



Adding Interactivity to a Web Server with 4D

By: David Adams

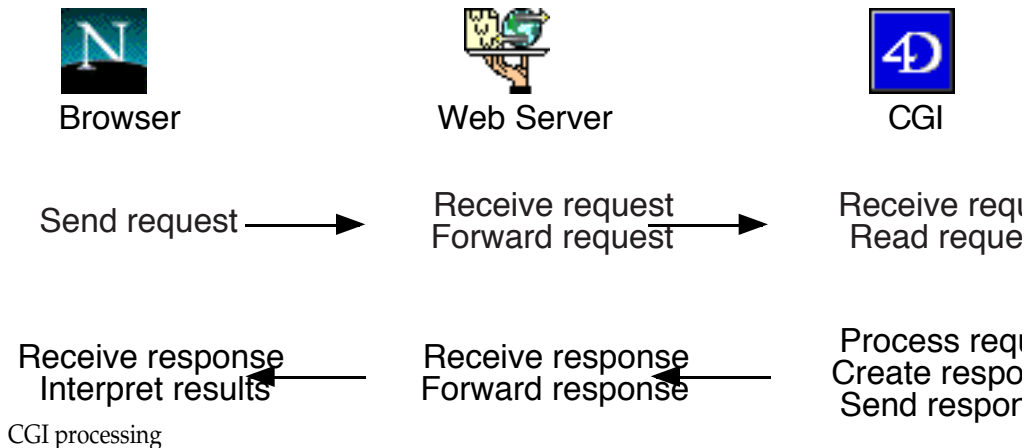
Introduction

World Wide Web servers have the ability to send requests to an external program for processing, and to interpret the results the external program returns. The server and the external program use a standard called the Common Gateway Interface ("CGI") to exchange data in an understandable way. This technote describes techniques which allow you to use 4th Dimension as a CGI to a Web server. By doing so, you can use 4D to add searching, data collection, image map processing, and other interactive features to a Web site.

If you are unfamiliar with any of the Internet terms used in this technote please refer to the "Definition of Terms" section.

Why and How CGI's are Used

A Web server is software that implements the HyperText Transport Protocol ("HTTP"). HTTP is quite simple. It regulates how a browser asks for data, and how the server returns it. Because of this simplicity, Web servers have been ported to virtually every platform that supports TCP/IP. Most Web servers do not directly support custom features like clickable image maps and database searches. Instead, they support access to these features through CGI, a platform independent standard defining how data is sent to an external program for processing, and how the resulting data is to be returned by the external program to the Web server. The external program is usually called a "CGI". Here is a simple diagram of how a client, server, and CGI interact:





As you can see, the Web server is doing little more than managing communication between the browser and the CGI. All custom processing is taking place in 4D. You simply need to learn how to receive, read, and reply to the requests. The exact steps and syntax depend on how you manage communication between 4D and your Web server. This is all you need to create custom Web-based applications.

4D as a CGI

4th Dimension is an ideal environment for writing a CGI. You have access to a local database, can connect to SQL databases, and can use any externals necessary to perform your processing. If you build your code correctly, you will also be able to support Web servers under different operating systems.

CGI Variables

A series of standard environment variables manage the data exchange between a Web server and a CGI. This is primarily what the CGI standard consists of. The exact names of the variables, how CGI's are executed, and how data moves between the server and the CGI are platform specific.

Inter-Application Communication on the Macintosh

On Mac OS, inter-application communication takes place through AppleEvents. The StarNine WebSTAR server creates AppleEvents of type "WWWΩ" and sends them to the CGI. This AppleEvent contains a set of fields that a CGI can read. This is how the standard CGI variables are transported within Mac OS.

4D has built in support for the minimal AppleEvents suite. To do CGI processing, you need support for the WWWΩ event. This requires either System 7 Pack from ISIS International, Inc. (isis@netcom.com), or NetLink/4D from Foresight Technology, Inc. (www.fsti.com). Neither of these packages require AppleScript. Both give you tools to receive, read, and write requests from WebSTAR.

Setting up a Form

The balance of this technote assumes that you are familiar with HTML coding and the form tags. If not, numerous books on this subject have been published, and there is also a wealth of information available for free on the Web. See the "Other Resources" section near the end of this technote for references to useful books and Web sites.

The following HTML code fragment creates the form which follows it:

TECHNICAL NOTE 96-

March 19



```
<HTML>
<HEAD>
<TITLE>Simple Search Example</TITLE>
</HEAD>

<BODY>
<H1>Simple Search</H1>
<HR>
<FORM METHOD = "POST" ACTION =
"/4D_For_Web_Folder/4D_for_Web.acgi$SimpleSearch">
<STRONG>Enter a first and last name to look for:</STRONG>
<P>
<INPUT NAME="First_Name" size=15>
<INPUT NAME="Last_Name" size=15>
<P>
<INPUT TYPE="Reset" VALUE="Clear">
<INPUT TYPE="submit" VALUE="Search">

</FORM>
<P>
<HR>
</BODY>
</HTML>
```

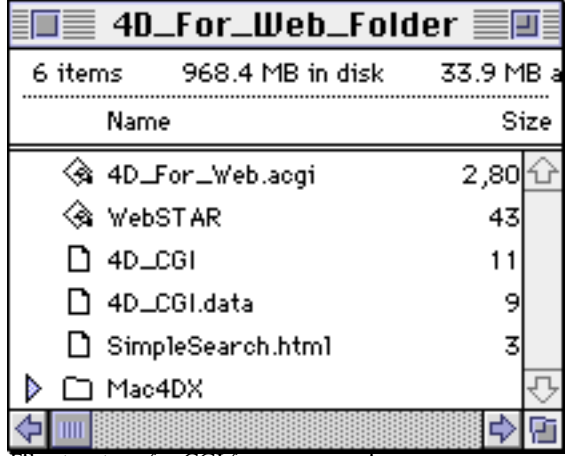


Resulting data entry form in Netscape

When the user presses Search in the browser, WebSTAR creates an AppleEvent and sends it to the application specified by the form's ACTION argument. In this case, the application is 4DForWeb.acgi. Note that the action specifies an *application*, not a database. Here is a screen shot of how this is set up:

TECHNICAL NOTE 96-

March 19



File structure for CGI forms processing

Refer to the manuals for WebSTAR and your external package of choice for complete instructions on setup, configuration and naming conventions.

Here is how the results of the previous search might appear:



A page of HTML returned in response to a simple search.



CGI Processing

After installing the necessary external package, you can use 4D, 4D Runtime or 4D Client as your CGI. Since the system involves exchanges of data between a Web browser, the Web server, and 4D, you will need to run all of these pieces of software at the same time to test your CGI system. As shown above, our CGI database needs to do five things:

- Receive request
- Read request
- Process request
- Create response
- Send response

Receiving the Request

In order to recognize AppleEvents of type WWW Ω , you must activate event processing. The following procedures illustrate this using both System 7 Pack and NetLink/4D:

For System 7 Pack:

```

If (False)
  \ Procedure:  S7_Startup ()
  \ By:        David Adams
  \ Date:      02/96
  \ Purpose:   This routine tells the database to recognize 'WWW $\Omega$ '
AppleEvents,
               and to use the routine S7_Manager to process them.
End if

  \ Declare local variables
C_LONGINT ($LErrorCode)

$LErrorCode := HandleAEVT ("WWW $\Omega$ "; "sdoc"; "S7_Manager")

  \ End of procedure

```



For NetLink/4D:

```

If (False)
  ` Procedure:  NL_Startup ()
  ` By:        David Adams
  ` Date:      02/96
  ` Purpose:   This routine tells the database to recognize "WWWΩ"
AppleEvents,
  `           and to use the routine NL_Manager to process them.
  `           The additional calls tell NetLink to open and manage 4 processes
  `           for incoming events, allowing your database to handle multiple
CGI
  `           requests simultaneously.
End if

  ` Set process that gets incoming requests.
NL_RequestProc ("NL_Manager"; 1; 32768; 4; 4)

  ` Return process management to NetLink.
NL_ProcessProc ("NL_Control2NL")
NL_AcceptReqs (1)

  ` End of procedure
    
```

Complete instructions for using these external packages can be found in their documentation. Once you have invoked event processing, your database will be able to process incoming requests, by launching the manager routine each time an event is generated. Note that with System 7 Pack you can manage only one request at a time (requests are queued), whereas with NetLink/4D you can manage multiple requests in multiple processes simultaneously.

Reading the Request

The AppleEvent contains standard fields to determine the action the Client wants to perform. These fields are defined in the WebSTAR documentation. Here are the fields passed by WebSTAR in the WWWΩ AppleEvent:

TECHNICAL NOTE 96-

March 19



Code	Description	Example
----	Direct parameter	SimpleSearch
kfor	search arguments	
user	user name	
pass	password	
frmu	from user	
addr	client address	192.9.200.131
post	post arguments	NameToSearchFor = Skutch
meth	HTTP method	POST
svnm	server name	192.9.200.131
svpt	server port	80
scnm	script name	/4D_For_Web_Folder/ 4D_For_Web.acgi
ctyp	content type	application/x-www-form- urlencoded
refr	referer	http://duo/SimpleSearch.html
Aght	user agent	Mozilla/1.1N (Macintosh; I; 68K)
Kact	action name	ACGI
Kapt	action path	/4D_For_Web_Folder/ 4D_For_Web.acgi
Kcip	client IP address	192.9.200.131
Kfrq	full client request	POST /4D_For_Web_Folder/ 4D_For_Web.acgi \$SimpleSearch HTTP/1.0 Referer: http://duo/ SimpleSearch.html User-Agent: Mozilla/1.1N (Macintosh; I; 68K) Accept: * / * Accept: image/gif Accept: image/x-xbitmap Accept: image/jpeg Content-type: application/x- www-form- urlencoded Content-length: 22 NameToSearchFor = Skutch

Each external package contains a simple command to extract a field from the AppleEvent:



For System 7 Pack:

```

` Declare local variables
C_TEXT ($tReferer) `          Referrer
C_LONGINT ($LErrorCode) `    Error code (0 = success)

$LErrorCode := GetTextParam (0; "refr"; $tReferer) ` Get referer (URL of calling
page)
    
```

For NetLink/4D:

```

` Declare local variables
C_TEXT ($tReferer) `          Referrer

` Request references are used by NetLink to track multiple events.
C_LONGINT ($LRequestRef)

$LRequestRef := $1
NL_GetParam ($LRequestRef; "refr"; $tReferer) `  Get referer (URL of calling
page)
    
```

NetLink also includes a command called NL_GetParamAry that parses all of the AppleEvent's fields into an array, which can be very convenient.

Processing the Request

Before processing the request, you must determine what the user wants to do, and what additional information he or she has supplied. In the example form, the direct parameter, i.e. SimpleSearch, is used to tell 4D what to do. Here is how the incoming calls are handled using each set of externals:

For System 7 Pack:

```

If (False)
  ` Procedure:   S7_Manager (Incoming form data)
  ` By:         David Adams
  ` Date:       02/96
  ` Purpose:    Parses and creates response to CGI AppleEvent
End if
    
```

```

` Declare local variables
C_LONGINT ($LErrorCode) `    Error code (0 = success)
C_TEXT ($tResultText) `    Result text
C_TEXT ($tReferer) `      Calling page URL
C_TEXT ($tWhatToDo) `     Action to perform
C_TEXT ($tPostArgs) `     Post arguments
    
```

TECHNICAL NOTE 96-

March 19



TECHNICAL NOTE 96-

March 19



```
` Initialize local variables
$tResultText := ""
$tWhatToDo := ""
$tPostArgs := ""

$LErrorCode := GetTextParam (0; "----"; $tWhatToDo) ` Get direct parameter
(action)

If ($LErrorCode # 0) ` Error condition
    $tResultText := CGI_ErrorText ("[Direct parameter test generated error code # " +
        (String ($LErrorCode)) + ".]")
Else ` Success condition
    $LErrorCode := GetTextParam (0; "refr"; $tReferer) ` Get referer (calling page
URL)
    S7_ParseFields ` Parse fields

    S7_ViewParams ` Display HTTP
information
    CGI_ViewFields ` Display form data

If ($LErrorCode # 0) ` Error condition
    $tResultText := CGI_ErrorText ("[Getting referrer generated error code # " +
        (String ($LErrorCode)) + ".]")
Else ` Success condition

    Case of ` I stash the form or search name into the direct parameter.
        : ($tWhatToDo = "") ` Unknown action.
            $tResultText := CGI_ErrorText ("[Empty direct parameter.]; "CGI Error";
                $tReferer)

        : ($tWhatToDo = "SimpleSearch")
            $tResultText := CGI_SimpleSrch (CGI_GetField ("First_Name");
                CGI_GetField ("Last_Name"); $tReferer)

    Else ` Unknown action
        $tResultText := CGI_ErrorText ("[Unrecognized direct parameter: " +
            $tWhatToDo + "]; "CGI Error"; $tReferer)
    End case

End if

End if

` Return results in record generated by Web server
$LErrorCode := PutTextParam (-1; "----"; $tResultText)
```

TECHNICAL NOTE 96-

March 19



End of procedure

TECHNICAL NOTE 96-

March 19



For NetLink/4D:

```
If (False)
  \ Procedure: NL_Manager (Request reference; Direct Parameter)
  \ By: David Adams
  \ Date: 02/96
  \ Purpose: The procedure for NetLink to execute when an incoming
request is received from the web server. This procedure will contain your
4D code to determine what action to take on the incoming request.
End if
```

```
\ Declare parameters
C_LONGINT ($1) \ Request reference
C_STRING (255; $2) \ Direct parameter
```

```
\ Declare local variables
C_LONGINT ($LRequestRef) \ Request reference
C_STRING (255; $sWhatToDo) \ Direct parameter
C_TEXT ($tReferer) \ URL of the requesting page
C_TEXT ($tResultText) \ Text to be returned to Web server
```

```
\ Input parameter reassignment
$LRequestRef := $1 \ $1 is populated by NetLink with the ID of the AppleEvent
record.
$sWhatToDo := $2 \ $2 is populated by NetLink with the direct parameter.
```

```
$tResultText := ""
NL_GetParam ($LRequestRef; "refr"; $tReferer) \ Get referer (URL of calling
page)
NL_ParseFields ($LRequestRef) \ Parse fields into arrays
```

```
NL_ViewParams ($LRequestRef) \ Display HTTP information
CGI_ViewFields \ Display form data
```

```
Case of \ I stash the form or search name into the direct parameter.
```

```
: ($sWhatToDo = "") \ Unknown action
$tResultText := CGI_ErrorText ("[Empty direct parameter.]; "CGI Error";
$tReferer)
```

```
: ($sWhatToDo = "SimpleSearch")
$tResultText := CGI_SimpleSrch (CGI_GetField ("First_Name");
CGI_GetField ("Last_Name"); $tReferer)
```

TECHNICAL NOTE 96-

March 19



```
Else `                                Unknown action
    $tResultText := CGI_ErrorText ("[Unrecognized direct parameter: " +
        $sWhatToDo + "]; "CGI Error"; $tReferer)
End case

NL_AppendReply ($LRequestRef; $tResultText) ` Append the text to the reply.
NL_FlushReply ($LRequestRef; 1) `        End of the stream for this
request.

` End of procedure
```

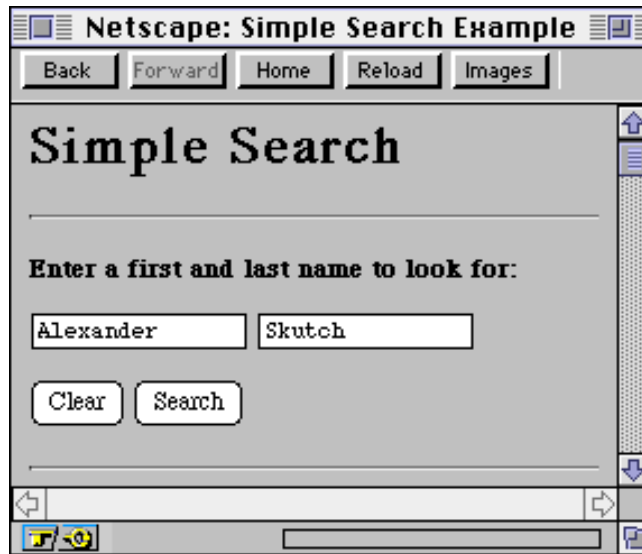
Each of these procedures performs the same tasks:

- Checking what the user has asked to do
- Returning an error if the request is unrecognized or unreadable
- Passing on a request for processing if everything appears fine
- Returning the results to the Web server

Notice that each of these routines call the same subroutines: `CGI_ErrorText` and `CGI_SimpleSrch` (listings for these routines can be found at the end of this note). The code is carefully constructed to separate handling the `AppleEvent` from processing the request. This approach allows you to easily switch between the two packages, and also allow you to add support for other packages, servers, and platforms as they become available. In addition, if you write an HTTP server inside of 4D, you will be able to reuse most of your existing CGI code.

Form Encoding

HTML form data is posted in a special format that you or your external package needs to unpack properly. The rules for form data packing and unpacking are quite simple, but need to be performed properly. The two rules you need to be aware of are name value binding, and URL encoding. The following example form will be used in this discussion:



A simple search with two fields.

Name Value Binding

The preceding form data is packed by the Web client and sent to the Server and then forwarded to the CGI like this:

```
First_Name=Alexander&Last_Name=Skutch
```

This may look a bit obscure, but is actually quite simple. The browser “binds” field names and field values with a conjoining “=” sign.

```
Field name=field value
```

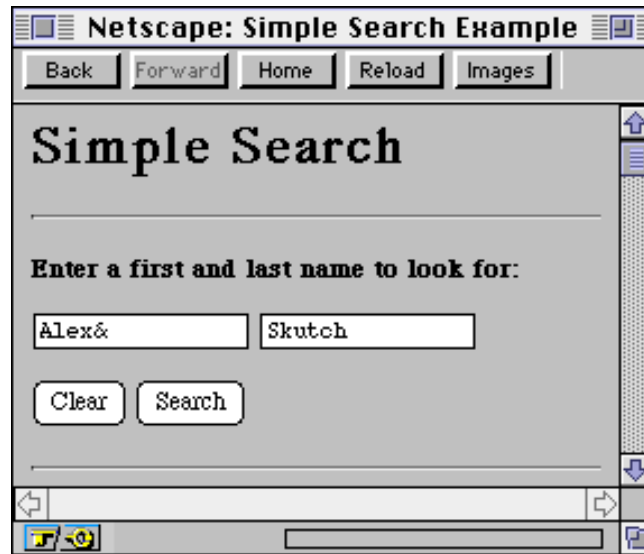
If there are multiple fields they are conjoined with the “&” sign:

```
Field 1 name=field 1 value&Field 2 name=field 2 value
```

NetLink automatically parses out field names and values for you when you use `NL_GetField` or `NL_GetFieldAry`. With System 7 Pack it is necessary to parse these values yourself. See `S7_ParseFields` in the additional code listings at the end of this note, or in the example database.

URL Encoding

As you have just seen the “=” and “&” symbols have a special meaning inside of the form data block. If a field contains either of these characters your program will not be able to unpack the form data correctly. Because of this, these characters are “URL encoded” into hex before being sent by the client.



A form with a special character

This form will be encoded and sent like this:

```
First_Name=Alex%26&Last_Name=Skutch
```

The %## format is used to represent a hexadecimal representation of a character. Several characters besides "=" and "&" are normally encoded in this fashion, so if you write a parsing routine you need to check for every instance of "%".

NetLink automatically URL unencodes incoming form data for you, so you do not need to write your own parser. With System 7 Pack, you need to perform this step yourself. See CGI_UnURLEncode in the additional code listings at the end of this technote.

Additional Notes on Form Data Blocks

In your CGI code you should not rely on the Web browser sending fields in their original order as not all browsers do so. In addition, you should not presume the presence of a field in all cases. Deselected check boxes, for example, are not included in the form data block.

Like all HTML documents, form data uses the character set of the current user, Latin-1 in most cases in the United States. The standard HTML character set is based on ISO standard 8859-1 and differs from the ASCII character set used by the Macintosh and many Windows systems. Converting between the HTML character set and your computer's character set will be the subject of a future technote.



Creating and Sending the Response

The manager routines include code for replying to the AppleEvent, as repeated here:

For System 7 Pack:

```
` Return results in AppleEvent record generated by Web server.  
$LErrorCode := PutTextParam (-1; "----"; $tResultText)
```

For NetLink/4D:

```
NL_AppendReply ($LRequestRef; $tResultText) ` Append the text to the reply.  
NL_FlushReply ($LRequestRef; 1) ` End of the stream for this  
request.
```

Both packages take care of constructing a valid HTTP 1.0 message header for your responses. As shown here, NetLink allows you to stream data incrementally to the server for downloading to the client. This takes advantage of WebSTAR's implementation of HTTP "server push". You need this feature if you are going to send 64 Kb of data or more. It is a good idea to return some text to the user quickly so that he or she feels that the system is responsive. It is also a good idea to restrict the amount of data returned so that a user on a slow connection is not overwhelmed.

NetLink also includes a special command to issue a redirection to another URL if you want to point the client to another Web page or network resource in response to a submission.

The Web is Not Just for The Internet

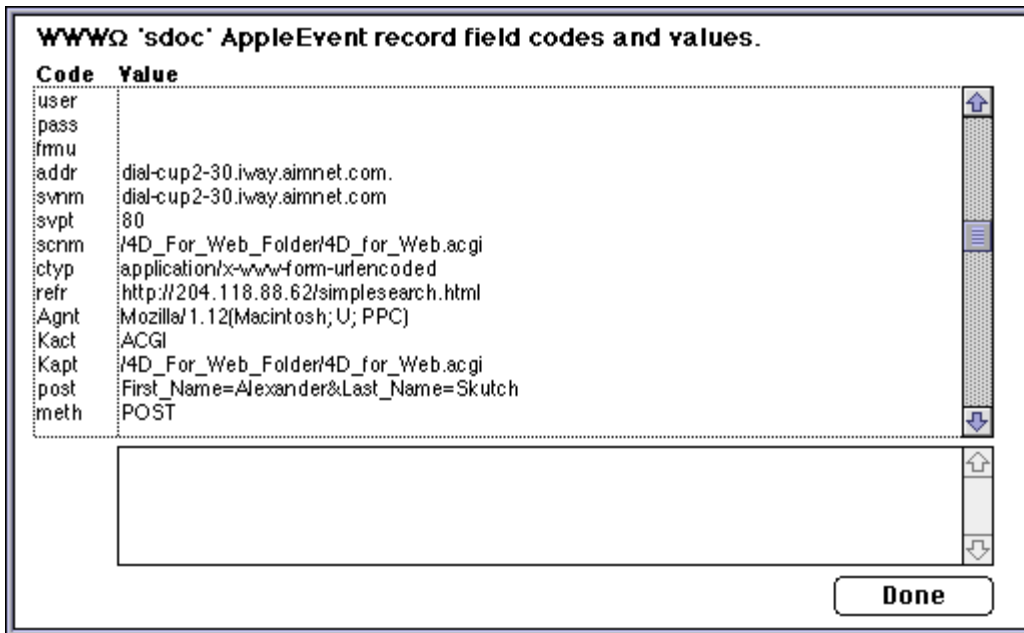
Keep in mind that Web browsers, servers, and CGI's can run on any TCP/IP network. By far the largest TCP/IP network is the Internet, but you can do everything discussed in this technote on your own LAN. In so doing, you can learn how all of this works, and test your solution before putting it on the Internet. These techniques are also effective for delivering internal systems that will never be put on the Internet.

The Example Database

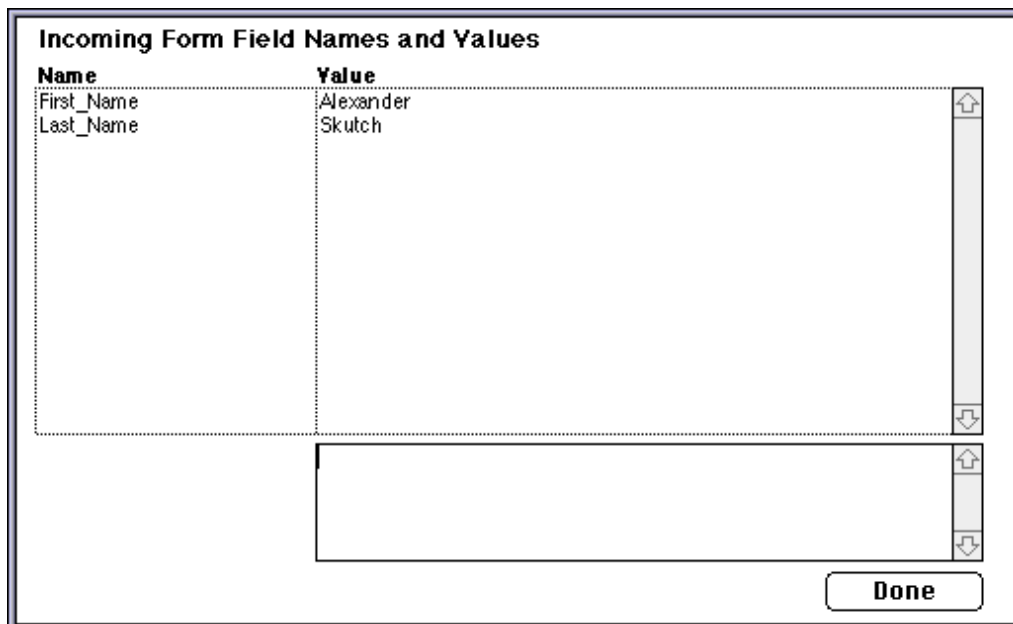
The example database includes all of the procedures discussed in this technote, as well as debugging utilities for displaying the contents of incoming AppleEvent and forms data. These utilities display dialogs like the ones pictured here:

TECHNICAL NOTE 96-

March 19



Incoming AppleEvent data parsed and displayed in a debugging dialog.



Incoming form data parsed and displayed in a debugging dialog.

Other Resources

The best source of information on the Internet is the Internet itself. If you prefer books,



several are recommended here:

- Jon Wiederspan maintains a very handy set of Web pages devoted to Internet software and standards at <http://www.comvista.com/net/Directory.html>.
- If you are developing with NetLink, Foresight's NetLink-Talk e-mail list is a great resource. Visit their web site for complete details (<http://www.fsti.com/productinfo/netlink.html>).
- *The HTML Manual of Style* by Larry Aronson (ZD Press, \$19.95) is a well written guide to HTML. By the time you read this note a new edition of this book, *The HTML 3.0 Manual of Style* will be available. There are dozens of books on this subject, so shop around.
- *Managing Internet Information Services* by Liu, Peek, et al (O'Reilly & Associates, Inc. \$29.95) is by far the best technical overview of the Internet.
- *TCP/IP Network Administration* by Craig Hunt (O'Reilly & Associates, Inc.) contains a wealth of technical information on all aspects of TCP/IP. The book is designed to enable you to find the information you need at your level of interest.
- If you are using WebSTAR or MacHTTP as a Web server you should pick up a copy of *Planning and Managing Web Sites on the Macintosh* by Jon Wiederspan and Chuck Shotton (Addison-Wesley, \$39.95). Chuck Shotton is the author of MacHTTP and WebSTAR.

Definition of Terms

This technote assumes that you are already familiar with the Web as a user, and have at least a basic understanding of HTML, but does not assume that you have yet created a Web site. Because the world of Internet protocols is rich in acronyms and other jargon derived largely from the UNIX world, a short definition of terms is in order.

HTTP (HyperText Transfer Protocol)

This protocol defines how a Web server and a Web client exchange data. The most popular Macintosh HTTP server is WebSTAR from StarNine. Previously this product was called MacHTTP from BIAP systems. The standard browsers are Netscape and Mosaic.

HTML (HyperText Markup Language)

HTML is a platform and display independent tagging system for structured documents. The language contains styling tags that are interpreted by each browser in a



manner suitable to the computer and operating system.

CGI (Common Gateway Interface)

A "CGI" or "ACGI" (Asynchronous CGI) is an external application with which an HTTP server communicates. The CGI standard defines how the server and CGI exchange data, but says nothing about what the external program can do. The external program, i.e. 4D in our case, can do anything you program it to so long as you return the results in a manner meaningful to the HTTP server.

Contact Information

NetLink/4D

NetLink/4D lets 4D act as a CGI with WebSTAR and other Macintosh Web servers. NetLink handles all Apple Event parsing and manages multiple processes to allow simultaneous database access from people accessing the Web server. NetLink handles all HTTP headers, supports all HTML tags, and can convert PICT images into GIF or JPEG format.

Foresight Technology, Inc. · <http://www.fsti.com/productinfo/net>

System 7 Pack

System 7 Pack allows full control over AppleEvents from within 4D. System 7 Pack can be used to completely automate the interaction between a Macintosh Web server and 4D.

ISIS International, Inc. · isis@netcom.com

WebSTAR

WebSTAR is currently the most widely used Web server on the Internet. Originally written by Chuck Shotton as MacHTTP it is currently distributed by StarNine, a division of QuarterDeck.

StarNine. · <http://www.starnine.com>

Additional Code Listings

CGI_ErrorText:

```
If (False)
  Procedure: CGI_ErrorText (Error text) -> Formatted error text page.
```

TECHNICAL NOTE 96-

March 19



```
` By:          David Adams
` Date:        02/96
` Purpose:     Error processing procedure
End if
```

```
` Declare parameters
C_TEXT ($0) `      Formatted error text
C_TEXT ($1) `      Error text
C_STRING (80; $2) ` Title
C_TEXT ($3) `      Path to referer page
```

```
` Declare local variables
C_STRING (1; $sCR) ` Carriage return character
C_STRING (80; $sTitle) ` Title
C_TEXT ($tErrorText) ` Error text
C_TEXT ($tReferer) ` Path to referer page
C_TEXT ($tResultText) ` Formatted error text
```

```
` Input parameter reassignment
$tErrorText := $1
$sTitle := $2
$tReferer := $3
```

```
$sCR := Char (13)
$tResultText := ""
```

```
$tResultText := $tResultText + "<HTML>" + $sCR ` The CRs are not necessary,
they
```

pretty format the HTML

source.

```
$tResultText := $tResultText + "<HEAD>" + $sCR
$tResultText := $tResultText + "<TITLE>" + $sTitle + "</TITLE>" + $sCR
$tResultText := $tResultText + "</HEAD>" + $sCR
$tResultText := $tResultText + "<BODY>" + $sCR
$tResultText := $tResultText + "<H1>" + $sTitle + "</H1><P><HR>" + $sCR
$tResultText := $tResultText + "The 4D Web cgi encountered a problem with
your submission." + "<P>" + $sCR
$tResultText := $tResultText + $tErrorText + "<P>" + $sCR
```

```
` Add a link back to the referring page
$tResultText := $tResultText + CGI_LinkReferer ($tReferer)
$tResultText := $tResultText + "</BODY>" + $sCR
$tResultText := $tResultText + "</HTML>" + $sCR
```

```
$0 := $tResultText
```


TECHNICAL NOTE 96-

March 19



```
C_TEXT ($1) `           First name
C_TEXT ($2) `           Last name
C_TEXT ($3) `           Referer (URL of calling page)

` Declare local variables
C_LONGINT ($LCounter) ` Loop counter
C_LONGINT ($LNamesFound) ` Number of names found
C_STRING (1; $sCR) `    Carriage return Character
C_STRING (80; $sTitle) ` Title
C_TEXT ($tFirstName) ` First name to search for, may be blank
C_TEXT ($tLastName) `  Last name to search for, may be blank
C_TEXT ($tResultText) ` Formatted results (a complete HTML page)
```

```
` Input parameter reassignment
```

```
$tFirstName := $1
$tLastName := $2
$tReferer := $3
```

```
$tResultText := ""
$sTitle := "Simple Search Results"
$sCR := Char (13)
```

```
$tResultText := $tResultText + "<HTML>" + $sCR `    The CRs are not necessary,
`           they pretty format the code
```

```
$tResultText := $tResultText + "<HEAD>" + $sCR
$tResultText := $tResultText + "<TITLE>" + $sTitle + "</TITLE>" + $sCR
$tResultText := $tResultText + "</HEAD>" + $sCR
$tResultText := $tResultText + "<BODY>" + $sCR
$tResultText := $tResultText + "<H1>" + $sTitle + "</H1><P><HR>" + $sCR
```

```
If ($tFirstName + $tLastName = "") `    No search values supplied.
    $tResultText := $tResultText + "<B>Error: </B>You did not supply any search
    values." + $sCR
```

```
Else `    Search conditions supplied.
```

```
Case of `    Construct and execute search based on values supplied.
```

```
: ($tFirstName = "") `    Last name search
    SEARCH ([Contacts]; [Contacts]LastName = $tLastName)
```

```
: ($tLastName = "") `    First name search
    SEARCH ([Contacts]; [Contacts]FirstName = $tFirstName)
```

```
Else `    Full name search
    SEARCH ([Contacts]; [Contacts]FirstName = $tFirstName; *)
    SEARCH ([Contacts]; & ; [Contacts]LastName = $tLastName)
```

TECHNICAL NOTE 96-

March 19



End case

```
$tResultText := $tResultText + "Your search statement: <BR>" + $sCR
$tResultText := $tResultText + "First Name: <B>" + $tFirstName + "</B><BR>" +
$sCR
$tResultText := $tResultText + "Last Name: <B>" + $tLastName + "</B><P>" +
$sCR
```

\$LNamesFound := Records in selection ([Contacts])

Case of

```
:( $LNamesFound = 0)
  $tResultText := $tResultText + "<B>0</B> names were found matching your
  conditions." + "<BR>" + $sCR
```

```
:( $LNamesFound = 1)
  $tResultText := $tResultText + "<B>1</B> name was found matching your
  conditions." + "<BR>" + $sCR
```

```
:( $LNamesFound > 1)
  $tResultText := $tResultText + "<B>" + (String ($LNamesFound)) + "</B>
names
  were found matching your conditions." + "<BR>" + $sCR
```

End case

TECHNICAL NOTE 96-

March 19



```
If ($LNamesFound > 0)
  FIRST RECORD ([Contacts])

  For ($LCounter; 1; $LNamesFound)
    $tResultText := $tResultText + "<HR><BR>" + $sCR
    LOAD RECORD ([Contacts])

    Case of

      : ([Contacts]FirstName = "") & ([Contacts]LastName = "") ` No name data
        ` Do nothing

      : ([Contacts]FirstName = "") | ([Contacts]LastName = "") ` One is blank
        $tResultText := $tResultText + "<B>" + [Contacts]FirstName +
          [Contacts]LastName + "</B><BR>" + $sCR

      Else ` First and last name
found $tResultText := $tResultText + "<B>" + [Contacts]FirstName + " " +
          [Contacts]LastName + "</B><BR>" + $sCR

    End case

    If ([Contacts]PhoneNumber # "")
      $tResultText := $tResultText + "Phone # " + [Contacts]PhoneNumber +
"<P>" +
        $sCR
    End If

    NEXT RECORD ([Contacts])
  End for

  UNLOAD RECORD ([Contacts])
End If

End If ` ($tFirstName + $tLastName =
"")

$tResultText := $tResultText + "<HR>" + $sCR

` Add a link back to referring page
$tResultText := $tResultText + CGI_LinkReferer ($tReferer)
$tResultText := $tResultText + "</BODY>" + $sCR
$tResultText := $tResultText + "</HTML>" + $sCR

$0 := $tResultText
```



 ` End of procedure

S7_ParseFields

If (False)

 ` Procedure: S7_ParseFields ()
 ` By: David Adams
 ` Date: 02/96
 ` Purpose: This routine parses incoming form fields into a pair of arrays:
 ` atFldNames: The name of each field
 ` atFldValues: The corresponding value in each field

End if

 ` Declare local variables

C_LONGINT (\$LErrorCode) ` Error status code from S7P calls
C_LONGINT (\$LPosition) ` Used to hold Position of target characters
C_LONGINT (\$LItem) ` Position of top item in array
C_LONGINT (\$LThisItem) ` Loop counter
C_LONGINT (\$LArraySize) ` Size of complete field array
C_TEXT (\$tPostArgs) ` Compound form data block from post
argument.
C_TEXT (\$tFieldData) ` Temporary holding for a field name and value
pair

 ` Initialize field arrays

 ARRAY TEXT (atFldNames; 0)
 ARRAY TEXT (atFldValues; 0)

 \$LErrorCode := GetTextParam (0; "post"; \$tPostArgs) ` Get the form
arguments.

 \$LItem := 1

 If (\$LErrorCode = 0)

 Repeat

 \$tFieldData := ""
 \$LPosition := Position ("&"; \$tPostArgs)

 If (\$LPosition > 0)

 \$tFieldData := Substring (\$tPostArgs; 1; \$LPosition-1) ` Holds field name &
value

 \$tPostArgs := Substring (\$tPostArgs; \$LPosition + 1)

 Else

TECHNICAL NOTE 96-

March 19



```
$tFieldData := $tPostArgs  
$tPostArgs := ""  
End if
```

TECHNICAL NOTE 96-

March 19



```
$LPosition := Position (" = "; $tFieldData) ` Find where name & value break  
apart
```

```
INSERT ELEMENT (atFldNames; $LItem; 1)  
INSERT ELEMENT (atFldValues; $LItem; 1)  
atFldNames{$LItem} := Substring ($tFieldData; 1; $LPosition-1)  
atFldValues{$LItem} := Substring ($tFieldData; $LPosition + 1)  
$LItem := $LItem + 1
```

```
$LPosition := Position ("&"; $tPostArgs)  
Until ($tFieldData = "")
```

```
$LArraySize := Size of array (atFldValues) ` How many items are there?
```

```
For ($LThisItem; 1; $LArraySize)  
  atFldValues{$LThisItem} := CGI_UnURLEncode (atFldValues{$LThisItem})  
End For
```

```
End if
```

```
` End of procedure
```

CGI_UnURLEncode

```
If (False)  
  ` Procedure: CGI_UnURLEncode (Text) -> Text  
  ` By: David Adams  
  ` Date: 02/96  
  ` Purpose: This routine converts character converted to hex back to  
  ` decimal ascii  
End if
```

```
` Declare parameters  
C_TEXT ($0) ` Passed value  
C_TEXT ($1) ` Cleaned value
```

```
` Declare local variables  
C_TEXT ($tSourceText) ` Original text  
C_TEXT ($tResultText) ` Formatted link to referrer page  
C_LONGINT ($LLength) ` Length of original string  
C_LONGINT ($LThisChar) ` Current character in loop
```

```
` Input parameter reassignment  
$tSourceText := $1
```

```
$tResultText := ""  
$LLength := Length ($tSourceText)
```

TECHNICAL NOTE 96-

March 19



```
For ($LThisChar; 1; $LLength)
```

```
  If ($tSourceText≤$LThisChar≥ = "%") ` Beginning of hex encoding sequence
    $tResultText := $tResultText + Char (BA_ToDecimal (Substring ($tSourceText;
      $LThisChar + 1; 2); 16)) `      Convert from hex and change to a
```

```
character
```

```
    $LThisChar := $LThisChar + 2
```

```
  Else
```

```
    $tResultText := $tResultText + $tSourceText≤$LThisChar>
```

```
  End if
```

```
End For
```

```
$0 := $tResultText
```

```
` End of procedure
```

BA_ToDecimal

For details on this procedure see: *Using Non-Decimal Numbers for Bar Coding and Other Purposes* by Jeff Browning (TN 96-15, March 1996)

```
If (False)
```

```
` Procedure:   BA_ToDecimal (Non-decimal; Base; {Base key}) -> Decimal
```

```
` By:         Jeff Browning
```

```
` Date:       02/96
```

```
` Purpose:    Converts non-decimal number to decimal number
```

```
End if
```

```
` Declare parameters
```

```
C_LONGINT ($0) `      Decimal number
```

```
C_STRING (255; $1) `  Non-decimal number
```

```
C_LONGINT ($2) `      Base to use (2 - 111)
```

```
C_STRING (111; $3) `  Base key string; optional
```

```
` Declare local variables
```

```
C_LONGINT ($LResult) `      Decimal number
```

```
C_LONGINT ($LBase) `        Base to use (2 - 111)
```

```
C_LONGINT ($LCurrChar) `     Current char being processed
```

```
C_LONGINT ($LMaxChar) `     Length of the base 36 number string
```

```
C_LONGINT ($LPos) `         Position of the current char in the key string
```

```
C_STRING (111; $sKey) `     Base key string (See Startup_BA)
```

```
C_STRING (255; $sSource) `  Non-decimal number
```

```
` Input parameter reassignment
```



```

$sSource := $1
$LBase := $2
If (Count parameters = 3) `           Base key string passed
    $sKey := $3
Else
    $sKey := ◇BA_sKey
End if

    ` Initialize local variables
$LMaxChar := Length ($sSource)
$LResult := 0

    ` Process each char in the source string moving from right to left
For ($LCurrChar; 1; $LMaxChar)

    ` Obtain the position of each char in the key string
    $LPos := Position ($sSource≤$LMaxChar - $LCurrChar + 1≥; $sKey)

    If ($LPos > 0) `           Char found

        ` Add to the result the number indicated by the position and place of the char
        $LResult := $LResult + (($LPos - 1) * ($LBase ^ ($LCurrChar - 1))) ` 36 ^ 0 = 1
    Else `           Error condition
        BEEP
        ALERT ("Incorrect source string. Character not found in key string")
        $LCurrChar := $LMaxChar ` Kick out of the loop
        $LResult := 0
    End if

End for

$0 := $LResult

    ` End of procedure

```

Summary

This technote has discussed using 4D to provide extra processing for a Web server like WebSTAR. 4D is well suited to providing the kind of interactive features people are growing to expect on the Web, including searches, data collection, image maps and conferencing. In addition, this technote contains code designed to help you build CGI systems that function correctly and are easy to maintain. Keep in mind that Web-based applications can be used both on the public Internet and any private company TCP/IP network.