approval by clapping hands; ◇ tepukan ◆ **2** - praise; ◇ pujian *a book that won the applause of the critics*; ◇ sebuah buku yg mendapat pujian drpd pengkritik.

from English-Malay

Figure 3. The web page for the entry "applause"

The HTML Form

The form used as an interface between user and server, the starting point of a session, is a plain HTML page. The link between this page and the application handling the action is made in the form declaration. The standard POST method is used for sending the message from the server to the application, wAlex.acgi.

We assume you are familiar with HTML (if not, check out the numerous informative pages available on the Web). The <FORM> tag specifies a form, which may include various items, such as text input (text field) and select input (menu). Notice that many less common characters (in English at least) are represented a specific way (for example the "ç" as "&ccedil"), and this is the beginning of big problems when you decide to deal with other languages, like French for instance.

Listing 1:
HTML Source of Figure 1

```
The dictionary lookup form

All the cosmetic and informative stuff at the
beginning of the page.
<TITLE>web-generic-ds</TITLE>
<TABLE CELLPADDING=3 WIDTH="100%">
<T>R
  <TD ALIGN=LEFT>
   <img ALIGN=middle src="http://161.142.8.195
/wSambre-us/ALEX/Gifs/wAlex-vert.GIF"
WIDTH=25>
   <img ALIGN=middle src="http://161.142.8.195
/wSambre-us/ALEX/Gifs/wAlexVLogo.GIF">
  </TD>
  <TD ALIGN=LEFT>
<H1><FONT SIZE=+7>GENERIC</FONT><B>R dictionary
web service</H1>
 </TD>
</T>R
</TABLE>

The form definition starts here.
<FORM ACTION="http://161.142.8.195/wSambre-us/ALEX
/wAlex.acgi" METHOD=Post>

<TABLE border UNITS=PIXELS CELLPADDING=2
```

```
WIDTH="100%">
<T>R
  <TD>
  <P>To look up for a word fill the text field
below and click in the
button "Submit".
You can search by prefixes by adding the
character "*" at the end of
the entry.<B>R
  </TD></T>R
<T>R
  <TD ALIGN=CENTE>REntry to look up <INPUT
TYPE="text" NAME="Entry" VALUE=""
MAXLENGTH=150>
in
<SELECT NAME="DictID">
<OPTION VALUE="2">Fran&ccedil;ais-Malais
<OPTION VALUE="3">English-Malay
<OPTION VALUE="1">informatique Fran&ccedil;ais-
Anglais-Malais
</SELECT>
  </TD></T>R
</TABLE>

The Submit button is defined here.
<P><CENTE>R<INPUT TYPE="Submit" VALUE="Look
up"></CENTE>R

One hidden item giving the name of the form we
are in.
<INPUT TYPE=HIDDEN NAME="FormName" VALUE="web-
generic-ds.html">
The form definition ends here.
</FORM>

Some links to specific dictionary forms are
presented here.
<H>R
Or if you prefer, you can go to specific form for
each of these dictionaries.

These forms allows filtering. <B>R
<A HREF="http://161.142.8.195/wSambre-us/ALEX
/Forms/web-FEM-ds.html">Fran&ccedil;
ais-Malais</A>
-
<A HREF="http://161.142.8.195/wSambre-us/ALEX
/Forms/web-EM-ds.html">English-Malay</A> -
<A HREF="http://161.142.8.195/wSambre-us/ALEX
/Forms/web-FEMterm-ds.html">informatique
Fran&ccedil;ais-Anglais-Malais</A>


The signature ends the page.
<P><IMG ALIGN = MIDDLE src="http://161.142.8.195
/wSambre-us/GIFgeneral/line.GIF"></br>
<FONT SIZE = 1> Last revision 24/11/95 -
ALEXv1.84 - wALEXv0.8  <IMG
SRC="http://161.142.8.195/wSambre-us/ALEX
/Gifs/yellow-diamond.GIF">
<A HREF="http://161.142.8.195/wSambre-us/STAFF
/ML-us.html">mathieu lafourcade</A> -
<A HREF="http://imag.fr
/IMAG/GETA.html">GETA</A>-<A HREF="http://imag.fr
/">IMAG</A> |
<A HREF="http://www.cs.usm.my
/">csUSM</A>-<A>UTMK</A> <IMG
SRC="http://161.142.8.195/wSambre-us/ALEX
/Gifs/yellow-diamond.GIF">

 Thanks to: Jon Wiederspan for CGI, all the FEM
staff
for the dictionaries, USM dan UTMK, IMAG and
GETA, and every person who
gave me some feedback.
</FONT><P>
```

Opening Pandora’s Box

How does the information get from the browser to the server to the Dictionary Service? In form-based queries, there are two methods (POST and GET) for submitting the data to the gateway script. We use only POST, as recommended by NCSA. POST (unlike GET) is not limited in the amount of information you can pass with a form.

A gateway script is needed on the server side to decode these arguments, process the query, and return an answer. In our case we have MacHTTP (the server application) and our MCL-based application, ALEX (which serves dictionary data). We have tested two different architectures for connecting MacHTTP to ALEX.

In the first solution, we transformed ALEX, our MCL appplication, itself into an acgi script. The second solution uses a small AppleScript applet to make the glue between MacHTTP and ALEX. Both methods have their own advantages and drawbacks.

ALEX (MCL) As a CGI Application

In the first solution MacHTTP interacts directly with ALEX, our MCL application (Figure 4). To make MacHTTP believe it is an acgi script, the name of the application must end with ".acgi". When MacHTTP receives a query from a WWW client requesting a file whose suffix mapping is ".acgi" (an executable application), MacHTTP will attempt to communicate with it via a custom Apple event of class 'WWW ' and ID 'sdoc'. We want the received request to be handled by ALEX; hence, our application has to listen (at least) to this particular Apple event. For this solution, we also have to change the name of the form action in the HTML file to wAlex.acgi (which is the name of our MCL application).

MCL directly supports the four required Apple events sent by the Finder: Open Application, Open Documents, Print Documents, and Quit Application. In addition, MCL provides a very easy way to support other Apple events by writing and installing new Apple event handlers.

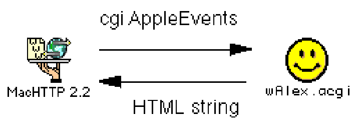


Figure 4. MacHTTP communicates directly with ALEX transformed into an ACGI application

Handling Apple events

An Apple event handler is a generic function specialized on the application class. Hence it is possible to customize the behaviour of the predefined Apple event handlers by specializing the application class. This is not necessary for our purpose. However, we have to write a new handler for the CGI event.

As MCL bypasses the Apple Event Manager's dispatch routine, you don't have to bother with how your function is called by the Apple Event Manager. Lisp takes care of dispatching and run-time error checking. If no error occurs, your handler should simply return in a normal way. The returned value is ignored. If an error occurs, your handler should signal an appleevent-error condition, with an error number and an optional error string.

An Apple event handler has four arguments: the *application* (always the value of the global variable *application*), the *Apple event* (you generally don't have to look at the record structure directly), the *reply* (another Apple event record, provided by the Apple Event Manager) and an optional handler *reference constant* (any Lisp object that can be used e.g. to distinguish two different installations of the same handler; this will be ignored for our purpose). Look at Listing 4.

Decoding arguments

MacHTTP provides a lot of information (client address, user name and password, arguments of the request, etc.). MacHTTP's documentation will give you the complete set of parameters and their four-byte codes.

To get the value associated with each parameter, you have to use the ae-get-parameter-char function. You specify the AppleEvent object and the parameter, and the function returns a string containing the required value.

In our case, we assume that the client issues a POST method (the GET method case can be easily deduced from this code). Hence, the arguments we need are the post arguments, accessible via the 'post' four-byte code.

The ae-get-parameter-char function returns a string consisting of all the variables of the form, and their values appended together. This format is not directly usable; hence, we will transform it into a list of variable-value pairs. For instance, the argument string:

```
"ENTRY=apple&FORMNAME=web-generic-ds&DICTID=3"
```

will be transformed into:

```
(:DICTID "3"
 :FORMNAME "web-generic-ds"
 :ENTRY "apple")
```

To transform this kind of string, we use one of our publicly available packages called geta-strings (ftp://ftp.digitool.com//pub/MCL/contrib/geta-strings-1.3.sea.hqx).

Listing 2:
wAlex.acgi 1/3 (MCL Application)

```
Decoding POST arguments
(defun create-keywords-from-args (args)
;; segment the string with '&' as a separator
 (let ((mylist (geta-strings::list-words args
"&"))
        pair
        result)
;; for each part, segment with '=' as a separator
and construct the result
  (dolist (lv-pair mylist)
    (setf pair (geta-strings::list-words lv-pair
"="))
    (push (second pair) result)
    (push
     (read-from-string
      (concatenate 'string ":" (first pair)))
result)
      )
    result))
```

This list will be simply used as a list of arguments for a function accepting all the variables defined in the HTML form as keys. Hence, we rely on the standard Lisp binding mechanism to bind each value to its variable. For our purpose, we ask the appropriate dictionary for an HTML version of the entry structure.

Listing 3:
wAlex.acgi 2/3 (MCL Application)

```
Getting the corresponding entry
(defun get-walex-entry  (&key dictid
                         dicttype
                         formname
                         entry
                         &allow-other-keys)
;; Now, we can directly use each variable to
query our dictionary
  (let ((dict (get-dictionary dictid dicttype)))
    (findAndTransmuteItem
      dict
      entry :output-format :html-1))
  )
```

Replying

MacHTTP expects the Apple event reply's direct parameter to contain an HTTP/1.0 header and HTML text that will be transmitted to the client. Hence, we have to construct an appropriate answer. The HTML/1.0 header will be built as a global variable:

```
;; Define the http 1.0 header (beware of crlf)
(defvar *crlf*
  (format nil "~A~A" #\Linefeed #\Newline))
(defvar *http-10-header*
  (format nil "HTTP/1.0 200 OK ~A~
             Server: MacHTTP ~A~
             MIME-Version: 1.0 ~A~
             Content-type: text/html ~A~A"
          *crlf* *crlf* *crlf* *crlf* *crlf*))
```

The reply itself will be returned by the findAndTransmuteItem function. Hence, we just have to put the header and answer in the Apple event reply's direct parameter with the ae-put-parameter-char function.

Listing 4:
wAlex.acgi 3/3 (MCL Application)

```
Defining and installing the Apple event handler

(defmethod my-ae-handler ((a application)
                          theAppleEvent
                          reply
                          handlerRefcon)
;; We don't use the handlerRefcon argument
  (declare (ignore handlerRefcon))
;; Just get the post parameter, decode it, handle
it and reply
  (let* ((post-args
          (create-keywords-from-args
            (ae-get-parameter-char theAppleEvent
                                   :|post| t)))
         (the-answer
          (apply #'get-walex-entry post-args)))
;; Build the reply and put it into the Apple
event reply's direct parameter
    (ae-put-parameter-char reply
#$keyDirectObject
                           (format nil "~A ~A"
                                   *http-
10-header*
                                   the-answer))
))
(install-appleevent-handler :|WWW | :|sdoc|
#'my-ae-handler)
```

An AppleScript Applet As Glue

In this second solution (Figure 5) we use an AppleScript applet to make the glue between MacHTTP and ALEX (MCL). As in the previous solution, the applet is named wAlex.acgi. It is now up to the applet to decode the argument passed by MacHTTP. This time the decoding is done thanks to several Scripting Extensions.

The Tokenize function allows you to transform a string with separating characters into a list. For example, from "A;list;of;word" you will get {"A" "list" "of" "words"} if you take ; as the separator. DePlus does the same, but with the + character (more specific and more efficient). Encoding a URL consists of replacing any problematic characters with their %xx counterpart (for example, the space is replaced by %20). Decoding a URL is the reverse process. DecodeURL_8bits does more by also handling the nightmarish problem of accents (and diacritics in general).

After being activated by MacHTTP, the wAlex.acgi applet calls specific AppleScript Programming Interface (ASPI) procedures of the dictionary server application ALEX (MCL).

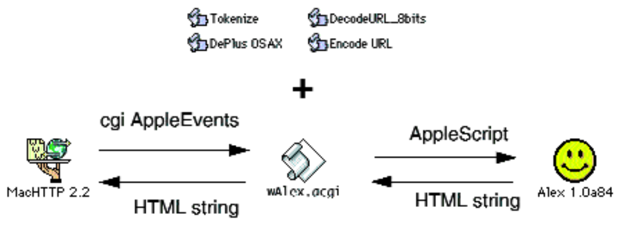


Figure 5. MacHTTP communicates indirectly with ALEX through an AppleScript applet

The communication between MacHTTP and wAlex.acgi is asynchronous. This means that MacHTTP is not waiting for the script to reply. But the communication is synchronous between wAlexScript and ALEX. As we didn't deal with time-out in our script, the default value of 60 seconds applies. ALEX has to reply within one minute (no user will stand a much longer delay anyway).

This solution is well adapted to our needs, as we would like to add more dictionaries that are not necessarily managed by our MCL application. With this solution, we need very little change to the AppleScript applet to integrate new tools.

The script

We use the POST method in our forms or in the links associated with the previous and next entries. In the entry point, the dispatching to the appropriate function is handled on the basis of the method argument.

Notice that for an AppleScript applet, when properties are set outside of the event handlers, they are "remembered" across launches. We want the script to be continuously open.

Listing 5:
wAlex.acgi (AppleScript Applet)

```
Properties and CGI entry point

-- This code comes from Jon Wiederspan and has
been modified to fit our purposes.
-- Some properties are defined for making it
easier to read.
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 theApp : application "Alex"

-- The main entry of our script.
on «event WWW sdoc» path_args ¬
 given «class kfor»:http_search_args, «class
post»:post_args, «class
meth»:method, «class addr»:client_address, «class
user»:username, «class
pass»:|password|, «class frmu»:from_user, «class
svnm»:server_name, «class
svpt»:server_port, «class scnm»:script_name,
«class ctyp»:content_type

-- Decode the Post arguments.
 try
    -- Tokenizing our arguments means creating a
list (or arguments) form a string
 set tokenized_pa to tokenize post_args with
delimiters {"&"}

-- Save the original AppleScript text item
delimiters.
 set oldDelim to AppleScript's text item
```

```
delimiters

-- Use "=" to delimit the pairs.
 set AppleScript's text item delimiters to {"="}

-- Retrieve our arguments (it is positional and
thus not very nice).
 set theEntry to (Decode URL
 (dePlus (last text item of item 1 of
tokenized_pa)))
 set theBaseID to (Decode URL
 (dePlus (last text item of item 3 of
tokenized_pa)))

-- Restore the original AppleScript text item
delimiters.
 set AppleScript's text item delimiters to
oldDelim

-- Returns the result (we don't use theFrom for
simplicity)
 return http_10_header &
 getAlexEntryWithID(theBaseID, theEntry,
post_args)

-- Error handling (not detailed).
 on error msg number num

 end try
end «event WWW sdoc»
```

For our purpose we consider only two arguments (although some others could be useful later on): the word to look up, and a reference to the dictionary. The entry is strictly the string given by the user. The base ID is strictly the selected option of the selected item (in the list of available dictionaries). You can see that in fact we pass the whole argument string along with the two decoded argument. We use this technique for handling optional arguments. In this case the string should be decoded by ALEX (exactly the same way it is done in our first solution). The value of post_args giving rise to Figure 2 is "ENTRY=apple&DictID=3". You can see from Figure 1 what settings in the form this refers to.

And the Winner Is...

The main advantage of the first solution is that the direct interaction between MacHTTP and MCL doesn't make use of AppleScript. The integration is lighter, and thus there is better overall performance in speed and robustness. But you have to delve into the decoding of AppleEvents.

The second solution is technically simpler, and you can very easily change the Lisp interface without changing the overall processing. You can also use other applications to manage dictionaries accessible by the same form. The choice of the target application is done by the applet.

For our actual implementation we retained the second solution, but the choice depends on the goals you assign to your application.

One last question can arise: where we should decode the arguments. We have presented two radically different ways of decoding, with either MCL or the AppleScript applet doing the job. Actually, in the current version of our server, we used a mixture of the two approaches. The AppleScript is used, but decodes only certain arguments (the dictionary name), and always passes the whole argument string to MCL. Then, MCL itself decodes the string with the help of the code presented in listings 2 and 3.

ALEX

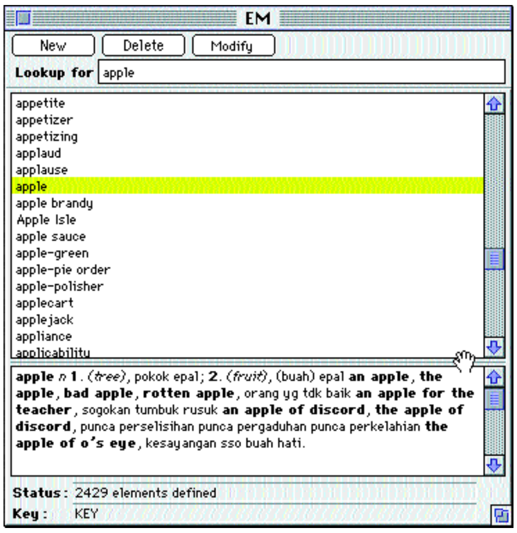
ALEX, the application that serves dictionary information, has been developed with MCL. This tool can be used as an end-user tool with a user interface, or can be remotely and facelessly controlled with AppleScript.

Overview: a small session

ALEX is a small tool allowing a user to open and browse dictionaries. The interface, although minimal, seems quite efficient for the assigned purpose. This tool has been developed as a part of research on the kind of interface needed (or desirable) for structured dictionaries.

The user can scroll through the entry list (not implemented with the list manager, which is limited to around 4000 items), or can make simple active lookups. By active lookup we mean that as the user enters characters into the text box at the top of the window the first entry starting with those characters is selected.

What makes ALEX somewhat different from other tools is that all the entries are kept structured and displayed on the fly, allowing the user to specify some filtering process. It is possible to ask for several kinds of formatting. Another aspect, still under investigation, is the complete scriptability of ALEX.



Lookup for apple

appetite
appetizer
appetizing
applaud
applause
apple
apple brandy
Apple Isle
apple sauce
apple-green
apple-pie order
apple-polisher
applecart
applejack
appliance
applicability

apple *n* **1**. *(tree)*, pokok epal; **2**. *(fruit)*, (buah) epal **an apple**, **the apple**, **bad apple**, **rotten apple**, orang yg tdk baik **an apple for the teacher**, sogokan tumbuk rusuk **an apple of discord**, **the apple of discord**, punca perselisihan punca pergaduhan punca perkelahian **the apple of o's eye**, kesayangan *sso* buah hati.

Status : 2429 elements defined
Key : KEY

Figure 6. An ALEX window displaying the entry "apple" of the English-Malay dictionary

Architecture

The implementation of ALEX is based on several layers, all implemented with MCL. MCL relies on a Dictionary Object Protocol. This manager allows the developer to define multi-key dictionaries and manipulate them. The DOP itself is based on Wood, a persistent Object-Oriented Database developed by Bill St. Clair (ftp://ftp.digitool.com/pub/mcl/contrib/wood/). The persistency allows a minimal memory footprint - a dictionary is never completly loaded into memory. ALEX can be considered as a reduced user interface for the DOP, allowing remote control by AppleScript.

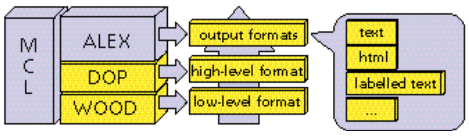


Figure 7. Layers of the actual implementation of ALEX.
The Wood layer stores the low-level format of the dictionary entry. The DOP dynamically decodes each entry in a high-level format. From this high-level format,
ALEX produces any output format.

AppleScript Programming Interface (ASPI)

The do script AppleScript command is a universal command for controlling ALEX. This is very practical for debugging and trying new ideas; you just have to throw Lisp code at ALEX through AppleScript. The dark side is that anyone can do anything. In order for this to work, don't forget to load the eval-server.lisp file located in the MCL Examples folder with a (require :eval-server) form.

We defined some AppleScript commands that can do specific actions: opening and closing a dictionary, looking up, inserting and deleting entries, etc. The two transmute functions are the ones used in the HTML code.

Listing 6:
ASPI

```
getAlexEntryWithID
-- Get the data in the HTML format with the id of
the base.
on getAlexEntryWithID(id, entry, args)
 tell TheApp
 do script ("(get-alex-item (alex::base
 (alex::get-dict-handler \"" & id &
 "\"  :id)) " & "\"" & entry & "\"" &
 " :output-format :html-1 & :args & "\"" args &
"\")")
 end tell
end getAlexEntryWithID
```

As you can see, scripting our MCL application consists simply of building strings that, once given to the Lisp interpreter, will produce some effect. Yes, you can do Lisp from AppleScript! This is simple and neat. Of course, you cannot ask an end-user to write some Lisp code from AppleScript; this is why you have to provide what we call an AppleScript Programming Interface (ASPI). It is a set of AppleScript commands that will control your application. The getAlexEntryWithID function is one of the functions of ALEX's ASPI.

Many of the functions come in two flavors, with dictionary access by ID or by name. For example, getAlexEntryWithID has its getAlexEntryWithName counterpart. Some of them are documented in Listing 7.

Listing 7:
Other ASPI Useful for ALEX

```
openAlexDictionary

-- Open a base from a given filename. Return an
index dictionary or 0 in case of error.
on openAlexDictionary(filename)
 tell theApp
 return do script ¬
 ("(alex::open-object :alex-base :return-index t
:file
 (pathname \"" & filename & "\"))")
 end tell
end openAlexDictionary

getAlexDictionaryName
-- Return the dictionary name based on the
dictionary id.
on getAlexDictionaryName(id)
 tell theApp
 return do script ("(alex:name (alex::base
 (alex::get-dict-handler \"" & id & "\"  :id)))")
 end tell
end getAlexDictionaryName


deleteAlexEntryWithID
-- Deletes an entry (if defined) based on the
dictionary id.
on deleteAlexEntryWithID(id, entry)
 tell theApp
 do script ("(delete-item (alex::base
 (alex::get-dict-handler \"" & id & "\"  :id))
:key "
 & "\"" & entry & "\"" & ")")
 end tell
end deleteAlexEntryWithID
```

Take some time to document your ASPI, and you have achieved the same result as developing your AppleEvents suite, but with less effort and time. One main difference from the more classical method is that your application has no AppleScript dictionary and you cannot use clauses.

Dictionary data structure

The entries are kept in a specific format in the dictionary. This format has been designed to compromise between file space and decoding time. For our dictionary a DOP base is linked to a text file which stands for a text buffer. The low-level format is basically a vector of index-length pairs pointing to this buffer. When an item is found, it is dynamically decoded into a high-level format (example in Listing 10).

The get-alex-item function of ALEX (called by getEntryWithID) basically retrieves one dictionary item and calls transmute-entry, which transforms the high-level format into any output format specifed by the developer. We take advantage of the generic function mechanism of CLOS (Common Lisp Object System) in MCL to have an easily extensible collection of specific formats. The selection of the appropriate method is done on the basis of specialised arguments of the function, basically the type of the current item and the desired output format (cf. Listing 9).

Listing 8:
Two CLOS Headers for Plain Text and HTML Formatting

```
transmute-entry
First method for of text output (code not
detailed).
(defmethod transmute-entry(entry
  (entry-type (eql :en-entry))
  (output-format (eql :text)) )
  )

Second method for one kind of HTML output (code
not detailed).
(defmethod transmute-entry(entry
  (entry-type (eql :en-entry))
  (output-format (eql :html-1)) )
  )
```

Listing 9:
High-level Format of ALEX Items

```
High-level format for “apple”

The high level format of ALEX is a (kind of)
property list. The first item is always the type
of the entry. The
rest of the structure is made of an open list
composed of a type and one or several string
```

```
values.
(:EM-ENTRY (:ENTRY "apple") (:C "n") (:N "1") (:G
"(tree)") (:ME "pokok
epal") (:N "2") (:G "(fruit)") (:ME "(buah)
epal") (:P "an apple") (:P
"the apple") (:P "bad apple") (:P "rotten apple")
(:MPE "orang yg tdk
baik") (:P "an apple for the teacher") (:MPE
"sogokan") (:MPE "tumbuk
rusuk") (:P "an apple of discord") (:P "the apple
of discord") (:MPE
"punca perselisihan") (:MPE "punca pergaduhan")
(:MPE "punca perkelahian")
(:P "the apple of o's eye") (:MPE "kesayangan
sso") (:MPE "buah hati."))
```

Requested output format

The code produced by ALEX for the entry “apple” is given in listing 10 (the code has been slightly edited for readability). Besides the cosmetic features of the page, the most interesting part of the listing is the handling of the “previous” and “next” entries.

Listing 10:
Source of Figure 2

```
HTML “apple”

Beginning of the header of the page and control
part for access to the previous and next entries.
<TABLE CELLPADDING=0 WIDTH="100%"><TD><IMG
SRC="http://161.142.8.195/wSambre-us/ALEX
/Gifs/walex-vert.GIF"
width=10> <IMG ALIGN=Bottom
SRC="http://161.142.8.195/wSambre-us/ALEX
/Gifs/stack.GIF"><b><FONT
SIZE = +6> apple </FONT></B> <TD ALIGN=Right>
<TABLE border=0>

The left arrow and the previous word. Here (and
only here) the GET method is used in the urls
(notice the
? character).
<TD VALIGN=CENTE>R
<A HREF="http://161.142.8.195/wSambre-us/ALEX
/wALEXScript.acgi?ENTRY=applause&FORMNAME=web-
generic-ds.html&DICTID=3"><IMG
SRC="http://161.142.8.195/wSambre-us/ALEX
/Gifs/left-arrow.GIF" BORDER=0></A></TD>
<TD VALIGN=CENTE>R<A HREF="http://161.142.8.195
/wSambre-us/ALEX/wALEXScript.acgi?ENTRY=applause&
FORMNAME=web-generic-ds.html&DICTID=3">
applause</A>
</TD>

The link back to the form.
<TD VALIGN=CENTE>R | <A HREF="http://161.142.8.195
/wSambre-us/ALEX/Forms/web-generic-ds.html">back
to form</A> | </TD>

The right arrow and the next word.
<TD VALIGN=CENTE>R<A HREF="http://161.142.8.195
/wSambre-us/ALEX
/wALEXScript.acgi?ENTRY=apple+brandy&FORMNAME=web-
generic-ds.html&DICTID=3">apple
brandy</A></TD>
<TD VALIGN=CENTE>R<A HREF="http://161.142.8.195
/wSambre-us/ALEX
/wALEXScript.acgi?ENTRY=apple+brandy&FORMNAME=web-
generic-ds.html&DICTID=3"><IMG
SRC="http://161.142.8.195/wSambre-us/ALEX
/Gifs/right-arrow.GIF" BORDER=0></A></TD></TABLE>
</TABLE>

The informative part of this page.
<HR SIZE=4><DL COMPACT>
<DT><B>apple</B>
<DD><I>n</I><IMG HSPACE=8
SRC="http://161.142.8.195/wSambre-us/ALEX
/Gifs/space.GIF">
<IMG HSPACE=6 SRC="http://161.142.8.195/wSambre-
us/ALEX/Gifs/red-diamond.GIF">
<B>1</B> -  (tree); <IMG SRC="http://161.142.8.195
```

```
/wSambre-us/ALEX/Gifs/cyan-diamond.GIF">
pokok epal <IMG HSPACE=6 SRC="http://161.142.8.195
/wSambre-us/ALEX/Gifs/red-diamond.GIF">
<B>2</B> - (fruit); <IMG SRC="http://161.142.8.195
/wSambre-us/ALEX/Gifs/cyan-diamond.GIF">
(buah) epal  rotten apple, bad apple, the apple,
an apple; <IMG SRC="http://161.142.8.195/wSambre-
us/ALEX/Gifs/cyan-diamond.GIF">
orang yg tdk baik  an apple for the teacher; <IMG
SRC="http://161.142.8.195/wSambre-us/ALEX
/Gifs/cyan-diamond.GIF">
tumbuk rusuk, sogokan  the apple of discord, an
apple of discord; <IMG SRC="http://161.142.8.195
/wSambre-us/ALEX/Gifs/cyan-diamond.GIF">
punca perkelahian, punca pergaduhan, punca
perselisihan  the apple of o's eye; <IMG
SRC="http://161.142.8.195/wSambre-us/ALEX
/Gifs/cyan-diamond.GIF">
buah hati., kesayangan sso.</DL>

Information to remind the user in which
dictionary he is looking.
<H>R<FONT SIZE=1>from English-Malay</FONT>
```

Note that the space character in "apple Brandy" has been replaced by a '+'. This has been done at the HTML page production level. Leaving the space would have led to strange and unpleasant results.

By the way, there is a bug (or, say, a slight but unpleasant defect) in this HTML page; can you figure it out? Looking attentively at Figure 2 might help.

Alternative Solutions, Shortcomings And Future Works

There are many unsatisfactory details in the solution we present here, either for the user experience or the developer's technical investment. We are working to fix or bypass some, but many remain beyond our reach, as we do not control the technology.

Filtering

Alternative forms (accessible from the page of Figure 1) allow some filtering on a French-English-Malay dictionary. As the kind of filtering is dependent on the selected dictionary, it was impossible to provide a single interface giving simultaneously the dictionary choice and the filtering options.

Diacritics

A common problem is encountered with diacritics. To properly produce HTML pages, it is compulsory to process each string of the dictionary item and to replace dubious characters by HTML codes. The little bug of Listing 10 is contained in "the apple of o's eye". The closing single smart quote character is not a valid one in HTML, resulting in poor formatting in Figure 2.

Speech

We are considering giving speech output to some information associated with an entry of our dictionary. On the Web server, a Text-To-Speech system could be installed and produce sound files to be downloaded by the client. But that's another story.

Conclusion

Technology integration is difficult for many reasons: you have to be reasonably knowledgeable in the different technologies you deal with, you have to compromise between many factors, and the whole thing is often hairy to debug.

As stated by Apple, AppleScript is the way to integrate technologies on Mac OS; but lots of people find it difficult to develop AppleScript-aware applications. With Macintosh Common Lisp, AppleScript awareness is given for free (the do script command). You can come up with a functional result quickly, satisfy your customers, and have a great tool to experiment with ideas and automated tasks.

Acknowledgments

We would like to thank all the people who participated in the development of the tools mentioned in this paper: Apple and Digitool, of course, but also Jon Wiederspan for his CGI-aware script samples. Thank you, Bill, for the tremendous support to MCL and Wood. We would also not wish to forget all the people involved in our French-English-Malay Dictionary, the staff of the UTMK laboratory in Penang, the French Embassy in Malaysia, the DBDP in Kuala Lumpur, and the GETA in Grenoble, France.

[*If you're curious about the authors and their work (and surroundings), start with the following Web pages, which I uncovered by guesswork (no word-searching necessary). You may need to brush up on your French first! - man*]

http://clips.imag.fr/

http://www.imag.fr/IMAG/IMAG.html

http://www-clips.imag.fr/geta/

http://www-clips.imag.fr/geta/gilles.serasset/

http://www.usm.my/

http://161.142.8.195/wSambre-us/sambre-welcome-us.html

http://161.142.8.195/wSambre-us/Staff/ML-us.html