Blend Then Serve: A Recipe for Creating Dynamic Content

Mitchell Friedman
Kipp Jones
Steve Moore
Kanir Pandya

HARBINGER CORPORATION

Abstract

The World Wide Web has spread rapidly. Part of its success has been the ease and power of HTML for creation of 'static' content. However, the Web continues to move from static HTML pages to an ever increasing amount of 'dynamic' content. This content is produced by the combination of information from many disparate sources into a logical view. These sources can include databases, existing document repositories, system resources, enterprise systems, and timely information such as stock quotes.

Where static pages were relatively easy to create and maintain, dynamic pages have proven more difficult. Many solutions have been devised over the past several years to make access to these various sources of information from a Web server easier. These solutions range from simple scripting languages to full blown development and operating environments.

We present one solution which provides many advantages over other currently available technologies. This solution provides powerful capabilities and programming constructs, while maintaining a simple interface for novice users. We call it the ***blender***.

Introduction

The Web has many strong characteristics which have contributed to its success. The low entry barrier for publishing content along with the relative simplicity of HTML has enabled vast amounts of information to become available, while the cross-platform access to this multimedia content has helped propel the Web to its near ubiquitous penetration. This new medium has inspired a number of new tools, languages, formats, protocols, and start-up companies, all with the intent to provide universal access to information. What has become more apparent over time is that companies' existing information is not always easy to get to from the Web.

Data by the terabyte has been amassed by nearly every company during the years of office automation. Not all of this information appears useful at first glance, but many companies are learning how to use old data for new purposes. By providing online access to mounds of information, new applications can be developed, statistical analysis done, internal access to corporate information provided to all employees if appropriate, and information flow increased. New information is generated at an alarming rate, as systems become interconnected, this data can be passed to the appropriate system, stored, or immediately analyzed, or it can cause further processing to be instigated.

Access to this information is only the first obstacle faced by systems integrators looking to use the Web as a solution. Beyond access lies the formidable challenge of presenting the information in a way which takes advantage of this new medium called the Web. Organizing and presenting this information as static HTML, created or converted once, is hard enough. Doing the same for real-time dynamic content is a challenge not many people have experience solving.

Indeed, many companies develop custom applications for each and every type of information they would like access to via the Web. And, while this may be an optimal solution for a limited set of problems, the difficulty in maintaining these many disparate solutions soon becomes an undesirable burden. In addition, tom maximize success, the Web designer must also be a sophisticated programmer to handle real-time requests, as well as knowledgeable of existing enterprise systems. This is not a likely skill set.

Our approach was to add another layer between the Web and our access to 'dynamic' data. This is not a new approach, indeed the entire Common Gateway Interface [CGI] is based on this notion. Our intent was to build up this layer, to create a more robust method for connecting to various data sources. Not unlike other companies, we desired access to our enterprise database servers, the ability to connect to our mainframe computers, and an ability to interconnect our various Unix boxes, as well as providing a method for our customer's to create and maintain Web accessible catalogs for online ordering and electronic commerce. Not only did we need a broad range of capabilities from this layer for connectivity, we also needed a method for using this which matched the various skills and desires of our Web and software developers.

We present our solution - an implementation of a token expansion, interpretive language. This language combined with an extensible, open, persistent process is the basis for what we refer to as the ***blender***. The name depicts the purpose of the system, to take various inputs, process them appropriately, and return a finished product. Blender is not only a language, but also a scripting environment. Blender allows Web site creators to develop dynamic HTML pages. Applications typically consists of active server pages with some preprocessing directives. This allows the content developer to work with the familiar language -- HTML- and insert scripting commands where applicable.

This paper purports to describe and document the usage and workings of the ***blender*** library (***blenderlib***) and the ***blender*** driver daemon (***blenderd***) developed as part of the Instant Net Presence (INP) project for

Harbinger Net Services (HNS), between September 1995 and the present. It also has some discussion of a small CGI program called CGIBlend, which is used as part of the ***blender*** system.

What follows directly is a short foray into the history of the blender and some related work by others in the field. Section 3 then describes the ***blender*** language, section 4 an architectural and technical breakdown of the ***blender*** system. Section 5 describes several enhancements and extensions currently in development. We end with our conclusions and acknowledgments.

Development Perspective

How it began

The original impetus of the blender project was to create the look and feel of a set of pages separate from their content. The separation of the content from the layout allowed a Web designer to change the layout or even the technology itself and the content would follow along.

The software development team needed their Web Designers to create and maintain the templates for the content, however this would be exceedingly difficult if they were buried in Perl [Perl] or even Java [Java]. In addition, we could see the need to provide not just content statically blended but also dynamic pages combining information from any data source including completely proprietary systems not likely to have a specialized Web tool already available. Lastly, we were unsure of our target platform and database so we knew it had to be at least somewhat cross-platform capable.

There were many tools available but none fit our needs:

- Available right away with our chosen set of features.
- Able to be included in our products (i.e. not pay royalties)
- Any database or platform.
- Small enough to bundle in our Windows client software.
- Simple enough for our Web Designers to work with.
- Fully extensible on our schedule.
- Rapid and dedicated maintenance support

We had no interest or time to create a new language, our intention was to do some investigation and purchase a solution. Some of the tools met most of these needs but were not truly available. Most only supported database connectivity. Some, only simple HTML manipulation logic. None were extensible enough to meet our needs.

To fill our immediate needs, (November, 1995) we created a base language using the Microsoft Visual C++ compiler, using only iostreams and standard C functions and types. It was an interpretive token-expansion language whose default behavior was to pass a character through unchanged. Upon initialization, Blend classes registered as semantic handlers for named routines. When a token was reached, the relevant blend class was called. These initial classes included blend string variables as well as the following blend functions: $if, $else, $endif, $not, $equal, $date, $upcase. These blend classes were quickly ported to GNU C++ under Solaris (December, 1995) and proved easily maintainable.

A simple CGI program (cgiblend) was created which would take its current environment and the HTTP post information, package it up, and send it to a persistent server (blenderd). At this time we started using the RogueWave [RogueWave] class libraries as a cross-platform, cross-database, TCP/IP solution.

The blenderd (using RogueWave) would add all the name-value pairs to the blender as blend string variables, open up a file pointed to by the variable $blender , and attach its output to an ostream tied to the TCP/IP socket. The output would be sent immediately to cgiblend, through the Web server and back to the browser. At this point, the blender was creating simple and powerful dynamic Web content.

A set of database tokens ($find, $insert, $update, $remove) were created and tested on NT using ODBC to Microsoft SQL Server and on Solaris using Sybase. An IPC mechanism ($msg) was implemented. At this point (January, 1996), the blender could do everything we needed on both NT and Solaris, as well as being extended easily and indefinitely.

Related Work

A variety of methods have been devised to connect the Web to extant data sources. Many times, a dedicated program is written to take care of specific needs. This program, in general, is invoked via the Common Gateway Interface provided by all modern Web servers. The idea of a 'common' gateway goes a long way to providing the type of access desired, but falls short for full enterprise, mission critical applications.

Several problems with standalone CGI applications are readily apparent. These include:

- the performance 'hit' taken to startup each CGI process upon request,
- the difficulty of maintaining state,
- the cost of establishing connections to databases,
- maintenance of cgi-bin directories,
- creation and maintenance of cgi applications in every language known to computer,
- inability to run distributed applications
- inability for web content devlopers to take advantage of dynamic Web based data

The ***blender*** system attempts to solve these problems and several other specifics, as discussed below. Other systems such as Netscape's server side JavaScript [JavaScript] in conjunction with an application API such as NSAPI [NSAPI] provide many of the same advantages but lack the extensibility we will describe in the ***blender***.

Several systems such as PowerBuilder [PB], provide a vast array of capabilities for the savvy developer, but are specific to database applications. Other systems such as NeoSoft's NeoWebScript [NEOSOFT] implement much the same functionality as the blender, but do so within the HTTP server itself, and in this case through the use of TCL. In fact Microsoft's server-side ActiveX scripting [Denali] environment provides some of the features provided by ***blender***. We will continue to monitor the progress of other systems such as these to determine if an advantage would be gained by switching to one of these other technologies. In the meantime, we feel the ***blender*** provides a high level of maintainability and flexibility for our ongoing projects.

The *Blender* Language

Definitions

Before we get too far into the discussion of the blend language, we present some definitions to help the reader follow along.

Blend language:

an interpretive token based command language whose default action is to pass a character thru unchanged. Contains basic string manipulation, arrays, math subroutines, database access, network connectivity, system level call capability, and general programming constructs.

Blender:

Generally used to describe the blend system, but specifically refers to a library (blendlib) which consists of the scanning/parsing routines, base blend classes, and the various blend extensions like network and database. The ***blender*** library has been used in both Windows based client software (INP) as well as Unix and NT based server processes (***blenderd***).

Blenderd:

A multi-threaded persistent process (daemon/service) that listens on a TCP/IP socket awaiting connections from the network. In general, these connections are received via a CGI program (cgiblend) or an NSAPI program (nsapiblend) when a request is made to a properly configured Web server.

"blend file":

A blend file is simply a file which contains some tokens which you wish to be processed by the ***blender***. On the server side of things (i.e. for Web developers), these files are placed in a specified directory structure where the ***blender*** will know to look. These files represent the 'template' from which the HTML will be generated from.

The blend language

The ***blender*** language evolved from a simple token expansion language to a general purpose language. Many of the common programming constructs such as conditional statments, loops, and functions are now part of the core language.

Initially, when the ***blender*** language was young, it was enough to have our own home grown scanner/parser for interpreting the incoming blend files. This, however, is no longer true. The complexity of the applications which are being built using the ***blender*** technology are fast outpacing the initial implementation's ability to support the developers.

To combat this problem, we have formally defined the language using a modified BNF notation, a portion of which is shown below. With the BNF in hand, we are now able to take advantage of both lex and yacc for our

parsing and scanning routines. Figure 1 shows a sample BNF grammar for the blender language.

```
statements : statements statement | statement ;
statement : while_statement | eq_statement | '$' FUNCTION '(' expression ')' ;
while_statement : '$' WHILE '(' expression ')' statements '$' ENDWHILE ;
eq_statement : '$' EQ '(' expression ',' expression ')' ;
expression : statement | name | name ',' expression ;
name : NAME ;
```

Figure 1. BNF for *blender* language

Blender's most powerful feature is this interpretive token based command language that lets the Web Site developer actually insert small programs in the HTML/blend file side. These program snippets can be changed on the fly by the Web Site developer without recompiling, and the blend code to be executed can come from anywhere - not just the HTML/blend file. In other words, not only can "replacement" fields be defined which a C/C++ program can replace, and the Web Site developer can move around at will, but in fact the data itself can be formatted and parsed in a programmatic way.

For example, say there is a piece of data called, "NUMBER" that can be retrieved by using some back-end API call. Upon retrieval of the value from the back-end, ***Blender's*** command language allows an HTML page developer to simply include that value in the outgoing stream - i.e.

```
<HTML>
...
You have accessed this page $number times.
...
</HTML>
```

But it also allows more flexible, conditional embedded dynamic output - for example:

```
<HTML>
...
<H1>Hello $HTTP_USER </H1>
$if($gt($number,1000))
     You have accessed this site $number times which is way too much!
$else
     You have only accessed this $number times - keep in touch!
$endif

Current Local time is : $CDate(0)  $TZ
...
</HTML>
```

These are the simplest of examples but illustrate the point that ***Blender*** has a mixture of unrestricted C++/C access capabilities plus a command language that you can embed in the HTML which allows the Web site developer to take bits of data which are programmatically generated and conditionally generate different HTML output in a programmatic way, and change it on the fly without getting into the C/C++ side and recompiling etc.

To help explain the difference and the relationship between CGI programming and Blend programming, we describe it as:

CGI programs are 90% program and 10% HTML, while blend files are 10%program and 90% HTML. Much of the programming burden has been offloaded to the blender itself and removed from the Web developer's sight.

The ***Blender*** Library is a set of C++ classes in use cross-platform on Windows 95/NT(Visual C++) and Sun. It implemented an interpreted token-expansion based language. Besides being highly recursive it is also highly extensible. Currently three extensions are in place:

- INP Based Extensions using MFC (Windows 95/NT Only)
- Database Extensions using RogueWave
- Network IPC Extensions using RogueWave

The base library itself is based on the RogueWave RWCString class libraries, which provides a large amount of string manipulation capability to the core ***blender*** library.

Summary of Functions

Blender Base Functions

To complete our discussion of the blend language, we include a sampling of the functions currently understood by the blender. The core language consists of functions such as:

Token	Description
$set(<variable>,<value>,<type>)	Creates type'd variable with value
$delete	Remove variable from ***blender***
$if(<conditional>)	If conditional is true, returns Y
$else	Action taken if $if is false
$endif	Parsing continues after $if or $else
$and(<conditional>, <conditional>,...)	If all conditionals true, returns Y
$not(<conditional>)	If conditional is false, returns Y
$or(<conditional>, <conditional>, ...)	If any conditionals are true, returns Y
$eq(<conditional>,<conditional>)	Compares conditions, returns Y if true

$break	Exits $while loop
$date(<strftime string>)	Uses strftime to print formatted current date/time
$while(<conditional>)	Run until conditional is false
$endwhile	Parsing continues after $while conditional is false
$infile(<variable>)	reads the file and starts parsing of the file.
$pipe(<command>)	Tries to open system command with pipe and return output
$var(<value>)	Uses contents of value as a variable name and expands

Blender Database Extensions

To support our database needs, a set of extensions was created. This blend set is optional.

Token	Description
$find(<tablename>)	Using value of ***blender*** variables of the same name as columns, select rows.
$insert(<tablename>)	Using value of ***blender*** variables of the same name as columns in tablename, insert row
$remove(<tablename>)	Using value of ***blender*** variables of the same name as columns in tablename, delete row
$schema(<tablename>)	Return the name of the columns in tablename one at a time in variable $SchemaName<tablename>
$storedproc (<storedprocname>)	Run the stored procedure using the name of the input parameters in the stored procedure. Retrieve one row at a time like $find. Retrieve output parameters and ReturnValue upon receipt of last row.
$sql (<sqlStatement>)	pass raw sql statement to the database and retrieves the results.
$drop(<tablename>)	Remove information about tablename from ***blender***.

In addition to those listed above, the language contains the following extensions:

- *Math Extensions*
- *Blender* String Extensions
- *Blender* Vector Extensions
- *Blender* Network Extensions

...

Architecture

The ***Blender*** combines a simple token expansion based, interpreted language with a language processor. This system is usually used as a persistent multi-threaded process. It is implemented in C++ and its base implementation is currently in use cross-platform on Windows 95, Windows NT, Sun Solaris, and HP-UX. In general, a file is read into the ***blender*** and comes out the other side with various expansions. The ***Blender*** knows very little about HTML. In fact the base ***blender*** knows nothing about inter-process communication nor database access.

Blenderd is a multi-threaded process which implements the ***Blender*** library. It has the ability to run in a batch mode. ***Blenderd*** is also a daemon which is intended to work as a secondary tier server for Web based requests. A request from a browser causes the invocation of CGIBlend. Upon receiving a request from CGIBlend, ***Blenderd*** uses the ***Blender*** to expand the selected blend file and send the output back to CGIBlend, which in turn passes this back through the Web server to the original browser which initiated the request.

A 'blend' file is specified as part of a request. The ***blenderd*** processes the file, expanding and processing tokens it recognizes, passing through data which it does not recognize.

CGIBlend/NSAPIBlend are small programs responsible for connecting a Web server to ***Blenderd***. They package up all the environmental variables including the ones built into the run-time environment, the Web server and the browser. It also packages up any additional name/value pairs sent as part of an HTTP post. It sends back to the Web server anything received from ***Blenderd***.

Blenderd was constructed using an application framework that was designed and implemented by the authors. The design strategy behind the framework was to provide a set of components for facilitating client/server development in a cross-platform environment. The framework provides all of the interfaces and mechanisms necessary for controlling and reporting application state and behavior. The basic elements of the framework is a collection of recurring design patterns that describes problems that occur over and over again. Collectively, these patterns provide the core solution to the application problem domain.

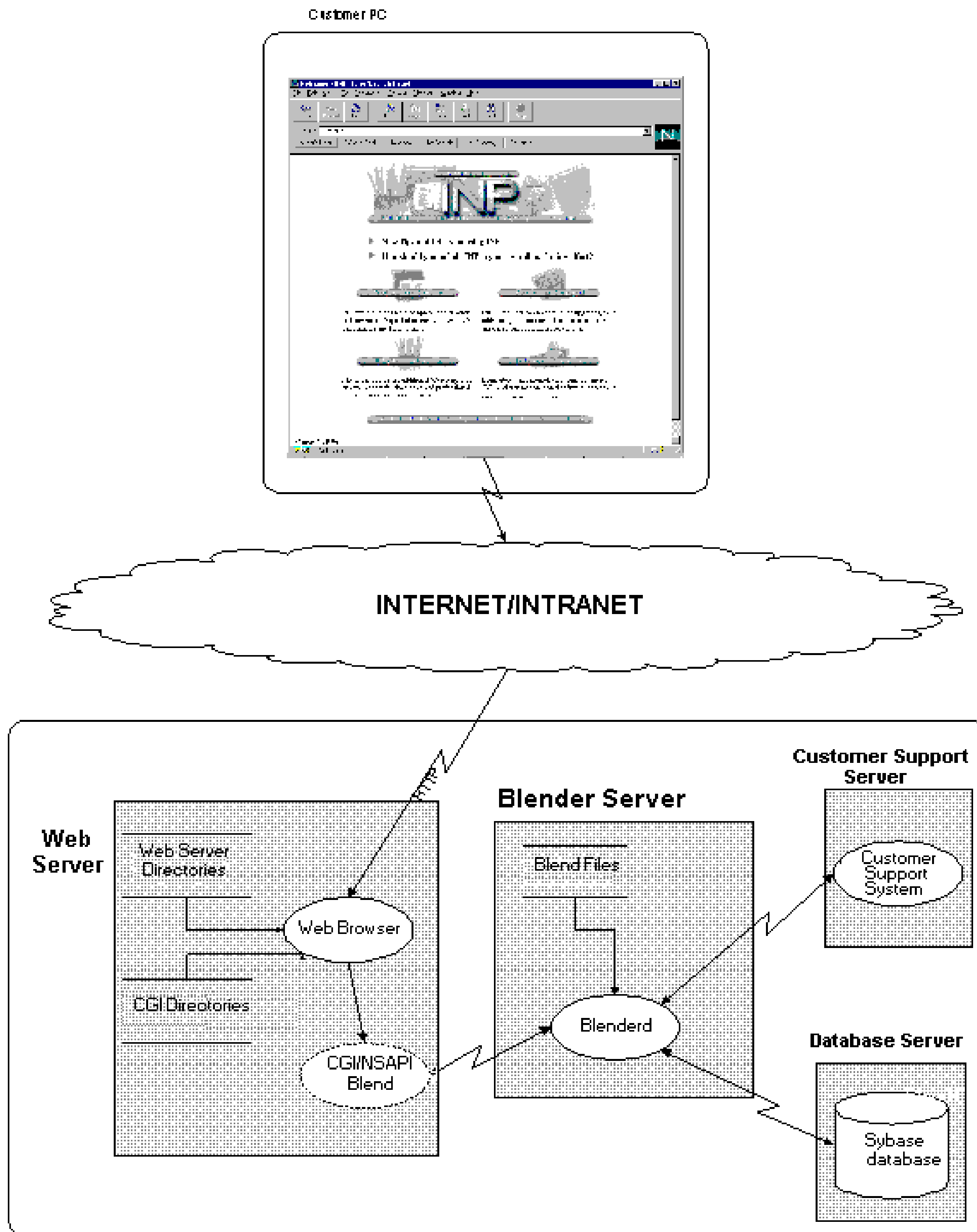


Figure 2

The usage of the framework is driven by the object-oriented data abstraction mechanisms provided by the framework architecture. Customizations and modifications are accomplished through data abstraction mechanisms that are prevailed upon the developer by the framework. The architecture provides the flexibility

needed to completely define the problem solution without compromising the behavioral properties of the framework.

Blender provides a set of abstract base classes contain as the necessary attributes and operations for ensuring proper usage. The goal was to provide a facility that could be easily adapted and extended to support additional application domains. Base class generalizations are also provided for support of cross-platform development. The intent is to provide a set of abstractions that are localized and easily modified without affecting the application domain. Blender language extensions and additional token processing can be accomplished through base class specializations. The base class definitions provide the operations and state information for runtime control. Moreover, the base classes provide interface descriptions that are used to define the syntax, semantics, runtime behavior, and entity relationships as needed to extend the language.

CGIBlend / NSAPIBlend

CGIBlend is a small program written in C. Currently it is compiled under Sun using the GNU g++ compiler and on Windows NT using the MS compiler. It is intended to be called by a Httpd server through the CGI facility and has been tested with various servers including NCSA, Apache and NetScape, and MS Internet Information Server (IIS).

CGIblend connects to ***Blenderd*** and sends the environmental and data name/value pairs separated by new-lines. It then waits for return data. Anything retrieved from ***Blenderd*** is immediately written to its out stream, which is sent back to the Httpd and on to the browser.

The NSAPI program is essentially same as cgiblend, but is called directly from the NetScape server as part of a shared object rather than being 'exec'd. The operation of the two programs is very similar but each provides advantages over the other.

For example, given the simple HTML form presented below, we see the environment variables that will be sent to the ***blender*** via CGIBlend.

Html Request:

```
<form method="POST" action="/cgi-bin/cgiblend">
<b>Select Blender File</b>
<INPUT name="blender"
size=20 value="">
<input type="submit" name="BlendSubmit"
value="Blend">
<input type="reset" value="Reset">
</form>
```

Given the above form, a condensed version of the available blend variables is shown below:

Environment:

```
HTTP_HOST=blend.harbinger.net
```

```
HTTP_REFERER=http://blend.harbinger.net/cgi-bin/cgiblend?blender
=index.blendSCRIPT_NAME=/cgi-bin/cgiblend
HTTP_ACCEPT=image/gif, image/x-xbitmap, image/jpeg,image/pjpeg, */*
HTTP_USER_AGENT=Mozilla/2.0 (WinNT; I)
CONTENT_LENGTH=39
...
SCRIPT_NAME=/cgi-bin/cgiblend
```

Input:

```
blender=testing.blend
BlendSubmit=Blend
```

When a browser performs a GET with an additional request such as http://www.harb.net/ /cgiblend?***blender***=doit.blend, the request is expanded into the environmental variable QUERY_STRING="***blender***=doit.blend". This special token, as with HTTP_COOKIE are automatically parsed to create separate blend variables for each of the name/value pairs, allowing any of these values to be easily accessed from within blend files.

Blenderd

Blenderd is the primary blender driver program which is implemented on top of several Harbinger created general C++ classes, HCObject and HCVserver. It is written in C++ and has been in use on Windows 95/NT, Sun Solaris, and HP-UX systems. It uses RogueWave's C++ class libraries, currently utilizing Tools.h++, dbTools.h++ and net.h++ to achieve much of its cross-platform capabilities. It performs both database and interprocess-messaging based communication primarily using the ***blender*** library.

Blenderd can be configured to open up a number of database connections during startup. After initialization, it listens on a specified TCP/IP port for a CGIBlend request. Upon receiving a new request it performs the following:

1. Starts a new thread.
2. Initializes a blend from the ***Blender*** library, including Net and Database ***Blender*** extensions.
3. Retrieves all the name/value pairs until a blank newline is received.
4. Parses QUERY_STRING and expands into name/value pairs.
5. Parses HTTP_COOKIE and expands into name/value pairs.
6. Parses name/value pairs and adds them to the ***Blender*** as string variables.
7. Looks for the variable named "***BLENDER***" and tries to open the specified blend file. If it fails, ***Blenderd*** will use a specified missing.blend file to present an error to the requestor.
8. Attaches a RogueWave portal stream to its socket and starts up a blend. As the ***Blender*** writes out its expansions, these will be immediately sent back to CGIBlend.
9. Ends the stream.
10. Closes the socket.
11. Shuts down the Thread.

An Electronic Commerce Example [WebPO-Peachtree]

Below, we provide an example of how the blender technology has been put to use. This is a real world example of how we have been able to use this tool.

A recent offering by Peachtree Software and Harbinger, includes the capability for creating an online Web based catalog and ordering system. Orders placed via the Web are routed back to the Web site creator for integration into their Peachtree Accounting system. This system integration was accomplished through the joint efforts of the two companies, and will save re-keying of orders placed over the Web.

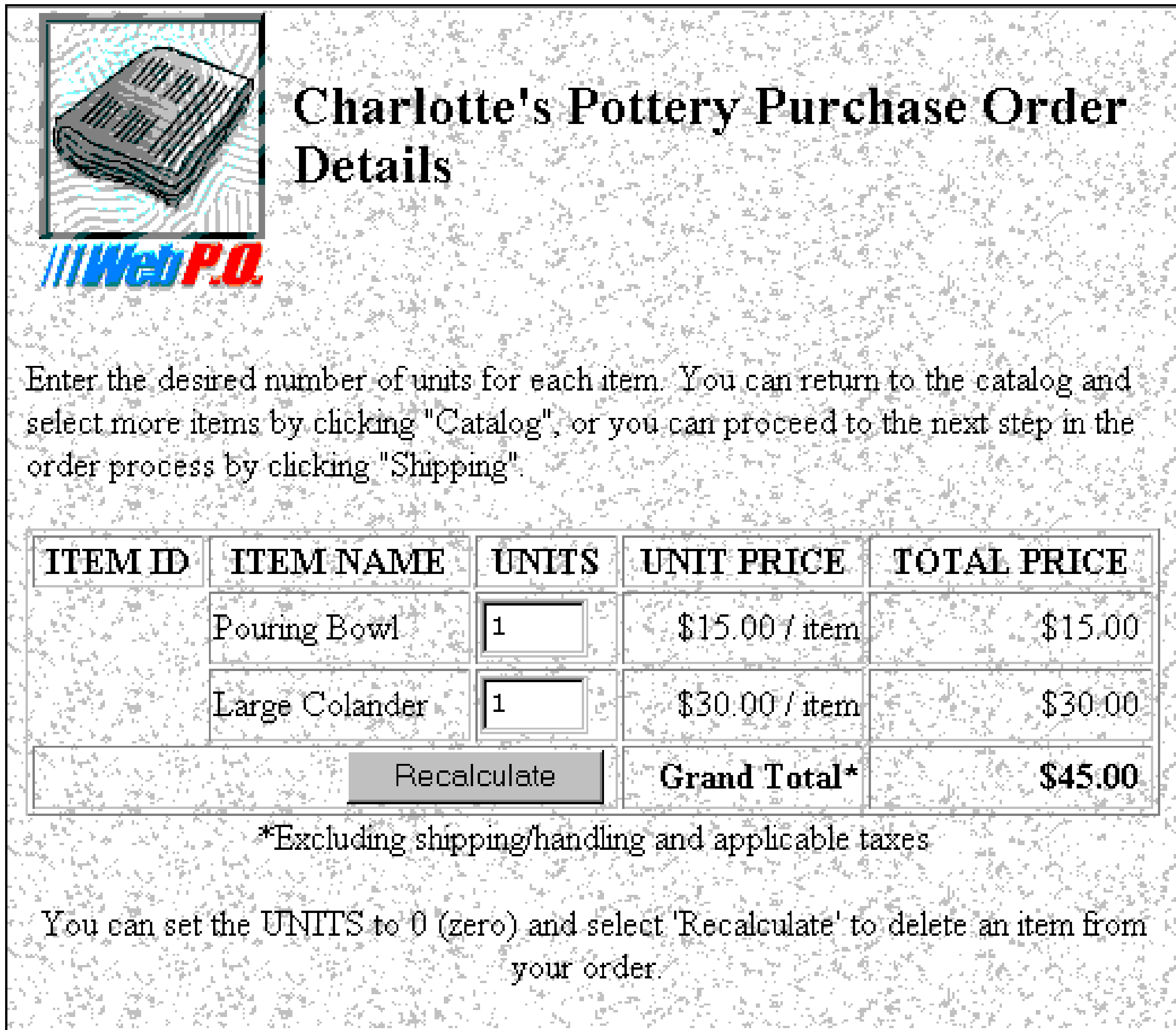
To enable the online ordering and catalog capabilities, the blender was used extensively. From 'blending' in the products for the catalog, to calculating subtotals, to formatting the message for the accounting system, to creating and handling cookies, to working with SSL. All this was accomplished with no additions to the blender language.

A small example of the blend code may help to solidify the reader's understanding. The blend code below is a partial excerpt from the WebPO process, in particular, this code calculates the totals of the order, setting up a form which allows the user to modify the quantity of items to purchase.

```
...
<!-- build the line item table -->
        $set(tempcolspan,4,int)
  <TABLE border=1 width="100%" >
    <TR>
      <TD align=center><STRONG>ITEM ID</STRONG></TD>
      <TD align=center ><STRONG>ITEM NAME</STRONG></TD>
                $if($eq(*DisplayCategory,"TRUE"))
                        <TD align=center><STRONG>CATEGORY</STRONG></TD>
                        $inc(tempcolspan)
                $endif
      <TD align=center><STRONG>UNITS</STRONG></TD>
      <TD align=center><STRONG>UNIT PRICE</STRONG></TD>
      <TD align=center><STRONG>TOTAL PRICE</STRONG></TD>
    </TR>
        <!-- loop through selected products for display purposes -->
        $set(loop,0,int)
        $set(bigTotal,0,int)
        $while($ne(*PCount,*loop))
                $if($var(ItemUnits$loop$))
                        $set(aVal,$var(ItemUnits$loop$))
                $else
                        $set(aVal,0)
                $endif
                $set(aVal,$fmt("%.0f",$aVal,"double"))
                $set(tempCost,$fmt("%.2f",$var(ItemCost$loop$),"double"))
                $set(aTotal,$mult($aVal,$tempCost$))
                <TR>
                        <TD>$var(ItemCode$loop$) </td>
                        <td>$var(ItemName$loop$) </td>
                        $if($eq(*DisplayCategory,"TRUE"))
                                <TD>$var(ItemCategory$loop$)</TD>
                        $endif
                        <TD>
                        <INPUT TYPE="text" NAME="ItemUnits$loop$" VALUE="$aVal" SIZE="5">
                        </TD>
                        <td align="right">
                        $set(tempCost,$fmt("%.2f",$var(ItemCost$loop$),"double"))
                        $var(ItemCost$loop$)
```

```
                              $if($var(ItemUnitOfMeasure$loop$)) / $var(ItemUnitOfMeasure$loop$)$endif
$
</td>
                              <td align="right">$Currency$$fmt("%.2f",$aTotal$,"double") </td>
                              </TR>
                              $set(bigTotal,$add($aTotal,$bigTotal))
                              $endif
                    $inc(loop)
          $endwhile
...
</table>
```

This blend code will produce something like that shown below:



This example shows the blender in action to help enable electronic commerce via the Web. The blender was not only used to create the Web interface, but also it:

- Created the catalog
- Stored the order
- Formatted the purchase order in the vendor's choice of Peachtree, comma delimited, EDI-850, etc.
- Mailed the order to the vendor

- Sent a confirmation to the purchaser
- Allowed SSL entered data to be accessed securely by the vendor

Future Directions

Extensions

The blender system continues to go through revisions and extensions. The ease of use has made it a useful product within our company, being used for several products and services currently in development. To fully support this development, we are creating a more configurable startup of the server.

This configuration support will allow the inclusion/exclusion of extension 'packages' at startup time simply by modifying a configuration file. This capability will allow individual blenders to only contain the specific functionality needed for the task at hand. We are modeling this behavior after the Apache [Apache] 'module' style of configuration, and will take advantage of shared objects/dll's.

Other extensions include:

- Creation of base blender/blenderd, not using RogueWave.
- Enable Web-based EDI (Electronic Data Interchange) using ASC X12 and Edifact.
- NSAPI/ISAPI version of cgiblend.
- Port to additional systems/platforms including SGI and AIX.
- Port to additional databases including Oracle and Informix.

Performance

To increase the performance of the blender, we are in the process of tweaking the underlying blender lib to eliminate as much overhead as possible. Additionally, we are enhancing the server based code to include such things as blend caching, connection affinity, and look-ahead blend file loading. We expect the caching strategy to have the most immediate impact, eliminating initial load times for high active content.

Conclusion

The ***blender*** system provides easy access to many sources of data for creating dynamic Web content. This system is based on sound architectural and object oriented ideals, providing a solid platform for future growth.

The most powerful 'feature' of the ***blender*** is the ability to extend the language and the interfaces to the current functionality. Indeed, we have added interfaces to such things as XIPC, network connectivity, system calls, database access (from MS SQL Server, to Sybase, to Progress via ODBC), extensive string manipulation, and a set of math functions.

This ability to 'enhance' the ***blender*** by extending the language is also a potential problem as we continue to use and refine the tool. As it grows, we need to keep the core ***blender*** maintainable, and allow the inclusion/ exclusion of functionality on an as-needed basis. This will keep the 'foot print' manageable, yet allow specific

applications to take advantage of the extensible nature of the ***blender***.

One challenge for the ***blender*** is to increase the performance and throughput capabilities. If the ***blender*** does not provide advantages in this arena, it will be sure to fail. The performance issues should not be insurmountable, as the ***blender*** starts with many advantages over traditional CGI based programs; persistent process, persistent database connections, relocatable object, and light-weight threading all work towards speeding the process up.

Acknowledgements

We would like to acknowledge Jeff Garbers for starting this project and providing the initial implementation. A special thanks goes to Kendall Ware for providing the graphics for blender. We would also like to thank all of the folks at Harbinger and our families for putting up with us and letting us tell the world about our work. Lastly, some of the authors would like to acknowledge Bruce Eckel [Eckel] for his contribution to our C++ abilities.

References

[Apache] Apache Users Guide
URL: http://www.apache.org/docs/

[CGI] Common Gateway Interface - original documentation.
URL: http://hoohoo.ncsa.uiuc.edu/cgi/overview.html

[Denali] Writing a Smart Server with Denali, Mary Kirtand.
URL: http://www.microsoft.com/mind/1296/denali.htm

[Eckel] Bruce Eckel, Thinking in C++. Prentice Hall, 1995

[FastCGI] FastCGI, a proposal for bettering CGI
URL: http://www.fastcgi.com/

[Java] Ken Arnold, James Gosling, The Java Programming Language. Addison-Wesley, 1996.

[JavaScript] JavaScript Authoring Guide
URL: http://home.netscape.com/eng/mozilla/Gold/handbook/javascript/index.html

[NeoSoft] NeoSoft, using TCL (Tool Command Language)and the Apache server Overview.
URL: http://www.neosoft.com/neoscript/docs/overview.html

[NSAPI] Netscape Server API URL: http://home.netscape.com/comprod/server_central/config/nsapi.html

[PB] PowerBuilder 5.0 & The Web, Curt Lang & Jeff Chow
URL: http://nt.powerbuilder.com/vol3num2/web1/powerbui.htm

[Perl] Larry Wall, Randal L. Schwartz, Programming Perl. O'Reilly & Associates, 1991.

[RogueWave] RogeWave class libraries. URL: http://www.roguewave.com/