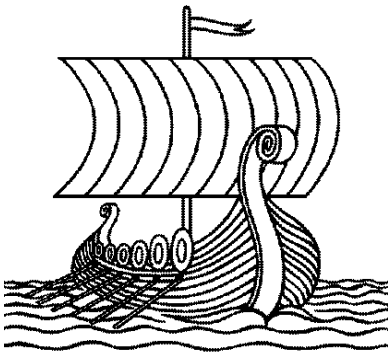

Q'Nial

Using CGI-Nial

Version 6.3

August 2005

Preface

Introduction

CGI-Nial is a special version of Console Nial that can be used as a CGI processing language for supporting Web processing. It provides a powerful computational ability to compute dynamic HTML to support forms within a Web-based application. CGI-Nial is executed by a URL reference that requests the execution via the Common Gateway Interface of a file that triggers the executable *cginial*. Such files are Nial Definition files (.ndf) or Nial Form files (.nfm) On Windows systems the suffix is used to trigger the execution of CGI-Nial. On Unix, the files begin with the specific code to trigger the executable.

This document introduces the concept of a Nial Form, describes the additional built-in facilities that have been added in CGI-Nial and discusses how the facilities are used to build Web-based applications.

Nial Form files

A Nial form file is a special file layout intend for direct use with CGI-Nial to generate dynamic HTML. The file has three sections: a header, an HTML template and Nial program code.

The default header is:

```
#!/bin/csh
exec /usr/local/bin/cginial $0 $*
NODEBUG
NOLOG
NOVARIABLES
NOPROFILE
```

The first two lines are for Unix versions and are used to trigger the execution of CGI-Nial, assuming it has been place in "/usr/local/bin". They are treated as a comment by the Windows version.

The next four lines are used to control debugging capabilities provided as part of the form processing process. If NODEBUG is replaced with DEBUG then the execution of the Nial code in the form file is output and will appear in the HTML document. If NOLOG is changes to LOG, then the execution of the Nial code is placed in the log file *cgi_form.log*. If NOVARIABLES is changes to VARIABLES then the values associated with names passed by CGI are displayed at the top of the HTML page. Finally, if NOPROFILE is replaced with PROFILE, then the execution of the Nial code is profiled and the information is written to *cgi_form.prf*.

The HTML template is a block of HTML code with special tags that indicate where data is to be replaced dynamically. The overall layout of the page as displayed in the browser can be set up by this fixed HTML code. The HTML code can be hand crafted or prepared with the help of an HTML editor.

There are two kinds of special tags: NIALVAR and NIALINCLUDE. The first tag can be used anywhere within an HTML construct in the form "NIALVAR:varname" where an explicit string would go. When the template is processed the string associated with *varname* is substituted for the tagged name.

The second tag is used like an HTML construct. The syntax <NIALINCLUDE:varname> is placed where you want a list of dynamically generated HTML strings to be placed. When the template is processed the entire construct is replaced by the list of strings associated with *varname*. Here is a sample of a HTML template from the file *simple.nfm*:

```
<HTML>
<HEAD>
<TITLE>Simple test of a Nial form</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF" TEXT="#333333" link=ff0707 alink=ffffff
vlink=5f9ec0>
<FONT FACE="ariel"><FONT FACE="Helvetica"><FONT SIZE=2>
<p>
<hr size=4>
<p>
<NIALINCLUDE:outputresults>
<hr size=4>
<form METHOD = GET action="/cgi-shl/simple.nfm">
<p>
In the input area below you may enter a name. You will
receive a greeting with this name.
<p>
<input type=reset value="Clear">
<input type=submit  value="Execute">
<input type=string name="Name" value="NIALVAR:Defaultname">
</form>
</FONT></FONT></FONT>
</BODY>
</HTML>
```

The page displays a message that is generated as *outputresults* when it is processed, and presents a form with an input field and two buttons marked Execute and Clear. The initial contents of the input field is determined by *Defaultname*. The action field for the form points to the same Nial form file.

The Nial code section of the form consists of any number of Nial definition and statements. The purpose of the code is to compute the values of any of the fields that are dynamic in the HTML code. The Nial code from *simple.nfm* is:

```
Defaultname := 'Tom';

Generate_simple_form IS {
  IF existval 'Name' THEN
    Name := getfirstval 'Name';
    IF empty Name THEN
      Outputresults := [''];
    ELSE
      Outputresults := [link 'Hello ' Name];
    ENDIF;
  ELSE
    Outputresults := [''];
  ENDIF;
  Varnames := ['Outputresults','Defaultname'];
  Varpairs := Varnames (EACH value Varnames);
  SetVarPairs Varpairs;
}

Generate_simple_form;
```

The main routine of the Nial code in this case is *Generate_simple_form*. The first part of this routine sets up the value of *Outputresults*. The second part uses the suggested way of setting up the associations between names in the Nial code and names used in the special tags. The alternative would be to declare *Outputresults* as a NONLOCAL name in the routine. *Defaultname* is declared outside the routine and hence is global. The advantage of using the association list created by *SetVarPairs* is that it avoids the possibility that a name you are using interferes with a name used in the template processing code. The template code first looks in the association list. If the varname is not found there then it is queried as a Nial variable.

If the name of a Nial form file is given to *cginial* as its argument, then it is automatically processed using the process described below. An alternative approach is to use a Nial definition file that explicitly does the equivalent processing. The latter approach makes sense when you want one control program to determine which of several possible pages is to be displayed next.

CGI-specific Routines

The cgi-library consists of a number of routines for assisting in dealing with Nial scripts triggered from CGI. It consists of two parts, a set of utility routines similar to the standard Perl file cgi-lib.pl used by many Perl CGI scripts, and a set of routines to do the Nial Form processing. The first part provides routines to assist in processing command line and environmental variables that form the arguments and parameters to CGI scripts. The entire library is built into CGI-Nial.

Utility Routines

The following is a summary of the utility routines defined in the cgi-libray included in CGI_Nial:

Readparsecgiinput
: This expression extracts all CGI arguments regardless of their source and place them into an association list. It's result is a boolean indicating if there were any CGI variables actually processed.

addkeyval Key Val
: This routine is used internally when building the association list for CGI arguments. It takes a string or phrase key and an arbitrary value, and stores them in the association list. If there is already a value for that key, then the new value is added to the existing value. The association list keeps the values for each key in a list. Under CGI control, the values will always be strings.

getvals Key
: Looks up the name of the CGI variable Key (string or phrase) and returns the list of values associated with the supplied key. The result is always a list, possibly empty. The Key names do not distinguish case.

Setup_globals
: Sets a few of the global variables used by the other routines. It should be called as the first action taken by a *.ndf file used as a CGI script. It is called automatically in processing a Nial form file.

Printheader
: Prints the standard header to designate the file as an HTML content file to the output stream. This must be called before any output can be produced. (Otherwise your browser will report something like: "No blank line separating header and contents"). It is called automatically in processing a Nial form file.

htmltop Title
: Prints the necessary HTML code to start an HTML page and give the provided Title (string or phrase).

Htmlbot
: Outputs the matching HTML code to the *HtmlTop* operation.

Methget Returns true if the CGI script is running with the GET method (see CGI documentation). It is used internally in parsing the CGI variables.

Methpost Returns true if the CGI script is running with the POST method (see CGI documentation). It is used internally in parsing the CGI variables.

Mybaseurl Returns the base URL to this script.

Myfullurl Returns the full URL to this script, including the argument string.

Printvariables Displays all values in the CGI variable association list.

Printenvvars Displays the current contents of the systems environmental variables at the time of script processing.

cgierror Msg Used to return the string Msg as a fatal CGI error message to the caller. The user should close the output with *HtmlBot*.

cgidie Msg Used to terminate the script after providing the caller with the string Msg.

rowstohtml Tblrows Maxwidth
Routine to take a list of list of strings representing the entries in a table and decorate them with HTML tags to create the table rows. Entries are truncated to length Maxwidth.

listtohtml A Bordersize
Routine to generate an HTML table from a Nial list of lists of strings. It uses rowstohtml with Maxwidth = 80. Bordersize is used to indicate the width of the border, with 0 indicating no border.

arraytohtml A Bordersize
Routine to generate an HTML table from a Nial table of strings. It uses rowstohtml with Maxwidth = 80. Bordersize is used to indicate the width of the border, with 0 indicating no border.

Nial Form processing Routines

The following is a summary of the routines defined in the cgi-library included in CGI-Nial that are used for Nial Form processing:

get_template_value Name
Gets the Nial array value associated with Name, first looking in the hidden Varpairs list and then, if not found, executing it as a Nial variable.

setvarpairs Varpairs
Stores the association list in a hidden variable.

Process_template
Processes the template replacing the special tagged fields with the corresponding Nial string or list of strings values. The reulst is the list of lines to be displayed.

int_loaddefs Code
Simulates a *loaddefs* from a list of lines. Code is a list of strings selected from the Nial Form file. The result is the HTML output needed to display the code and the results of execution. This is used in processing a form and by the demo that executes Nial code.

process_form_file Filenm

Does the processing to separate the Nial Form file, Filenm into its three sections, executes the Nial code and sets up debugging, logging or profiling information if requested. It returns the HTML Template to be processed.

Process_script Routine called internally by CGI-Nial to process a Nial form file provided as the argument to *cginial*.

CGI-Nial on Windows Systems

To run Nial scripts on Windows Systems, you must first create an *Association* between a particular extension and an executable. In this case the extension should be ".ndf" and the executable should be the CGINIAL.EXE file. Additionally, a similar association between "*.nws" files and CGINIAL.EXE can be made, and you will then be able to execute workspaces. Most Windows based Web Severs rely on associations to launch applications. The install program provided with the Nial Tools sets up these associations.

Some of the details of how this works are very dependent on the particular Web Server that is running.

CGI-Nial on Unix Systems

Unix systems don't have file association, by they do allow executable scripts to identify the command processor to run on the contents of the script. The following is an example of what a Nial script should look like on a typical Unix system. It is critical that there is no space between the first two lines. The "#" on the first line will been seen as a comment by the Nial interpreter, and the remainder of the file can be normal Nial code.

The file must have the extension ".ndf" or ".nfm" and must have its execute bit set (ie. chmod +x xxx.ndf).

Example Unix Nial script

```
#!/bin/csh
exec /usr/local/bin/cginial $0 $argv[*]

%-- Set up the global variables;
Setup_globals;

%-- Printout the standard header/content info;
Printheader;

%-- Process the CGI Arguments;
Readparsecginput;

%-- Output the basic HTML title info;
htmltop 'NIAL CGI Test';

%-- Let's display the variables that were passed to this script;
Printvariables;

%-- Print Environmental Variable (lots'o'stuff);
Printenv;

%-- talk to the user a little bit;
writescreen 'Checking for code<br>';

%-- Extract the CGI variable "code", if available;
Code := getvals 'code';

%-- If we find something in the variable, then execute with Nial;
IF not empty Code THEN
  writescreen '<PRE>';
  writescreen  picture execute first code;
  writescreen '</PRE>';
ELSE
  writescreen 'No Code to execute<br>';
ENDIF;

%-- Clean out the HTML output;
Htmlbot;


Bye
```

NOTE: The above example works under both Unix and Windows 95/NT. In the latter case the first line is not interpreted by the operating system and is treated as a *remark* by CGI-Nial.

Routines to support Persistence

The following routines are part of the CGI Library to support persistence of data between frames. These allow the direct storage of Nial data in binary files that only survive for one data exchange.
The routines are:

Uniqueno

generates a unique integer used in the name the temporary file.

reset_uniqueno Startno
resets the unique number to the give start number.

deletefile Filenm
deletes the named file using the appropriate file command for the system.

buildpersistentstore Varnames Vals
creates a unique filename and stores the names and values given as the arguments. Returns the unique filename. This is usually passed as a hidden variable in the form.

extractpersistentstore Filename
reads the names and values of the variables from the named file, returns them as a pair of lists and deletes the named file.

Creating a CGI-NIAL based application

The development of a Web application requires the design and management of a number of HTML pages. For pages that just display static data, or which contain only links to other pages, the HTML for the page can be created once and stored for subsequent use. However, for applications in which Web technology is being used to access data dynamically, or in which computations are being done on request, the response to a request is HTML text that includes a mixture of static text and dynamic data.

CGI-Nial can be used to build such Web applications. There are several issues that arise in attempting to build such applications using CGI-Nial. First, the flow of control between the pages has to be managed. Nial Systems Limited has built sample applications using two different ways to handle the passing of control between pages. A discussion of these techniques is given below. Second, techniques have to be found to handle data flow between pages. We have experimented with two approaches for data flow: using hidden input variables in the forms, and achieving persistent data using temporary Nial data files. The former is reasonable for scalar data items, but complicates the design of the forms. The latter is necessary if the data objects are large because of arbitrary space limitations within Web servers.

Explicit Control Flow

In the Business Rules demo and the Table Query demo accessible on the NSL home page at www.nial.com, the control of flow is centrally managed. The *action* associated with each form is a single Nial definition file. It loads all the required definitions, and selects which Nial form file to process based on a "FROM" variable that is encoded in each page as a hidden variable. This approach allows the control flow management code to decide between alternatives based on data values passed from the previous form. Since a single Nial definition file is used, it can be replaced by a Nial workspace with a Latent expression. This provides slightly faster processing and privacy of the application code. For this approach the calls to the routines that process the Nial form files and the templates is included in the central definition file.

Implicit Control Flow

In the Bug List application, there is no central control. Each Nial form file has either links to other pages or has explicit ACTION values that trigger the next page. In this application, since no choice points are needed, the flow of control can be handled by each form setting its action to be the next logical form. For

simple applications with only one or two pages this is a reasonable approach. One advantage of using Nial forms directly is that the processing of the form and the template are done automatically.

Data Flow by Hidden Variables

In the two demos accessible from the NSL home page, the data computed in the processing of one form is passed to the next form by storing it in hidden input variables in the form. This is a mechanism provided by HTML and CGI to pass information between the client and server processes. The values are passed in either GET mode or POST mode. In GET mode, they are visible in the command line argument that is displayed by a browser during a client request. In POST mode they are hidden from view. The biggest drawback of using hidden variables for data flow is the arbitrary size limitation imposed by some Web Browsers and the computational effort to parse arbitrarily long parameter lists.

Data Flow by Persistent Files

In the Bug List application the textual information stored to describe a problem can be of arbitrary size. We found it convenient to provide a mechanism to have one Nial form file store a set of variables when processed and to have a subsequent form file retrieve them when processed. Two routines to do this are provided. When the data is extracted the temporary files are deleted.