Max Server Pages Developer’s Overview

MSP-OVW Release 1.00 2000-03-27b

Trademark Notice

Max™, Joiner™, Joiner™, Joiner 96™, Joiner 97™ are trademarks of PlugSys International LLC.

All trademarks for products discussed in this document and not named above are those of their publisher.

dBASE™, dBASE III Plus™, dBASE IV™, Visual dBASE™ are trademarks of Borland/Inprise and dBASE Inc. Clipper™, CA-Visual Objects™ and CA-Clipper™ are trademarks of Computer Associates International. Internet Information Server™, MSDOS™, Windows™, Windows 95™, Windows 98™, Windows NT™, Visual Basic™, FoxPro™, Visual FoxPro™, Front Page™, Front Page Express™, Front Page 2000™, Visual Interdev™, Visual Studio™ and SQL Server™ are trademarks of Microsoft Corporation. Blinker™ is a trademark of Blink Inc. Dreamweaver™ is a trademark of Macromedia, Inc. HomeSite™ is a trademark of Allaire Corporation. Netscape Enterprise Server™ and Netscape FastTrak Server™ and iPlanet™ are trademarks of the Netscape/AOL/Sun Alliance and iPlanet.

Read Me First:
Introducing Max Server Pages

This manual is designed to give you an overview of web development in general and Max Server Pages in particular.

How To Use This Manual

We recognize that many developers will be adopting Max Server Pages as their first or earliest experience working with web application development. So this manual will serve these parallel purposes:

- To introduce and reinforce concepts and strategies for development in the web environment (covering such constructs as HTML, Javascript, forms and query strings)
- To illustrate how Max Server Pages works.
- To demonstrate how MSP fits within new or existing web sites.

How This Manual Is Organized

Max Server Pages appeals to many audiences, so you can best address your needs according to your previous experience:

Are You New To Web Development?

- Start by reading **HTML Fundamentals** beginning on page 41.
- Expand your knowledge by continuing with **More Web Development Issues** beginning on page 75.
- Do more research. This manual provides web links to relevant material. (See **Useful Links** on page 88.)
- Obtain some books on HTML and Javascript to use as reference works.
- Then focus on the specifics of Max Server Pages with **MSP Development Fundamentals** beginning on page 23.
- Then get some ideas about configuring your development environment. (See **Creating Your Development Environment** on page 55.)

Do You Already Have Web Development Experience?

- Your first stop depends on your level of expertise. If you already have some experience, you may benefit from **More Web Development Issues** beginning on page 75. (Now that you have explored the essential concepts and features of HTML, this chapter will take you to the next level. The material includes: query strings, Javascript and security issues.)
- Focus on the specifics of Max Server Pages with **MSP Development Fundamentals** beginning on page 23.
- Then get some ideas about configuring your development environment. (See **Creating Your Development Environment** on page 55.)

Have You Installed Yet?

- When you are ready to install Max Server Pages and configure your web environment, consult The ***MSP Installation And Configuration Guide***.

Draw Your Own Map: A Quick Tour

- A chapter-by-chapter briefing appears in **Manual Structure** beginning on page 19.

Introducing Max Server Pages

What Is MSP?

If you wanted to search the Internet, you could explore many different approaches to "programming" dynamic web pages.

Most of these tools were written as if they forgot about the database during the design work. Then they paste database access on to the back end. This results in a poor experience. Too much coding and too little control.

If a development tool can't easily maintain structured information in a database and query it, the tool is too primitive. If the application development language is verbose, you waste valuable time.

Max Server Pages (MSP) was conceived from the beginning to take what is best about database development and merge it with the structure of HTML. MSP relies on the time-tested Xbase language for database access and weaves data together into web pages. Using the text formatting, graphics and hypertext, you can create web pages displaying dynamic information.

The marriage is pretty simple once you understand the Xbase model and HTML structure.

HTML Is A Simple Concept; MSP Makes It Powerful

If you've never created web pages before, this is something to work with as soon as possible. (We have provided a quick path to success later in this manual. See **Getting Started: Putting It All Together** beginning on page 73.)

Web Architecture IS Client/Server

A web application relies on a client/server architecture. The user employs a browser client to make page requests to the web server. This is performed using the Hypertext Transmission Protocol (HTTP).

The browser requests a page by sending a Uniform Resource Locator (URL) to the web server. The server loads this address. In the simplest cases, the address points to an ASCII file with hard-coded HTML text.

In this simplest case, the URL path closely parallels the physical directory structure of the web server's content. So if the web server is running on a Microsoft platform, a request to http://www.yourcompany.com/products/best.html calls a file called *best.html* that is located in *C:\web_pages\products*.

Adding Logic To A Web Site: CGI

Originally the web was intended to share long academic text material among scientists in remote sites around the world. This kind of content worked extremely well as a series of ASCII files.

But as the expectations increased, people wanted to see graphics and to make choices. If you've spent any time developing forms for computer users, you already realize that such a system needs conditional logic and loops. But these are present in HTML. Hypertext Markup Language (HTML) was purely designed to format text and provide embedded links of related material.

To extend a web site with program logic, Internet developers added a specification known as the Computer Gateway Interface (CGI). Simply described, CGI provides a way for a web server to load a script or an executable by directly requesting it as part of a URL. The CGI specification also provides for a list of server-side environment variables that CGI-compliant scripts and executables can access. These include data elements like the URL, the server's name and can potentially provide fieldnames and field values from web forms.

The most common CGI processing language is Perl. It is an interpreted language with roots in UNIX. Perl is available for Win32 as well.

Keep What You Already Have: Max Server Pages Cooperates

MSP is a technology designed to extend web sites without requiring you to abandon the work yoiu have already done. It works side-by-side with CGI scripts. So if you already have Perl handling simple web forms, you can adopt MSP to deal with more sophisticated requirements.

Making Your First MSP Web Page

Starting with MSP is easy because it uses HTML as its foundation. Here is an example web page which only uses HTML:

```
<!-- hello.html -->
<HTML>
  <HEAD>
     <TITLE>Simple Web Page</TITLE>
</HEAD>
<BODY>
  <H1>Hello World</H1>
</BODY>
</HTML>
```

This is so simple, you can type it, save it to disk and then drag it onto your web browser to view it.

Your First MSP Template: 60 seconds later

Once you have installed Max Server Pages on your web server, you can invoke this same code as an MSP template. You won't be doing that under ordinary conditions. But we just want to demonstrate that MSP is not some strange foreign technology. If you learn standard HTML, you can use your skills for web development:

- Just change the filename to **`hello.msp`**. Put the file in a web server content directory granted scripting permissions and try accessing it with a web browser.
- You won't see any difference because you have not added any new features.

Your Second MSP Template (2 minutes later)

- Now add something from the world of Xbase to demonstrate the dynamic capabilities of MSP. We have saved the newly modified template as **hellodate.msp**:

```
<!-- hellodate.msp -->
<HTML>
  <HEAD>
     <TITLE>Simple Web Page</TITLE>
</HEAD>
<BODY>
  <H1>Hello World</H1>
  <P>
  Today is <%=CDOW(date())%>,
  <%=CMONTH(date())%> <%=DAY(date())%>, <%=YEAR(date())%>.
  </P>
  And the time is now <%=TIME()%>.
</BODY>
</HTML>
```

NOTE: This manual includes much more information on common web development construction. Some of the most important material focuses on how to create web forms. The introduction section is **Web Forms** on page 44..

Manual Structure

This section summarizes of all chapters in this manual. Even if you are planning totally new coding with MSP, it pays to read this chapter. It shows you how to get from a world you already know into an exciting new place.

MSP Development Fundamentals

This chapter is the core of the documentation for Max Server Pages. It explores how to structure MSP templates and how to pass data from server to browser and back again. Here's where you learn to weave together dynamic web pages using Xbase, HTML and Javascript.

Max Server Pages and Xbase

At its core, Max Server Pages is a server-side Xbase language and data access engine.

Starting with obvious uses of Max Server Pages, you can:

- add a graphical interface to existing applications
- integrate a suite of applications using a web interface
- integrate an organization's applications with the Internet
- share information among multiple sites
- make web development cost-effective
- build data-driven web sites quickly
- capture persistent data from users anywhere in the world (or in your office) to your web server
- share data of, by and for other business systems
- develop highly personalized web content by controlling what the user sees
- implement customer service on the web
- provide e-commerce capabilities

Getting Web-Wise

The above list only begins to describe the possibilities. Your experience developing Xbase or other database applications can be immediately exploited for most aspects of web development:

- string manipulation
- data type conversions
- calculations and formulas
- database structures and data persistence

Nothing good comes without some effort. Building a web application will require a new approach to user interface design and programming:

- menu systems and navigation strategies
- screen / page design (including new kinds of screen objects like pulldowns, push buttons, check boxes and radio buttons)
- integration of hypertext links
- if your application has access to the Internet, you might want to exploit information outside the organization

Max Server Pages Adds Convenience

We have added several features to classic Xbase so you can develop web applications faster. These language features appear in Max and Max Server Pages:

- **Long string delimiter**
- **String expansion**
- **Unary operators**

Max Server Pages Library

To speed up your web development effort, PlugSys is continually expanding the possibilities. Max Server Pages comes with these functions to help in processing HTML forms:

- **MSPFieldCount()** on page 69.
- **MSPFieldName()** on page 70.
- **MSPFieldValue()** on page 71.

NOTE: Be sure to visit the PlugSys web site for additions to the library of Xbase code. We invite Max Server Pages developers to contribute new code and share tips and tricks.

The User Interface: Relying On HTML

When writing an Xbase application for the web, you cannot employ traditional user interface features like @ SAY GET and READ. Instead you will rely on HTML to instruct users' browsers how to visually render your content. This guide cannot possibly provide you with an HTML education; if you are new to the web, you must learn HTML.

Easy To Get Started With HTML

The good news is that HTML is relatively simple to master. Most early learning experiences start with viewing the HTML for web sites you can find on the Internet. (With Microsoft Internet Explorer, you select View/Source. With Netscape, View/Page Source.)

Everything depends on your learning style. Some people like to view source of web pages that seem interesting or simple enough to explore. From there, you might try saving the web page to your hard drive and then make simple changes to the HTML to observe what happens.

If you are willing to take the time, we recommend that you download an evaluation copy of Dreamweaver (from Macromedia). It provides excellent ways to analyze, change and create web pages. (See **Page Design Tools** on page 103.) We believe that this product is an important part of any web developer's tools.

NOTE: If you are using Microsoft Windows as your development desktop, you will find development simpler if you make simple configuration adjustments. (See **Getting Control: File Associations** on page 56.).

The beauty of the web is that you can start simple. You can save HTML from web pages to your hard drive for later examination. Or you can create your own pages as a learning exercise.

Using a simple text editor, you can create web pages from the beginning or modify HTML you have "borrowed" from the Internet. Once you have web pages on your hard drive, just launch them in your browser:

- Most browsers will open files in your local file system.
- With most operating system configurations, your web browser will already be associated with .html and .htm files.

As you make incremental changes to your local text files, remember to refresh (or reload) your browser so that it rereads those changes.

At your first opportunity, we recommend that you start using a good HTML editor. The best will provide color coding and code completion. If you want a fast way to learn optional clauses to HTML tags, one of these editors will be your guide. (See **Text Editors** on page 102.)

Multiple Forms Per Page

A web page can contain multiple forms. This is especially useful when you want to provide something similar to an Xbase **BROWSE**. Just make sure that each of the forms has a unique **name**. You can code templates that dynamically generate form names (and populate query strings and provide initial values for fields):

```
<HTML>
  <HEAD>
     <TITLE>Demo of generated multiple forms</TITLE>
  </HEAD>
<BODY>
<%
select it
do while ! EOF()
     // to output quotes, use chr(39) for " and chr(34) for ’

     <!-- next line uses field values to generate unique form names -->
     ? [<FORM name=] + chr(39) + [cust_] + it->code + CHR(39) + ;
     [ action=] + CHR(39) + ;
     [/msppages/formhandler.msp?custid=] + it->code + CHR(39) + [>]

     ? [<INPUT type="text" name="namelast" value="] + it->namelast + [">]
     ? [<INPUT type="submit"]

     </FORM>
     skip
  enddo
%>
</BODY>
</HTML>
```

NOTE: If you are new to HTML forms, see the introductory material in **Web Forms** on page 44.

NOTE: The above example uses "classic Xbase" string concatenation. But we recommend that you consider using the Max **`StrExpand()`** function with the substitution meta characters **`{$}`** and long string delimeters **`{{`** and **`}}`**. The fragment below is a cleaner, way of handling the same code as above.

```
<% LongString := {{<FORM name="cust_{$}"
action="/msppages/formhandler.msp?custid={$}">

<INPUT type="text" name="namelast" value="{$}">
<INPUT type="submit"}}

? strexpand( LongString, it->code, it->code, it->namelast) %>
```

For more information, consult the ***Max Language Reference***.

Commenting MSP Pages

There are multiple layers to any web development project. So when you comment your work, you should be thinking about which layer the comment is relevant to:

- Is the comment that you will be comfortable exposing to the user? Although comments don't appear on the displayed web pages, browsers allow clever users to view the source.
- Is the comment something you want to see for development, testing and debugging only? Use HTML comments. But then remember to remove them before going into production. Or better: use MSP to include a debug variable and make output of certain HTML comments conditional. This allows you to :
  Is the comment purely for viewing by programmers working on server-side templates? Then use classic Xbase comment characters like *, // and && within MSP code fragments. These comments are not passed from the MSP engine to the resulting web pages:

```
<%
myvar = 5
// open the DBF table file:
USE mytable ORDER idno ALIAS myt IN 0
* now find the customer ID:
SEEK "CRUZ"    && customer name is capitalized
%>
```

Max Server Pages Development Techniques

Embedding Xbase Code In MSP

The idea of Max Server Pages is to use the HTML web page as the governing structure for your web development. This allows you to use the same tools as HTML specialists when designing the look and feel of web pages. Whether you are working in a color-coded HTML text editor or a visual page designer, you can easily move from designing the look of pages to coding the data access.

Because MSP development involves coding within an ASCII file that will be used to generate dynamic content on request, we call the units you work ***templates***. This best describes the idea that you are not building the pages themselves, but describing the look and feel, the data and rules necessary to generate the actual pages at runtime.

MSP Templates Evaluate Xbase Code And Return Values

No matter how complex your embedded Xbase code, its ultimate goal is to output values that become a part of the host web page. Following is a simple template for a page that displays information about the date and time:

```
<HTML>

<HEAD>
<TITLE>Date and time from Max Server Pages</TITLE>
</HEAD>

<BODY>
<H1>Welcome to Max Server Pages</H1>

The current day of week is <% ? CDOW( DATE() ) %> and the full date is
<% ? STR( day( date() ) )+ " " + CMONTH( date() ) +" "+ YEAR( date() ) %>
<HR>
Thank you for visiting this web page.
</BODY>
</HTML>
```

NOTE: MSP evaluates code ***on the server*** and ***when last generated***. So the date and time in the date/time example will not dynamically update at the browser. When you need runtime functionality, you should use Javascript, the client-side language. (See **Are you new to Javascript? Introductory information about Javascript appears elsewhere in this manual. (See Javascript Overview on page 70.)** on page 93.)

The above sample is deliberately simple, but you can use most Max language constructs (operators, functions, system variables and commands) within the <% %> markers. Just remember these rules:

- The only valid Xbase input/output commands permitted in MSP blocks are the **?** and **PRINT** commands. Don't use such things as **SET CONSOLE**, **SET ALTERNATE**, **SET PRINTER**, **@ SAY** and **@ GET**.
- The Xbase block embedded within Max Server Pages templates should return values for display on the web page.
- There are no live connections between a web page and the browser once the page has been transmitted to the browser. Design your applications so that the browser sends data or "messages" back to the server. Use query strings or hidden fields to transmit information when a web page is submitted. (For more information, see **Hidden fields** on page 65 and **Query Strings** on page 69.)

The Scope Of An MSP Template

Here are the simple facts about the architecture of an MSP-based application:

- Code within an MSP template is implicitly scoped to that template.
- The visibility and lifetime of all data in an MSP system begin and end with the existence of a given template-generated web page.
- Data can be saved when a user submits a form or when a template redirects to a URL for a "save template." (See **Form Handler Templates** on page 90.)
- Data can be passed from one web page to another using hidden fields or query strings. (See **Query Strings** on page 69.)

Embedding Xbase Fragments

One benefit of the implicit scoping is that you can break the branches of Xbase control structures into separate code fragments:

```
<% IF balance < 100 %>
  <H2>Low credit balance of <% ? transform(balance,"999,999.99") %><H2>
<% ELSE %>
  <I>You have exceeded $100</I><BR>
<% ENDIF %>
```

The same code can be done as one continuous MSP block:

```
<% IF balance < 100
  ? [<H2>Low credit balance balance of ] +
     transform(balance,"999,999.99")
ELSE
  ? "<I>You have exceeded $100</I><BR>"
ENDIF %>
```

Locating Procedures And Functions In A Template

Place Procedures and Functions At The End Of The Template

Many people assume that the **RETURN** statement is a terminator for a function or procedure. But in reality, Xbase products only "realize" that a routine has ended when it sees another procedure or function following.

If you place a full procedure or function at the top of a template, MSP will assume that everything following it is a part of the routine. And as a result, the remainder of the template will not be displayed:

```
<html>
<body>
This is section belongs to the body.
<%
use countries query [select * from countries where country = ] +
SQLFormat(CountryCode)
if !used()
  FatalError(SQLErrorMessage())
endif

Function SendEmailNotification(email)

SendMail(email)
return
%>

<h1>THIS WILL NEVER BE DISPLAYED</h1>

</body>
</html>
```

Instead, structure your template like this:

```
<html>
<body>
This is section belongs to the body.
<%
use countries query [select * from countries where country = ] +
SQLFormat(CountryCode)
if !used()
  FatalError(SQLErrorMessage())
endif
%>

<h1>THIS WILL BE DISPLAYED</h1>

</body>
</html>

<%
Function SendEmailNotification(email)

SendMail(email)
return
%>
```

Form Handler Templates

For every web form you create, you will need to create a form handler template. We generally describe these as ***submit templates*** or ***save templates***. Handler templates typically perform these tasks:

- accepts passed fieldnames and their values
- saves values to the database
- calls additional pages
- performs necessary calculations
- reacts to business rules
- acknowledges the submission by transmitting more HTML to the user's browser
- requests confirmation
- performs server-side validation (and transmits errors back to the browser)

Simple Form Example

```
<%
  // set a constant for the font
  face := "Verdana, Arial, Helvetica, sans-serif" size="-1"
%>

<% // when submitted, the form handler (submit template) is called %>
<FORM ACTION="./formhandler.msp">
  <TABLE>
    <TR>
      <TD valign="top">
        <font <% ?? face %> >
        Last name: </font>
      </TD>
      <TD>
        <input type="text" name="lastname" value="<%=table->lastname%>">
      </TD>
    </TR>
    <tr>
      <td valign="top">
        <font face="Verdana,Arial,Helvetica,sans-serif">Comments
        </font>
     </td>
      <td>
        <textarea cols="40" rows="4" name="comments"><%=table->comments%>
        </textarea>
      </td>
    </tr>
    <tr>
      <td valign="top"></td>
      <td>
        <input type="checkbox" name="cb_mail" value="Y" checked>
        <font <%=face%> >Add to mailing list<br>
        <input type="checkbox" name="cb_meals" value="Y">
        Expensive meals</font>\
     </td>
    </tr>
    <tr>
      <td valign="top"></td>
      <td>
        <font <% ?? face %> >
          <input type="radio" name="rb_fruit" value="Apple">
          Apple<br>
```

```
          <input type="radio" name="rb_fruit" value="Banana">
          Banana<br>
          <input type="radio" name="rb_fruit" value="Orange">
          Orange
           </font>
        </td>
    </tr>
    <tr>
      <td valign="top">
       <font <% ?? face %> >
       Day</font>
       </td>
      <td>
        <select name="sel_days" size="1">
          <option value="0">Sunday</option>
          <option value="1">Monday</option>
          <option value="2">Tuesday</option>
          <option value="3">Wednesday</option>
          <option value="4">Thursday</option>
          <option value="5">Friday</option>
          <option value="6">Saturday</option>
        </select>
      </td>
    </tr>
    <tr align="center" valign="middle">
      <td colspan="2">
        <input type="submit" name="Submit" value="Submit">
      </td>
    </tr>
  </TABLE>
</FORM>
```

Simple Form Handler

This simple code shows how simple it is to accept passed fields:

```
<html>
<!-- formhandler.msp -->
<body>
<H1>MSP Form Handler</H1>
<H1>Listing Fields<H1><HR>
<font face="verdana,arial,helvetica,sans-serif" size="-1">
<%
   ?? "Fields received: " + ;
        LTRIM(STR(INT(MSPFieldCount()))) + "<BR>" PUREHTML

  // list out fieldnames and values
    for i = 1 to MSPFieldCount()
        ?? "<B>" + MSPFieldName(i) + "<B> = " + MSPFieldValue(i) + ;
            "<BR>" PUREHTML
    next
%>
</font>
</body>
</html>
```

Do You Need Javascript In MSP Development?

This section explores the relationship of Javascript to your MSP development effort.

NOTE: Are you new to Javascript? Introductory information about Javascript appears elsewhere in this manual. (See **Javascript Overview** on page 70.)

Javascript is ideally suited for runtime operations. And the first place where web developers encounter Javascript is when they need to handle forms. Because MSP is a server-side technology and Javascript comes along with all common browsers, use Javascript for runtime, client-side operations:

These are typical uses for client-side Javascript:

- form-level or field-level validation
- calculations
- alerts dependent on runtime values or interactions between fields
- cascading series of picklists where the user's interaction with the first picklist determines what will be displayed in the following (for example showing Software products in the first list operating systems in the second and version numbers in the third)
- runtime branching logic (for example showing "Shipping Free" when a total order value exceeds $99.)

If you've been developing Xbase applications, you are accustomed to the **`VALID`** clause, which tests for logic at the field level at runtime. Web interaction follows a client/server model. The browser requests a page and the server transmits a stream of ASCII text and bitmapped images. Once the data arrives at the browser, the server and the database are disconnected from user activity at the browser. Then when the user clicks on the Submit button on a form, the data is transmitted back to the server. This is when your MSP application evaluates the data and interacts with the database.

For efficiency, and for the sake of user comfort, it is best whenever you can to build runtime intelligence into your web pages. If there is a data integrity problem in a web form (missing field, incorrect value, etc.), the best time to handle this is at runtime. Your decision is whether to handle this at the form level (**`onSubmit`**) or at the field level (**`onBlur`**).

Javascript (Client-Side) and Max Server Pages (Server-Side): A Partnership

Max Server Pages does not conflict with Javascript in the context above because MSP is a ***server-side*** application engine. You use MSP to output HTML tags, text and data extractions. These are "manufactured" at the web server and then "shipped" to web browsers on request.

If you need Javascript code to interact with the user, that code must be part of the web page sent to the browser. You embed Javascript as a part of the text in your MSP templates.

NOTE: Are you new to Javascript? Introductory material appears in **Javascript Overview** on page 77. Also see **Javascript Links** on page 89.

Divide And Conquer

Has the user ordered more merchandise than his or her credit limit allows? Without rules that fire at runtime, no one will know whether the browser will deliver a valid order. Here is where you would plug in a Javascript function that would fire in reaction to an event (onClick or onBlur are examples of browser events).

With MSP and Javascript sharing the process, you can have great control. Here is how you might divide responsibilities so that users are aware of their credit limits and are prevented from submitting excessive orders:

On The Server Side

The user clicks on a link to an order form. The system provides some method of user identification. When the MSP form template is called, the MSP code within that form is evaluated at the server. This code checks the customer database identifying the current credit limit for the current user.

The code that does this is embedded within a template containing Xbase code segments and HTML:

```
<HTML>
<HEAD>
<script LANGUAGE="JavaScript">
<!--
// this is a Javascript function
// it gets executed on a user event once it appears on a browser
function limitOK( FormObject ) {
  var userlimit = <% * this is an MSP code segment
     USE credit ORDER customercode IN 0
     SEEK customerid
     ? credit->creditlimit // outputs the value
                           // for MSP to replace into the web page
  %>;
}
// -->
</script>
</HEAD>

<BODY>
  <FORM>
     <!-- HTML Order Form Goes Here -->
  </FORM>
</BODY>
</HTML>
```

- MSP evaluates the MSP code within the template. MSP code segments are marked with **<%** to begin and **%>** at the end.
- MSP integrates the results in place of the Xbase code segements.

The manufactured page is shipped to the browser and appears as if it was a hand-crafted web page. In the above sample, the Javascript variable **userlimit** arrives at the browser already given a specific assignment value (evaluated and output at the server side before the page is "manufactured").

NOTE: MSP gives you tremendous control over the look and organization of your pages. The quality of the HTML is your responsibility, just as the quality of your Xbase applications have always been a direct result of your thought.) If you are new to HTML, we strongly recommend that you study the topic further. Most web developers would be wise to establish a working relationship with a web-oriented visual designer. This ensures you are working to a good visual design.

This Xbase segment in the order form template is designed to embed the customer's credit limit inside the web page. This value will be compared by a Javascript function against a runtime calculation of the total order:

Why Not Xbase On The Client Side?

If Xbase is so great on the server side, why isn't there an Xbase engine for the browser too?

There is no technical barrier to implementing a client-side Xbase engine. And we would love it if users already had Xbase on every machine. But every modern browser already comes with at least one language engine-- and that language is Javascript. If you wanted to execute Xbase code at the browser, you would have to create a delivery and installation mechanism. In a client/server world, it is smarter to take control of the server because it is in your own hands. Leave the client characteristics to the browser publishers!

HTML Fundamentals

This chapter cannot possible teach you everything you should know about HTML, Javascript or other web development topics. The goal is to give you a general background. We strongly recommend that you do further study and research if you are not familiar with HTML.

But the purpose of this chapter is to give you a foundation to understand web development and to establish confidence in an experienced developer. If you have programmed before, you have the necessary skills required for web development. And if you have programmed with an Xbase product before, Max Server Pages will soon feel friendly and familiar.

HTML: From The Beginning

HTML Is A Markup Language

HTML is an acronym for *Hypertext Markup Language*. Markup languages have been around for as long as there have been computers. In the days before graphical user interfaces, markup languages were a common way to format text for printing. Early PC users will remember that even WordStar had a simple markup language. UNIX has nroff and troff.

You know you are not talking to an experienced programmer if they tell you they want to learn "HTML programming." HTML was not conceived to provide the functionality one expects from a programming language; control structures and branching logic just don't exist in HTML.

What Is HTML Designed To Do?

HTML was initially conceived as a way of publishing text. In its beginning there was no way of displaying graphics or specifying fonts. But HTML was able to describe the structure of a document effectively. Web page authors could specify heading levels much the way a database developer might have multiple heading levels in a report.

The HTML Structure

Every web page is placed within an **<HTML>** tag.

Every page should have one **<HEAD>** and one **<BODY>** section:

- The **<HEAD>** section contains the page title and specialized **<META>** tags read by search engines or web browsers. If you want Javascript functions to be generally accessible within the page in question, you place them in this section as well. For the most part, the **<HEAD>** section does not provide user-oriented "content." (The **<TITLE>** tag is the notable exception.) You might think of this section as the equivalent of the setup area of an application (with all your **SET** commands).
- The **<BODY>** section is where the user-accessible content goes. If you are converting old Xbase programs to run on the web, this is where your @ SAY and @ GET code would be replaced with HTML.

Below is the sequence of commonly occurring HTML sections:

```
<HTML>
  <HEAD>
     <TITLE>Demonstration Of Web Page Structure</TITLE>
  </HEAD>
  <BODY>
     <H1>This is a level 1 heading</H1>
     Text can appear anywhere below that heading.
     You can even press the Enter key. But web browsers only
     recognize a forced break or paragraph tag like the one
     that follows:<BR>Now this is a new line. To create a new
     paragraph, you use a different tag:
     <P>This is a new paragraph.</P>
     <H2>This is a level 2 heading</H2>
  </BODY>
</HTML>
```

General Rules About HTML

- HTML tags are enclosed within angle brackets: **<HTML>**
- Almost all HTML tags come in pairs with a start and stop tag: **<BODY>** and **</BODY>**
- Tags are not case sensitive: **<BODY>** is the same as **<body>** or **<Body>** or **<boDy>**.
- Many tags have additional clauses (optional and required). These are generally called ***attributes***. Most tag attributes also require a value:
  <IMG SRC="/images/picture.jpg">
- When specifying the value of a tag attribute, use quotes to delimit the value. You will see many web pages that omit the quotes, but this is a poor coding practice. As web development matures and web browsers become more dependent on XML, software will require more precise specifications of values. Under those conditions, quoteless code could generate errors and/or fail to transmit important data.
- HTML does not require indentation of any kind. Indenting HTML code does not change how it is displayed on the user's browser. (We recommend indentation for clarity.)
- HTML browsers do not translate your carriage returns as visible newlines on a browser. This allows you to format your HTML source conveniently for your own purposes. For browser-rendered paragraph breaks, use the **<P>** and **</P>** tags. For simple line breaks, use the **
** tag.
- There is no need in HTML for a line continuation character. You may type the tags or content anywhere as long as you retain the proper sequence of things. This means you can either use the wrapping behavior of your text editor or just press returns wherever it is convenient for you.

Web Forms

Web forms serve many of the same functions as a classic Xbase data entry form under MSDOS. Users can add, edit and insert data into the database. But web forms are also graphical; they include graphics and provide hypertext links to information from other applications and information sources on your local area network, wide area network or the worldwide Internet.

- Each form is loaded into a user's browser. One the server transmits the page containing the form, the connection to the browser is immediately terminated.
- To return data to the server, the user must submit the form. If the user chooses to close the browser, visit another site or turn off the computer, no data is transmitted back to your server.
- To submit the form your application must invoke a URL pointing to an MSP template that knows how to process the form data. This step creates a new connection from the browser to the server.

NOTE: The following information about forms and their component elements is intended to present an overview. It would be impossible to cover every attribute for every HTML tag. If you are new to web development, we hope this material is a friendly introduction. We strongly recommend that you obtain specialized books on HTML, Javascript and web development techniques.

Further information appears in this manual on HTML forms. See the following sections for additional information:

- **Multiple Forms Per Page** on page 27
- **Simple Form Example** on page 34
- **Simple Form Handler** on page 36

Form Structure

If you are new to web development, you may find it useful to open a text editor and create a sample web form. Keep it simple. After you have finished this chapter, you'll have enough information to make your first attempt. If you search web sites for more information or buy a book, you can make more advanced efforts.

NOTE: A simple example appears in **Trying Your First Form** on page 51. We encourage you to try coding something similar if you are new to the web. Also see related material in **Forms** on page 94

The <FORM> Tag

Everything to be contained on a given form fits within the **<FORM>** and **</FORM>** tags. (The example below is broken into multiple lines for readability here. This will work in practice, but you have the option of formatting the HTML as a single line or multiple.)

```
<FORM
METHOD="post"
NAME="formName"
ACTION="/msp/urlForSubmitTemplate.msp"
>

<!-- fields, buttons, images, hypertext go here -->

</FORM>
```

HTML forms transmit results to the web server using one of these methods:

- **GET**- the form's contents (name/value pairs) are transmitted as the query string portion of a URL. **This URL can possibly become visible to users** as the "location address" at the top of their browsers.
- **POST**- the form's contents are URL encoded and transmitted **transparently to the user**. These data are embedded within escape sequences.

	GET	POST
Fields: (name/value pairs)	Visible in the URL as the query string. (See **Query Strings** on page 76 for the syntax.)	Not visible in the URL. Transmitted as an encoded data stream between the browser and server.
Hidden fields	You use these as a way of conveying information from the user's browser back to the server. ***NOTE:*** The user does not see hidden fields on the face on the form. But the user can see the name and value of these fields by viewing the source in the browser.	

Input field

These are the most commonly occurring items in web forms. They are the equivalent of an Xbase **@ SAY GET**.

```
<input type="text" name="fieldname" value="defaultOrFieldValue"
size="displayLength" maxlength="MaximumDataEntryLength">
```

- Unlike Xbase fields, there are no format masks to control the data entry.
- HTML has no concept of data types; fields accept alphanumeric and printable ASCII characters. MSP will transparently convert data. So when a form field returns the value "5" and you need to **REPLACE** it into a numeric field, MSP avoids a data type mismatch.
- You must choose between validation on the client-side (these happen at runtime while the form is still displayed on the user's browser) or server-side (these happen only after the user submits the form and data has been sent back to your server).
- Your web pages will be much more friendly if you use client-side validation as much as possible. (See **Use Javascript For Data Validation** on page 77.)

Password field

These fields are very similar to input fields. But the user input is masked with ***** characters as the user types. The contents of a password field cannot be copied and pasted.

```
<input type="password" name="regpassword">
```

Textarea

These fields are intended for large amounts of text. When you want to collect data into a large character field or a memo field, use this form item.

```
<TEXTAREA name="comments" COLS="40" ROWS="5">
This is default text appearing in the field. Or you can use an MSP code
fragment to get data from the previously accessed database:
<% ? customer->comments %>
</TEXTAREA>
```

Checkbox

These fields are excellent for use with logical fields in Xbase or any place where a true/false on/off data value is needed. The text for checkbox items is not an attribute of the checkbox item. Instead it sits "outside" the checkbox item as part of the web page:

```
<input type="checkbox" name="cb_sendmail" value="Y"> Send me email
```

Radio button

This is a user interface item used when you have a small set of choices and you want the user to specify only one value. The beauty of MSP is that you can dynamically generate the list of radiobuttons on the server side using programming logic or database references. To create a group of radiobuttons, make sure that each radiobutton in the group has the same **name** attribute. The radio button clicked by the user will be the one that returns the value attribute for the group.

```
<input type="radio" name="datatype" value="Clipper">
<input type="radio" name="datatype" value="dBASE">
<input type="radio" name="datatype" value="FoxPro">
<input type="radio" name="datatype" value="Sybase">
<input type="radio" name="datatype" value="Microsoft SQL Server">
<input type="radio" name="datatype" value="mySQL">
<input type="radio" name="datatype" value="Postgres">
<input type="radio" name="datatype" value="Oracle">
```

Select field

This item appears as either a single-row "pulldown" list or a multi-line list.

You determine the presentation using the **SIZE** attribute. (A value of 1 presents a pulldown and any value higher than that generates a list for the number of rows you specify.) If you specify a **SIZE** attribute higher than 1, the browser will automatically generate a vertical scrollbar when needed. (Most browsers automatically set the width of the **SELECT** list to the longest string in the **OPTION** list.

To allow multiple choices, include the **MULTIPLE** attribute.

```
<SELECT name="fruits" MULTIPLE SIZE="3">
  <OPTION>Banana</OPTION>
  <OPTION>Apple</OPTION>
  <OPTION>Orange</OPTION>
</SELECT>
```

Using MSP code fragments you can dynamically generate **SELECT** lists.

NOTE: You may see some web sites that leave out the closing </OPTION> tags. Although today's browsers will usually accept this coding style. Future browsers may be confused.

Hidden fields

Hidden fields are used to transmit data from one web form to another. The field is not visible in the user's browser-- unless the user chooses to view source.

```
<input type="hidden" name="fieldname" value="defaultOrFieldValue">
```

NOTE: Do not be misled by the name! Hidden fields are not secure. If you intend to use hidden fields for valuable or proprietary data, consider encrypting the information at the server side. (Be sure to see **Exposing Data: Are You Giving Away The Keys?** on page 86.)

Button: Submit

Almost every form has a **`submit`** button unless the form has objects with event handlers for **`onChange`** or **`onBlur`**.

```
<input type="submit">
```

The **`value`** attribute for a button is optional. If no **`value`** clause appears, the browser normally displays the word "Submit". But if you supply one, it becomes the text on the face of the button:

```
<input type="submit" value="Register Now">
```

Button: Reset

The **`reset`** button allows the user to clear the values in a form with one click. This button type ordinarily appears on a web form next to the submit button. Just like the **`submit`** button, if you don't supply a **`value`** attribute, you can rely on the browser to supply a default caption ("Reset").

```
<input type="reset" value="Clear this form">
```

Button: User Defined

These buttons are assigned a **`value`** attribute by you. This becomes the text on the button face.

Create an **`onClick`** event handler to be activated when the user clicks this button. One approach is to call the **`onClick`** event handler can directly call a URL with another MSP page as below:

```
<input type="button" value="Order Now"
onClick="location='http://www.your.com/msp/order.msp'">
```

Submitting A Form

There is more than one way to submit a form:

- The classic approach- specify a URL in the **`ACTION`** clause. Then include a **`submit`** button to trigger submission.
- Create a generic button on the form. Create an **`onSubmit`** event handler and specify the URL to be invoked when triggered by a user click on this button.

Regardless of the technique you use, you will need to create a ***submit template*** (also commonly called a ***save template*** because its purpose is almost always to save data to a database.) The purpose of the submit template is to respond to form submission and process the form contents.

Buttons and Javascript Event Handling

A more typical situation is when you want to pass values from the current form back to the web server. One technique is to have the **`onClick`** event handler call a Javascript function. You pass the form object as a parameter to your Javascript function:

```
<input type="button" value="Register"
onClick="javascript:MyFunction( this.form )">
```

Here is a primitive illustration of how all of this would work in the context of a full page:

```
<HTML>
<HEAD>
  <TITLE>Demo of button + Javascript</TITLE>
  <SCRIPT language="javascript">
  <!-- Hide script from old browsers that don't support Javascript
  function myFunction( TheFormObject ) {
     // this is the base URL along with the query string
     var URL = "http://www.yours.com/msp/mypage.asp?email=";

     // now concatenate the URL with the value of the form field
     URL = URL + TheFormObject.emailfield.value;

     // call the next URL by passing user information
     // this information comes from the form fields
     // location = "<URL>" tells browser to
     // load another URL into the current browser window
     location = URL;
  }
  // end hiding of Javascript -->
</HEAD>

<BODY>
  <FORM NAME="something" ACTION="" METHOD="get">
     <!-- form fields and other form items -->
     <input type="text" name="emailfield" value="">
     <input type="button" name="regbutton"
        onClick="javascript:myFunction( this.form )">
  </FORM>
</BODY>
</HTML>
```

NOTE: This manual contains some introductory information about Javascript. (See **Privacy Matters** on page 88.) Additional links on Javascript and web development appear at the end of the manual. (See **Useful Links** on page 88.)

Trying Your First Form

Here is a quick path to learning about the basics of web forms:

- Put at least one field of each type on the form. (Don't forget to put a **`submit`** button on the form.)
- Leave the **`ACTION`** clause empty at first and keep it simple.
- Save it with any file extension that your system recognizes as an HTML document.
- Put it on your web server and try it again.
- Set the form **`method="get"`**.
- Create a simple template to be called when the form is submitted. (This is commonly called a ***submit template*** or more specifically a ***save template*** because you normally want to save data.).
- Put the submit template on your web server. Keep it very simple for your first try. It could simply say **`<H1>Submitted</H1>`**.

  (In subsequent tests, you could add MSP code for displaying and manipulating variables if you like. This would help you to learn how to receive and process variables passed to a page from a submitted form.)
- Create an **`ACTION`** clause in the form. This must specify the full URL for your ***submit template***. This template performs any necessary processing (data transformations, post-submission validation, etc.) Some developers also perform the data updates in this template. Other developers prefer to isolate the database interaction to another template called from the submit template.
- When you submit the form during testing, the browser will display a URL with query string in the location field of the browser. The full URL includes the path for the submit template (which you specified in the **`ACTION`** clause of the form) and query string (including the name/value pairs for all the fields in the form).

You can expand on this exercise by adding database logic. The first phase of

URL (Uniform Resource Locator)

The URL is the addressing mechanism for the web. It consists of these parts:

- protocol
- hostname
- domain name
- virtual or physical directory (optional)
- virtual or physical filename (optional)
- query string (optional)

Relative vs. Absolute URLs

- An **absolute URL** is a full pathname which completely defines how to reach a referenced page or resource. This path has no relation to the site where the link has been placed:
  `http://www.plugsys.com/products/index.html`
- A **relative URL** provides only enough information to reach a referenced page or resource relative to the location of the current page or resource:
  `products.html`
  `./products.html`
  `../index.html`
  `/ordernow.html`

NOTE: Preceding a path with a slash is a far better strategy than specifying the directory navigation (`../up/downagain/sideways/foundit.html`). As long as you specify the toplevel directory in this relative URL, you are allowing yourself to change the relative location of the directory path containing the current link. (On the other hand, your *referenced path* must not be changed or a broken link will result.)

When To Use An Absolute URL

- You want to link to an external web site
- You want to point to a specific server

When To Use A Relative URL

- When you want to keep your options open- this allows you to move your content to other servers with other hostnames and even other domain names.
- Most of the time

Creating Your Development Environment

Regardless of your web server platform, most developers program using development tools on Microsoft desktops. If you are running applications on a Microsoft Windows platform, this chapter offers you customization ideas and tools selection advice. We hope this helps to increase your productivity as you develop Max Server Pages applications.

Making Life Simple

We can't promise to make everything in your life simple. But we can help you to make your web development experience pleasant and cooperative.

- **Getting Control: File Associations** shows you how to launch applications interacting with your MSP templates and how to invoke text editors.
- **Recommendations: Software Tools** explores popular tools used by professional web developers. The good news is that they work very well with MSP.

Getting Control: File Associations

It is possible that another application has already installed MSP files and inserted a file association for operating system access. If so, there are entries in the Windows registry. This section explains how to quickly use the Windows visual interface to customize Windows for editing .msp files.

1. Open ***My Computer*** and display a ***Windows Explorer*** window.
2. Navigate to any of your .msp files. (If you haven't created any yet, either navigate to installed sample .msp files or create an empty text file with the .msp extension.)
3. If you are not displaying the *Details* view, pull down the ***View*** menu and select ***Details***.
4. While highlighting an .msp file, make a note of the description in the *Type* column of this Explorer view.

- If the Type description is "*.MSP file*," this means that no explicity association exists for this file type. Resume with **Creating A New MSP Association** on page 57..
- Otherwise resume with **When An .MSP Association Already Exists** on page 56..

When An .MSP Association Already Exists

1. Pull down the ***View*** menu.
2. Select ***Folder Options***.
3. Click on the ***File Types*** tab.
4. Navigate through the list of file types searching for the the description text you noted earlier.
5. When you have highlighted the correct description, examine the area on the dialog called *File type details*.

Confirm that this selection is related to .msp files.

6. Click the ***Edit*** button.
7. Now focus your attention on the ***Actions*** area of the dialog.
8. Click on the ***New*** button.

Now you will start defining a new action to be associated with this file type. An action invokes an application capable of handling this file type. Typical application types you would want to associate with .msp files include:

- text editors
- web page design tools
- HTML validators

You can add many actions to a file type.

NOTE: Some tools that automatically launch your .msp files will use the first user-specified action. In this case, the first action you insert has a little more significance than the subsequent ones. Other tools will specifically look for an *Open* action. We recommend that you select an application to be the default "file opener." And then make *Open* the first action you add.

Make *Open* the first action you add to the .msp file type:

9. Type **Open** in the ***Action*** field.
10. Click on the Browse button to navigate to the application you want as the default file opener. (Windows will execute this application when you double click on an .MSP file in the Explorer.)

NOTE: The assignment of an application to an action can be changed by you at any time in the future.

11. Click on the ***OK*** button to complete the action assignment.

You can add as many new actions as you like. The name of these actions will appear on the right-click menu when you are working in the Windows Explorer. This can be extremely handy to edit .msp files using a visual web tool, Notepad and your favorite programmer's editor.

Creating A New MSP Association

1. Pull down the ***View*** menu.
2. Select ***Folder Options***.
3. Click on the ***File Types*** tab.

4. Click on the ***New Type*** button.
5. Type in a description to appear in Windows Explorer.
   (We recommend *Max Server Pages template*.)
6. Click on ***New...*** to create a new action.
7. Type **Open** in the ***Action*** field.
8. Click on the Browse button to navigate to the application you want as the default file opener. (Windows will execute this application when you double click on an .MSP file in the Explorer.)

NOTE: The assignment of an application to an action can be changed by you at any time in the future.

9. Click on the ***OK*** button to complete the action assignment.

You can add as many new actions as you like. The name of these actions will appear on the right-click menu when you are working in the Windows Explorer. This can be extremely handy to edit .msp files using a visual web tool, Notepad and your favorite programmer's editor.

Recommendations: Software Tools

If you have experience programming, HTML is not a tough "language" and the web is not a difficult environment to master. But in approaching something new, it is always a good idea to get some resources to help you learn, help aids to keep you informed and tools to speed up your development cycle. Here are some thoughts to help you get started:

Text Editors

A quick search of the web will turn up a long list of text editors for programming and/or web page creation. Fortunately you can test most of these before committing time and money to any of them. Many programmer-oriented editors are adopting features to simplify HTML production. Other editors are specifically for the web world and will not be much help for Xbase programming.

Because the Windows environment has a large audience, there are many choices.

- If you are looking for a commercial editor to assist you with both Xbase programming and HTML, MultiEdit (**http://www.multiedit.com**) is a rather good choice. It is primarily aimed at professional programmers, with specialized behaviors for most popular languages (including Xbase). It also incorporates HTML color coding and help features.
- Although there are many HTML editors, the best single choice is HomeSite (**http://www.allaire.com**). It provides code completion, color coding, superb help and a preview mode.
- If you want to start with a very limited budget, try 1st Page. At the time this manual was published, you could download a free version from **http://www.evrsoft.com/download/**.

Page Design Tools

Many programmers distrust or dislike visually oriented design tools. But because websites are visual things, it's a good idea to have a simple but powerful tool to help you prototype your web pages.

Unfortunately, most of these visual products deserve a bad reputation among programmers because they generate questionable HTML or because they destroy any changes you make to the HTML. The best known tools, Microsoft FrontPage and NetObjects Fusion, are inappropriate for professional web application development for both of these reasons. Microsoft Word offers some HTML capabilities, but it also generates code you would not want to use in many cases.

Dreamweaver (**http://www.dreamweaver.com**) is the best balance of easy visual design and respect for your own work. The product allows you to make changes in a visual design window or within a simple text editor window. When you make a change in one window, the other one immediately reflects the difference. You might also find this approach educational as you learn HTML.

Max Server Pages Language Extensions

This chapter focuses on Max language features specificly for MSP development or that enhance the web development process. For a fuller exploration of the Max language set, see Max Language Reference.

Overview

This section lists language features unique to Max Server Pages:

- **MSPConfigVar()**
- **MSPDecode()**
- **MSPEncode()**
- **MSPFieldCount()**
- **MSPFieldName()**
- **MSPFieldValue()**

Other common language elements are relevant for server side web development. This section summarizes these features to bring them to your attention.

Environment Variables

getenv() retrieves the value of CGI environment variables. See **CGI Environment Variables** on page 82.

String Handling

- **Long string delimiter**- when you are writing code to generate HTML text streams, you will commonly require multiple lines of text. So Max introduces the long text delimiter. This is available in all Max applications:

```
myStory := {{When I was a small child, I remember going to my
grandmother's house to watch her bake. She made the most wonderful
pastries. The kitchen smelled great from the moment I knocked on
the door. I could smell the rich aroma from the window as I walked
home from another baking (and eating) session.}}
```

- **String expansion**- When you develop dynamic web applications, you will want to embed data values into long strings of HTML text. So Max Server Pages simplifies your coding with the combination of {$} substitution metacharacters and the `StrExpand()` function:

```
<% productList := {{
                <TABLE>
                   <TR>
                   <TD>Product Code</TD>
                   <TD>Product Name</TD>
                   <TD>Price</TD>
                   </TR>
                   <TR>
                   <TD>{$}</TD>  && Data item 1 for replacement
                   <TD>{$}</TD>  && Data item 2 for replacement
                   <TD>{$}</TD>  && Data item 3 for replacement
                   </TR>
             }}

//    string to expand  replacement 1 replacement 2 replacement 3
//              v            v           v           v
 ? strexpand( productList,prod->code,prod->name,prod->price )%>
```

NOTE: The above example is one technique tor setting up expansion of Xbase values into a web page. Somne developers like to use this method for long strings. For more typical examples, see **Embedding Xbase Code In MSP** on page 29.

Avoiding Unwanted Line Breaks

Max Server Pages will add HTML
 line break markers when you use the ? operator. These are alternatives when you want to avoid line breaks:

- **`?`** <expression> **`HTML`**
  By using the new **`HTML`** keyword, MSP will not attempt to add a **`<BR>`** marker or translate HTML metacharacters (like **`<`** and **`>`**) into their display equivalents (**`<`** and **`>`**).
- **`??`**<expression>
  MSP takes advantage of classic Xbase behavior. The results are output without a line break.
- **`<%=expression%>`**
  MSP evaluates and returns the expression results without adding a line break.

Operators

= Evaluation Operator

- Within Xbase code brackets, you can ask MSP to evaluate an expression or function and return a value using the **=** operator.

```
<%=date()%>
```

NOTE: Do not include any spaces between the opening code brace **<%** and the **=** operator. (This limitation may be removed in later versions.)

Unary Operators

Max supports these unary operators:

- +=
- -=

These are examples of the unary operators and their classic Xbase equivalents:

+=	**payment += 5**	**payment = payment + 5**

	`St = "PlugSys"`
	`St += space(1)`
	`St += "International"`
	`? St`
	`* returns "PlugSys International"`
		`St ="PlugSys"`
		`St =St + space(1)`
		`St =St + "International"`

+-	`pmt = 20`
	`pmt +- 3`
	`* pmt is now 17`
		`pmt = pmt - 3`

The example above demonstrates a single row in a table. But what happens when you need to SKIP through a database table and keep building a string? Use the **+=** operator to incrementally build the full HTML stream:

```
// Top of HTML table-- all hard-coded
productList : = {{
             <TABLE>
                <TR>
                <TD>Product Code</TD>
                <TD>Product Name</TD>
                <TD>Price</TD>
                </TR>
          }}

// Now skip through the table
```

```
SELECT prod
  DO WHILE not EOF()
     // build one row of HTML table at a time
     // use metacharacter for expansion in the next step
     tRow := {{
                 <TR>
                 <TD>{$}</TD>
                 <TD>{$}</TD>
                 <TD>{$}</TD>
                 </TR>
     }}

     // now replace the metacharacters with data values:
     tRow := StrExpand(tRow, prod->code, prod->name, prod->price)

     // expand the full string
     productList += tableRow
     SKIP
ENDDO

prodList += "</TABLE>"

? prodList
```

For a fuller exploration of the Max language set, see Max Language Reference and Migration Guide.

MSPConfigVar()

returns the content of a variable (entry) defined on the MSP Configuration file.

MSPConfigVar(MSPentry)

Platforms:

☐☐Web

Return Value:

The value of the entry appearing in the MSP configuration file.

Remarks:

You may use this function to

- retrieve the standard configuration file entry values (MSPDir, LogDir, DataDir, etc)
- custom entries you have added to the config file for your own purposes

Example:

```
SamplesDir := MSPConfigVar("MSPDir")+"/samples"
MyDataDir := SamplesDir + "/emailsubscription"
SET DEFAULT TO &MyDataDir
USE subscriptions
? "There are " | recno() | "email subscriptions on our database!"
```

Also see:

MSPFieldCount(), MSPFieldName(), MSPFieldValue()

MSPDecode()

returns a string previously encoded with **MSPEncode()** or Javascript encode().

```
MSPDecode( String )
```

Return Value:

Returns the original string to its state before it was encoded as with MSPEncode() and Javascript encode().

Remarks:

You may use this function to

- restore a previously encoded string to its original state.

For more information, see **MSPEncode()**.

Example:

```
<%
  var := "*This string has spaces. And a question? info@plugsys.com"
  ? var
 %>
 <HR>
<%
 stringX := MSPEncode( var )
 ?? stx
 %>
 <HR>
<% = MSPDecode( stringX ) %>
 <HR>
```

Also see:

MSPConfigVar(), **MSPEncode()**, **MSPFieldCount()**, **MSPFieldCount()**, **MSPFieldName()**, **MSPFieldValue()**

MSPEncode()

creates a string with embedded hexadecimal tokens to preserve non-alphanumeric, non-ASCII characters (similar to Javascript encode() function).

```
MSPEncode( String )
```

Return Value:

Returns a string with embedded hexadecimal tokens. This preserves non-alphanumeric, non-ASCII characters (similar to Javascript encode() function).

Remarks:

When building a query string, you must ensure that the values are free of spaces and non-alphanumeric, non-ASCII characters. Unless you can be sure that a variable meets this requirement, use MSPEncode() to replace noncomplying characters with a hexadecimal equivalent. (Spaces are replaced with %20, for example.)

You may use this function to

- rPreserve spaces.
- Preserve accented characters

Example:

```
<%
  var := "*This string has spaces. And a question? info@plugsys.com"
%>
<%
 stringX := MSPEncode( var )
 ?? stx
 %>
 <HR>
<FORM action=”myTemplate.msp?string=<% = MSPDecode( stringX ) %>”>
...
</FORM>
```

Also see:

MSPConfigVar(), MSPFieldCount(), MSPDecode(), MSPFieldCount(), MSPFieldName(), MSPFieldValue()

MSPFieldCount()

Returns the number of form objects or query string name/value pairs found in the calling MSP page.

MSPFieldCount( )

Platforms:

☐☐Web

Return Value:

An integer representing the number of form objects or query string name/value pairs found in the calling MSP page.

Remarks:

This function is commonly used as part of a routine to handle a submitted HTML form. You can iterate through all passed fields, buttons, etc. Or if a form is passed using the GET method, you can use this to count the number of name/value pairs passed.

Example:

```
NumFields = MSPFieldCount()
for I = 1 to NumFields
    ? MSPFieldName(I) + " = " + MSPFieldValue(I)
next
```

Also see:

MSPConfigVar(), MSPFieldName(), MSPFieldValue()

MSPFieldName()

Returns the name of an HTML form field or name/value pair.

```
MSPFieldName( NumField )
```

Platforms:

☐☐Web

Return Value:

Returns the name of field <NumField> if defined or an empty string if NumField is not valid.

Remarks:

This function is commonly used as part of a routine to handle a submitted HTML form. You can iterate through all passed fields, buttons, etc. Or if a form is passed using the GET method, you can use this to retrieve the name portion of any name/value pair that has been passed.

Example:

```
NumFields = MSPFieldCount()
for I = 1 to NumFields
    ? MSPFieldName(I) + " = " + MSPFieldValue(I)
next
```

Also see:

MSPConfigVar(), MSPFieldCount(), MSPFieldValue()

MSPFieldValue()

Returns the value for the specified HTML field name or the value associated with a name in a query string.

```
MSPFieldValue( FieldName | FieldNum )
```

Platforms:

☐☐Web

Return Value:

Returns a string representing the value associated with the specified HTML field name or the value associated with a name in a query string.

Remarks:

This function is commonly used as part of a routine to handle a submitted HTML form. You can iterate through all passed fields, buttons, etc. Or if a form is passed using the GET method, you can use this to determine the value for each name/value pair or form object passed.

Example:

```
NumFields = MSPFieldCount()
for I = 1 to NumFields
    ? MSPFieldName(I) + " = " + MSPFieldValue(I)
next
```

Also see:

MSPConfigVar(), MSPFieldCount(), MSPFieldName()

Getting Started: Putting It All Together

Learn about MSP development by following these easy steps.Try coding and running these and you will

Hello World

Building the Essential Form

Building A Simple Form Handler

Forms: Adding Validation

Just Add MSP:
Database-Enabling Your Pages

More Web Development Issues

Now that you have explored the essential concepts and features of HTML, this chapter will take you to the next level. The material includes: query strings, Javascript and security issues.

Query Strings

Most web sites have directories of ASCII files sitting on the web server machine that serve as "static" web pages. To simplify, static web pages do not change. (Some pages may have simple Javascript code to display a date and time on the page, the number of site visitors or some other small textual change. But in these cases, the change is activated by static Javascript code that gets sent to the user's browser along with the static HTML.)

Increasingly the web is being populated with dynamically assembled web pages. Many of these pages are dependent on parameters that get passed from the browser to the web server. The best example of this is visible when you use a search engine like AltaVista.

This is the example of the URL that results if you search on AltaVista for PlugSys:

```
http://www.altavista.com/cgi-bin/query?pg=q&sc=on&q=PlugSys
```

A query string is the portion of the URL that follows the **?**:

```
pg=q&sc=on&q=PlugSys
```

The form for each element is a name/value pair in the form: **name=value**. That is, each "variable name" is passed along with its value.

If you need to pass more than one variable, you use an ampersand **&** to append each succeeding name/value pair.

This is an example of a typical Max Server Pages URL that includes a query string with two name/value pairs:

```
http://www.a.com/msp/script.msp?field1=Smith&field2=John
```

What if you need to pass characters that are not alphanumeric or if you need to include spaces? Use the hex value for the ASCII character number with a **%** for a prefix. The space character is **%20** and the quote is **%22**. Here is one example:

```
sc=on&q=%22PlugSys%20International%22
```


Javascript Overview

Obviously, this manual cannot teach you everything about an external programming language and the objects it manages. But we offer you this information to introduce you to Javascript.

Javascript is a language with a lot of similarities to C. But it is meant to be simple. So it does not include constructs like pointers. This is not Java or a subset of Java. Although it has the word "Java" in there, the similarities to Java are purely because they are influenced by the C language.

Originally developed at Netscape, the language was originally going to be called LiveScript. Once Netscape united with Sun to include a Java virtual machine with the browser, they decided a new name might provide added marketing momentum.

Some users of older Netscape web servers have used Javascript as a server-side scripting language. (If you are reading this manual, you probably have concluded that Xbase is a far more attractive server-side application development language.)

Most of us are familiar with Javascript as a client-side language. It comes for free with the two major browsers (Netscape Navigator/Communicator included Javascript beginning in version 2.0 and Microsoft Internet Explorer introduced its Jscript subset clone in 3.0 and released a reliable implementation beginning with MSIE 4.0.)

The Role Of Javascript

If you have experimented with the web as a user or as a developer, you probably have already encountered Javascript (or Microsoft's implementation known as Jscript). You may have wondered why Javascript exists and what it is used for.

Javascript was conceived as a way to provide **client-side** scripting. You embed Javascript in HTML. The script is sent to the browser along with the rest of the HTML page. And when appropriate events occur on the user's browser, a script is executed locally **on that user's browser** (without requiring additional requests to the web server).

Use Javascript For Data Validation

A classic use for Javascript is to enforce data validation on the user's browser when they are filling in a form. This enables data entry to occur without having to make multiple trips to the web server. Naturally this offloads web traffic (whether you are on a local area network or on the Internet).

In the case of an HTML form, Javascript is well suited to field-level or form-level data validation.A user fills in the form and then presses a "submit" button:

- Javascript code written by you is embedded in the web page along with the form. You have bound an **`onClick`** event on the Submit button to your script. The user clicks the Submit button. Javascript invokes a popup window advising the user of a data rule violation. No data ever goes across the wire to the web server, so Max Server Pages is not yet invoked.
- The user corrects the data entry error and again clicks the Submit button. This time the Javascript code encounters no problems, so it passes a request to the web server to execute a Max Server Pages application.
- The web server receives the form's data and passes the information to Max Server Pages. Max Server Pages responds to this input.

Javascript Characteristics And Conventions

- Javascript is case-sensitive- there is a difference between a variable named myVar and myvar (this applies to function names too).
- operators are mostly borrowed from C
- By convention, built-in and user defined function names with more than one syllable are on capitalized on the first syllable: **`myNewCalculatingFunction()`**
- Every command line ends with a semi-colon:

```
var myName = "Bob";
var myAge = 33;
```

- The operators are derived from the C language

NOTE: This manual includes several web links with more information. (See **Useful Links** on page 88.)

Sample Javascript Code

The following web page was found in the library at **http://www.javascripts.com**.

- The core of this page is the Javascript function **`VerifyEmailAddress()`**.
- The function is called at the form level (look for the **`onSubmit`** event).

It is presented here for you to further explore. You can test it by saving it as an html file and loading it into your web browser.

```
<!-- This script has been in the
http://www.javascripts.com Javascript Public Library! -->
<!-- This material may have been in a public depository, but certain
author copyright restrictions may apply. -->
<html>
<head>
<title>Email Verification: A Javascript Demo</title>
</head>

<body>
<script LANGUAGE="JavaScript">
<!--
function VerifyEmailAddress(EmailForm)
{
  var Reason  = "Email Address entered incorrectly.  Please Ok to re-
enter or Cancel to continue.\n\nReason:"
  var Success = "Email Address entered Correctly!\n\nScott G.
Walch\nsgw@CyberNext.com"
  var checkStr = EmailForm.email.value;
  var ix = (checkStr.length - 4)
  var RC = true;
  var x = AtSignValid = DoublePeriod = PeriodValid = SpaceValid =
ExtValid = RL = 0;
```

```
  for (i = 0;  i < checkStr.length;  i++)
  {
    if (checkStr.charAt(i) == '@')
      AtSignValid++;
    else if (checkStr.charAt(i) == '.')
    {
      if (x == (i-1))
        DoublePeriod++;
      else
      {
        x = i;
        PeriodValid++;
      }
    }
    else if (checkStr.charAt(i) == ' ')
      SpaceValid ++;
  }
  if (checkStr.indexOf(".com", ix) > -1)
    ExtValid++;
  else if (checkStr.indexOf(".edu", ix) > -1)
    ExtValid++;
  else if (checkStr.indexOf(".net", ix) > -1)
    ExtValid++;
  else if (checkStr.indexOf(".org", ix) > -1)
    ExtValid++;
  else if (checkStr.indexOf(".gov", ix) > -1)
    ExtValid++;
  RL = Reason.length;
  if (AtSignValid != 1)
    Reason += "\nOnly one '@' allowed, " + AtSignValid + " found.";
  if (PeriodValid == 0)
    Reason += "\nAddress must contain at least one period.";
  if (SpaceValid > 0)
    Reason += "\nNo Spaces allowed. Address contains " + SpaceValid + "
space";
  if (SpaceValid > 1)
    Reason += "s.";
  if (DoublePeriod > 0)
    Reason += "\nAddress contains multiple periods in a row.";
  if (ExtValid == 0)
    Reason += "\nInvalid Domain Suffix entered.\n(Valid: .com, .edu,
.net, .org, .gov)";
  if (checkStr.length > 120)
    Reason += "\nPlease limit the Email Address to 120 characters.";
```

```
  if (RL != Reason.length)
  {
    if (confirm(Reason))
    {
      EmailForm.email.focus();
      RC = false;
    }
    else
      RC = true;
  }
  else
    RC = true;
  return(RC);
}
//-->
</script>
<h1><font face="Arial" size="5" color="#000080"><strong>Email Entry
Validation</strong></font></h1>
<p><strong><em><font face="Arial" size="3">by Scott G. Walch</font></
em></strong></p>

<p><font face="Arial" size="3">The purpose of this script is to validate
an email
address entered by a web user.  Many times a user will enter an
address too quickly
or forget to add the &quot;.com&quot; to the end.  This script is
written to check
the syntax of a domestic email address (.com, .edu, .net, .org, .gov) and
check for the
existence of at least one &quot;.&quot;,  <u>only</u> one
&quot;@&quot; and ensures
there is no embedded spaces.
<br>See this script in production at: <a href="http://NetBank.com/
email.html">net.b@nk</a>
</font></p>

<form method="GET" action="http://www.javascripts.com/repository/
emailjava.html" onsubmit="return VerifyEmailAddress(this)"
name="EmailForm">
  <table border="0" cellpadding="0" cellspacing="0" width="50%">
    <tr>
      <td width="50%"><font face="Arial" size="3" color="#000080">
        <strong>Email Address</strong></font></td>
```

```
      <td width="50%"></td>
    </tr>
    <tr>
      <td width="50%"><input type="text" name="email" size="20"
maxlength="120"></td>
      <td width="50%">
        <input type="submit" value="Email" name="pbEmail">
        <input type="reset" value="Reset" name="pbRefresh">
      </td>
    </tr>
  </table>
</form>
</body>
</html>
```

CGI Environment Variables

All modern web servers generate a standard set of environment variables which are usable by your applications.

These are the standard CGI environment variables (Variables marked with a * are the most commonly used and the most reliable):

Environment Variable	Returns/Description	Comments
GATEWAY_INTERFACE	The CGI version number used by the web server.	
SERVER_NAME*	The current server's hostname (if it is registered in the global domain name service) or IP address (if a hostname and/or domain name is not in the DNS).	Use this to ensure that all URL's on your dynamic pages are on the current server. This is handy when you test on one server and deploy the application on another.
SERVER_SOFTWARE	The name and version of the web server software.	

SERVER_PROTOCOL	The protocol and version used to make the client request. (In most cases, this will be HTTP/0.9, HTTP/1.0 or HTTP/1.1.)	HTTP level supported by the web server.
SERVER_PORT*	The port number at which the server answered the current request.	Use in similar circumstances as **SERVER_NAME**.
REQUEST_METHOD	The method used by the web server to respond to the current request.	The seven methods are: GET, POST, HEAD, PUT, DELETE, LINK and UNLINK. GET is used for most static pages. GET and POST are the the methods used for submitting HTML forms.
PATH_INFO*	The path of the current request to the MSP page.	This consists of the part of the URL following the domain name. Can be used to call other scripts and MSP pages located on the same path as the current MSP template.
PATH_TRANSLATED*	The full physical path of the MSP page as reported by the operating system.	
SCRIPT_NAME*	The full virtual path of the current MSP page being run.	The path to the currently running MSP page. Consists of the part of the URL following the domain name.
DOCUMENT_ROOT	The web server's root directory for serving content.	

QUERY_STRING*	The data passed to the current MSP page. (This follows the **?** in the current page's URL.)	Vital for MSP templates with HTML forms. • If your FORM uses the GET method, the form field values are passed as the query string to the next page. • Pass values in the query string to provide logic for an MSP script.
REMOTE_HOST	The hostname of the user machine requesting the page. (This is not a very reliable source of information about site visitors.)	This might be more useful in an intranet environment, due to tighter control over network and software configuration..
REMOTE_ADDR	The IP address of the user machine requesting the page. (This is not a very reliable source of information about site visitors.)	This might be more useful in an intranet environment, due to tighter control over network and software configuration.
AUTH_TYPE	The authentication method used in validating the user request for this page.	
REMOTE_USER	The authenticated user name.	Available when you require web server-based authentication of the user.
REMOTE_IDENT	The user who initiated the request. (This is not a very reliable source of information about site visitors and does not work in most circumstances.)	

CONTENT_TYPE	The MIME type of the data requested.	
CONTENT_LENGTH	The length (in bytes) of the data passed to the MSP page.	
HTTP_FROM	The email address of the user. (This rarely returns a result because most browsers have disabled the information for privacy reasons.)	
HTTP_ACCEPT	MIME types recognized by the user's browser (returned as a list).	
HTTP_USER_AGENT*	The user's browser.	Useful for writing browser-specific code or keeping a log of browsers used by site visitors.
HTTP_REFERER*	The URL of the page that the user visited when clicking to the current MSP page.	Returns a null value if the current page is the user's first entry. This happens if accessed from a bookmark or when typed directly into the location field on the browser.

Retrieving CGI Environment Variables In MSP Pages

Use **`getenv( enVarName )`** to retrieve the value of a given environment variable. For example, to obtain the user's browser information:

```
You are using the following browser: <%=GETENV("HTTP_USER_AGENT") %>
```

Exposing Data: Are You Giving Away The Keys?

Web applications give you richer opportunities than anything that came before. You can link content from all over the office or all over the world. Graphics can assist text, and links can be placed all over.

But some development assumptions you lived with in the past may not be safe ones today! Because browsers render and respond to HTML text streams, users can potentially read, understand and intercept that data stream. Don't design your application to lock the front door but leave the windows open!

Think carefully about what data items you are willing to expose in your applications. Remember that users have brains. When someone sees one of your URL's, they can attempt to send other, similar URL's to your webserver. What will happen when someone submits other codes or values? Will your application crash? Will it willingly hand out confidential information (or money)? It doesn't matter whether someone is merely curious or trying to be destructive. Your job is to protect your data and applications!

Simple Precautions

- Do not expose primary keys unless the data is unimportant. Instead develop pseudokeys or other indirect identifiers. Try to use approaches with time-sensitive expiring id codes, user-specific sessions, etc. Map the client-submitted code to a server-side code. Let the sensitive processing happen on the server side. Keep your database schema and application logic your own secrets.
- Always remember that "hidden" fields are not invisible fields. Any browser user can display the source code and discover your "secret."
- Assume nothing and build sensible security into your applications. Remember that users can save HTML pages (including HTML forms) to their own systems. Once they have your HTML, they can alter values in hidden fields. That gives a clever or malicious person the ability to direct unexpected requests to your web server.

Internet vs. Intranet

There is only a small technological difference between applications written for the Internet or an Intranet. The distinction is more related to the nature of the environment you are writing for:

- Intranet applications are designed to be used by staff within a given organization. These typically run on a local area network or wide area network. These applications may or may not use password and security protections, depending on policy within that organization.
- Internet applications are designed to be accessed by people (or other systems) anywhere Internet access is available. These applications typically have a higher degree of visual design. Most applications (like a product catalog) are available to the general public and don't require user authentication of any kind. A growing portion of Internet applications are being designed for technical support, customer service and e-commerce. Such applications require authentication. Transactions involving cash transfers or sensitive information may also require the use of the Secure Sockets Layer protocol to encrypt data streams.
- Extranet applications are generally hosted in some public internet location. The difference is that the extranet is offered to customers, partners and suppliers of the organization sponsoring the extranet. Naturally an extranet site will contain confidential information. So this requires user logins and security precautions.

Security and SSL

If you are collecting confidential information from users:

- credit card information
- governmental data
- customer account information

you may want to encrypt the data passing back and forth using SSL.

What Is SSL?

Secure Sockets Layer (SSL) is a protocol for conveying private documents over the Internet. SSL uses a private key to encrypt data transferred over an SSL connection. Most e-commerce Web sites and Internet financial institutions use SSL to collect confidential user information, such as credit card numbers. Ordinarily your web browser will alert you when it is switching from unencrypted content to SSL mode. You will also see the URL prefixed by the https:// protocol identifier.

Links On SSL And Web Security

More information about security issues may be found on the Internet. These are some relevant links to start with:

- **http://www.verisign.com**
- **http://home.netscape.com/security/techbriefs/ssl.html**
- **http://developer.netscape.com/docs/manuals/security/sslin/contents.htm**
- **http://webopedia.internet.com/TERM/S/SSL.html**

Privacy Matters

Many users are becoming concerned with how web site operators use the information provided by the public in the course of purchasing merchandise, subscribing to mailing lists and entering contests. Some countries are already establishing privacy guidelines. If you are responsible for an Internet site which will collect information from the public, we recommend that you research the relevant privacy issues. One good place to start is **http://www.truste.org**.

Useful Links

We offer places to go on the web where you can obtain further information about web development. Because the web is a dynamically changing world, we cannot possibly keep this manual current or list everything available. For that reason, we recommend that you visit web search sites and adopt your own search strategies. (See **Search Strategies** on page 89.)

We hope Max Server Pages will help you contribute to its richness.

HTML Links

- **http://www.ncsa.uiuc.edu/General/Internet/WWW/HTMLPrimerAll.html**
- **http://htmlgoodies.earthweb.com**
- **http://www.hwg.org/resources**
- **http://www.nashville.net/~carl/htmlguide/index.html**
- **http://www.htmlguru.com**
- **http://www.blooberry.com/indexdot/html/index.html**
- **http://members.aol.com/teachemath/bclass.htm**

Javascript Links

- http://www.jsworld.com
- http://www.javascripts.com
- http://javascriptsearch.com
- http://javascript.internet.com
- http://www.webteacher.com/javascript
- http://developer.netscape.com/docs/manuals/communicator/jsguide4/index.htm
- http://developer.netscape.com/docs/manuals/communicator/jsref/index.htm
- http://home.netscape.com/eng/mozilla/3.0/handbook/javascript/index.html
- http://developer.netscape.com/software/jsdebug.html
- http://msdn.microsoft.com/scripting

XML Links

- http://www.w3.org/XML
- http://html.about.com/compute/html/msubxml.htm

Other Web Developer Links

- http://wsabstract.com
- http://www.webmonkey.com
- http://web.oreilly.com/
- http://hotwired.lycos.com/webmonkey/reference/html_cheatsheet/
- http://webopedia.internet.com/

Search Strategies

- http://www.altavista.com (includes excellent language translation features)
- http://www.northernlight.com
- http://www.google.com
- http://www.ask.com
- http://hotbot.lycos.com
- We strongly recommend a Windows desktop tool called *Copernic*. It searches multiple search web sites and then prioritizes results. It has proven a reliable assistant as we perform research. You may find more information at http://www.copernic.com.

Migrating Apps: Getting From Xbase To MSP

The best way to learn a new development environment is to compare it with technology you know well. So this chapter helps you think about porting your existing Xbase applications to Max Server Pages.

Even if you are planning totally new coding with MSP, it pays to read this chapter. It shows you how to get from a world you already know into an exciting new place.

Overview

The goal of this chapter is to help you to

- Develop effective strategies to port existing Xbase source code to the web environment.
- Leverage your Xbase knowledge as you plan your first Max Server Pages development project.

Assumptions

We will assume that you are porting an entire application from a classic character-based Xbase application to web-based system running on a private intranet (a web system running on your local area network). Of course Max Server Pages can be used equally successfully on the Internet. And so the techniques you learn here can also be applied to developing Internet applications. However most Xbase programs were previously written for use as internal organizational systems. These kinds of applications are best suited for the intranet environment.

NOTE: To ensure consistency, the code examples in this chapter will use Clipper conventions. Xbase has a rich core of common commands and functions. If you are more experienced in other flavors of Xbase, we recommend that you familiarize yourself with the Max Language Reference and the Max Developer's Guide.

NOTE: We assume that you already are familiar with HTML fundamentals, Max language extensions to Xbase as well as MSP-specific commands and functions. An introduction to these concepts can be found in **MSP Development Fundamentals** beginning on page 23. The more you know about the web environment, the more sophisticated your applications will be.

Architecture: Designing A Web Application

Web applications rely on a client/server architecture. Web clients (browsers) send requests to a web server. Upon assessing the incoming request, the web server will invoke Max Server Pages while passing to it information sufficient to execute your code. If your code sends properly formatted text to the standard output, your web server will transmit that back to the web browser as a web page. This request/response dialog can continue as long as the user continues to send URL requests to the server.

Here are some of the benefits you gain from this client/server model:

- You can place most of the intelligence on the back end. Whenever possible, you ensure data integrity at the server. This saves the simple tasks (like data validation) for the the web browser. If you build your logic into reusable modules, all applications accessing the database will consistently process incoming data.
- It simplifies the deployment of bug fixes, updates and upgrades of your applications. Whether users are in the next office or on another continent, you simply upload the code to the server. Subsequently users load their web browsers and log in and gain access to your latest release.
- You can do your development work remotely. Use FTP to upload and download site content, code, images, data. Use Telnet to access the remote server. (See **FTP and Telnet For Win32** on page 38.)

Isolate Your User Interface Code

The most obvious difference between web systems and classic Xbase applications is in the nature of the user interface. You need to start thinking about these user interface elements in an existing Xbase application:

- Forms- data entry screens
- Navigation- menus, help system

Forms

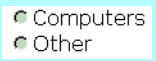
This is where most of your migration effort will be spent. Both the web and Xbase provide entry fields. Like most other HTML constructs, the form has opening and closing tags. Here is a simple example of form syntax:

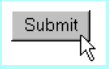
```
<FORM action="/msp/myprog.max?register>
Email: <INPUT TYPE="text" name="email">
<INPUT TYPE="submit">
</FORM>
```

NOTE: A fuller exploration of how forms work can be found in **Web Forms** on page 44..

User Interface Objects

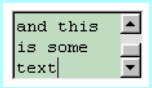
HTML supports the following user interface objects within forms:

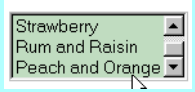
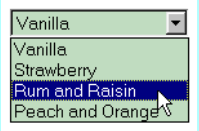
User Interface Object	Description	Sample Code
INPUT	Equivalent to GET fields.	`<INPUT TYPE="TEXT" NAME="email">`
PASSWORD 	A variation on a text entry field. This field shields the user entry by displaying * characters.	`<INPUT TYPE="PASSWORD" NAME="login">`
CHECKBOX 	Excellent for .T./.F. Can also be used when you want a field to take multiple values known in advance.	`<INPUT TYPE="CHECKBOX" NAME="employ" VALUE="Y">Employed`
RADIO 	Use this when you want the user to select one value from a known list of choices.	`<INPUT TYPE="RADIO" NAME="i_field" VALUE="comp">Computers` `<INPUT TYPE="RADIO" NAME="i_field" VALUE="other">Other`
BUTTON 	User clicks on button to execute a script or display another web page.	`<INPUT TYPE="BUTTON" NAME="delete" VALUE="Delete Record" onClick="JavascripFunc()">`

SUBMIT 	A specific type of button. When clicked, the FORM executes its ACTION clause.	`<FORM action="JavascripFunc()">`
		`<INPUT TYPE="SUBMIT">`
		`</FORM>`

RESET 	A specific type of button to clear all field contents on the form.	`<FORM action="JavascripFunc()">`
		`<INPUT TYPE="RESET" VALUE="Clear Form">`
		`</FORM>`

IMAGE	Similar to a push button. Instead of displaying a grey button, the form will show a graphic for the user to click on.	`<input type="image" name="logo" src="http://www.plugsys.com/images/pluglogo.gif" onClick="JavascriptFunc()">`

User Interface Object	Description	Sample Code
HIDDEN	This is a data field that is not visible to the user. You use this as a way of transmitting status flags. ***NOTE:*** Although this field is not visible to the end user on a form, people can still view the source code using their web browsers.	`<INPUT TYPE="hidden" NAME="formtype" VALUE="registration">`
TEXTAREA 	Intended for larger, variable length text. Ideal for use with MEMO field data.	`<TEXTAREA name="skills" ROWS="5" COLS="40"></TEXTAREA>`

SELECT list
 
 
	Used to present a list of choices. By default, accepts a single choice. This object may also be configured for multiple choices. May occupy one row (as a pulldown) or multiple rows by changing the **SIZE** clause.	`<SELECT NAME="flavor">`
		`<OPTION>Vanilla`
		`<OPTION>Strawberry`
		`<OPTION>Rum and Raisin`
		`<OPTION>Peach and Orange`
		`</SELECT>`

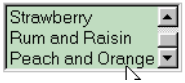
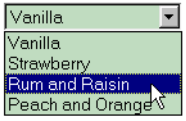
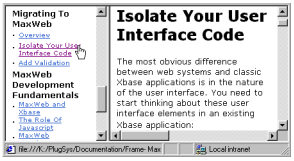
User Interface Object	Description	Sample Code
Hypertext link 	A reference to another web location. This link is clickable by the user. When clicked, the browser loads the URL specified by **A HREF**. Text or graphics can be made clickable by placing either between the **A HREF** and **A** tags.	`<A HREF="http://www.plugsys.com">Click here</A> now`

NOTE: This list is intended to introduce HTML user interface objects as an important part of exploring your migration path. We strongly recommend that you obtain reference materials to get a more detailed sense of the web environment. Some additional HTML information and discussions of web forms appears elsewhere in this manual. (See **HTML Fundamentals** beginning on page 41.)

NOTE: If your code uses "format" files for screens, you will have an easier time migrating your code because FMT files have already isolated your "screen forms."

Navigation

While Xbase provides the MENU command, there is no direct equivalent in HTML. These are common devices for navigation. They may be used individually or in combination:

Navigation Device	Description	Illustration
List of links	A list of links is the most conventional way to construct a "menu." This is a great opportunity to put your "menu system" metadata in a data table and use Xbase code to generate the HTML. (See **Generating A Link List Menu** on page 102.)	
Buttons	Create a group of buttons. The user presses a button and the next page is loaded or another routine is executed. (See **Push Button Menu** on page 102.)	
SELECT list	This menuing style most commonly appears as a drop down. Its most common role is to select a major section of a web site. Drop downs save a lot of space. But they are not a good choice when you want to include a long list of choices. The dropdown menu normally triggers a change using the onChange event. (See **Pulldown Menu** on page 103.) The multiline select list usually is accompanied by a "Go" pushbutton.	 
Frames	Frames are a part of a navigation system. They add more pages to a web site, but they allow a user to navigate through a menu in one region while changing the content area. (See **A Simple Frameset** on page 104.)	

Generating A Link List Menu

This technique places the page location and link text in the data table. The code skips through the table and displaying URL strings.

```
use menutable alias mt
do while !eof()
 url := [<A HREF="http://www.web.com/] + ;
    mt->webpath + ["] + mt->LinkText + "</A><BR>"
 ? url HTML&& displays text without modifying HTML
 SKIP
enddo
```

Push Button Menu

Create an arrangement of push buttons for the user to click on and invoke another MSP template, Javascript function or web page. Many browsers require that buttons appear inside the **<FORM>** tag:

```
<form>
    <input type="button" name="Edit" value="Edit" onClick="location='./
edithandler.msp'">
    <input type="button" name="Add" value="Add" onClick="location='./
addrecords.msp'">
    <input type="button" name="Search" value="Search"
onClick="location='./searchpage.msp'">
    <input type="button" name="List" value="List" onClick="location='./
listrecords.msp'">
</form>
```

Pulldown Menu

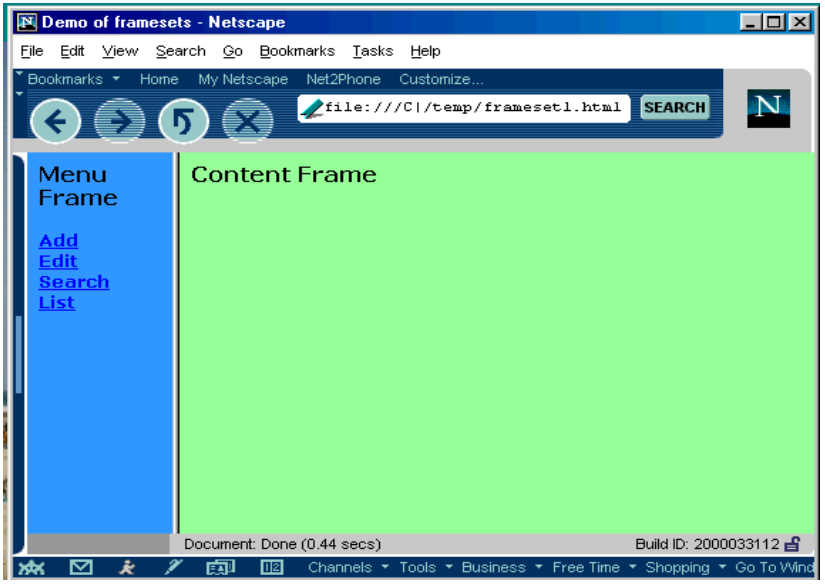
This has become a very popular navigation strategy, because it appears on a single line. You create onChange event handler to activate alternate pages or to call a Javascript function:

```
<html>
<head>
<title>Pulldown Menu With Event Handler</title>
<SCRIPT language="Javascript">
<!--
function goMenu( choice ) {
  // only go to new page if the value is not an empty string
  if (choice !="") {
     location = choice;
  }
}
// -->
</SCRIPT>
</head>
<body bgcolor="#FFFFFF">
<form name="menusystem">
  <!-- pass selected value to Javascript goMenu() func above -->
<form>
   <select name="myMenu" onChange="javascript:goMenu(this.value)">
     <option value="" selected>-- Select One --</option>
     <option value="./editpage.msp">Edit</option>
     <option value="./editpage.msp">Add</option>
     <option value="./searchpage.msp">Search</option>
     <option value="./listpage.msp">List</option>
   </select>
</form>
</form>
</body>
</html>
```

NOTE: Introductory material on Javascript can be found in **Javascript Overview** on page 77..

A Simple Frameset

Frames give you the ability to create multiple "window panes" or subwindows within a single browser window. This allows you to place your navigation in one pane while simultaneously loading other pages in a separately targeted pane.



Components of A Frameset

A frameset consists of:

- a main html frameset descriptor document that stores information about the frames it contains.
- the component frames

Be sure to give each frame a **`name`** attribute. This attribute is used to specify a target. When the user clicks on a navigation element, the **`target`** attribute tells the browser which frame should contain the newly loaded page. (The **`target`** attribute appears in **Navigation Frame** on page 105..)

Main Frameset Descriptor Document

```
<html>
<head>
<title>Demo of framesets</title>
</head>
<frameset cols="119,520" rows="*"
  frameborder="YES" border="5" framespacing="5">
     <frame src="navigation1.html" name="navigation">
     <frame src="content1.html" name="content">
</frameset>
<noframes>
<body bgcolor="#99FFFF">
This appears if the browser cannot display frames.
It is a good idea to fill this area.
</body></noframes>
</html>
```

Navigation Frame

The contents of this frame can be any conventional HTML code using any valid user interface objects described previously.

```
<html>
<head>
<title>Navigation Frame</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>
<body bgcolor="#3399FF">
<b><font face="Verdana, Arial, Helvetica, sans-serif">Menu Frame</
font></b>
<font face="Verdana, Arial, Helvetica, sans-serif" size="-1"><a href="./
addpage.msp" target="content"><b>Add</b></a><br>
  <b><a href="./editpage.msp" target="content">Edit</a><br>
  <a href="./searchpage.msp" target="content">Search</a><br>
  <a href="/listpage.msp" target="content">List</a></b> </font>
</body>
</html>
```

Add Validation

HTML does not provide a native technique for validating data. But client-side Javascript can perform that task. Javascript is a relatively easy language to learn. The more critical aspect of coding in Javascript is understanding the hierarchy of objects managed within a web browser. Because you will be manipulating those objects and their properties.

Client-side code cannot connect to server-side data. So Javascript data validation depends on values available to the browser at runtime. The strategy is to write your applications so that they send all necessary values to the browser.

Web browser clients do not maintain a persistent connection to the web server and the database. (The web server only consults the database when it receives a specific request to do so from a Max Server Pages application.)

NOTE: Both Microsoft and Netscape offer Javascript debuggers. These can be very helpful because they allow you to inspect object hierarchies and the various properties and events. For Microsoft, you can download from the web site and enable the script debugger. (More information can be found at **http://msdn.microsoft.com/scripting/**.) The Netscape debugger is available at **http://developer.netscape.com/software/jsdebug.html**.

Field Level Or Page Level Validation?

Javascript permits you to attach event handlers to individual fields, to the containing web form or both. Here is a simple example that uses both.

- The **onSubmit** event is handled at the form level and is only fired once the user attempts to submit the form.
- The **onBlur** event is handled at the field level when the user attempts to leave the field. It is also fired when the user attempts to submit the form.

```
<html>
<head>
<title>Validation</title>

<script language="Javascript">
<!--
function validateNumber(userValue,max, prompt) {
var userAmount = parseInt( userValue );

  if(userAmount > max)
     { alert( prompt + ": You cannot request more than " +
max.toString());
     return false; }
  else
     return true;
};

function notEmpty(userValue) {
  if(userValue == "")
     { alert( "Empty names are not allowed" );
     return false; }
  else
     return true;
};
// -->
</script>

</head>
```

```
<body bgcolor="#FFFFFF">
<form method="post" action="" onSubmit="javascript:notEmpty(
this.fullname.value )">
  <p><b><font face="Verdana, Arial, Helvetica, sans-serif">Credit limit
requested
    <input type="text" name="creditlimit" value="100.88"
onBlur="validateNumber( this.form.creditlimit.value, 55, 'Credit
Limit')">
    </font></b></p>
  <p><b><font face="Verdana, Arial, Helvetica, sans-serif">Name
    <input type="text" name="fullname">
    </font></b></p>
  <p>
    <input type="submit" name="Submit" value="Submit">
  </p>
</form>
</body>
</html>
```

Using The Max Extension DLL (MEDLL) Interface

What Is MEDLL?

The MEDLL interface allows you to call routines defined in external library files.

MEDLL is the interface supporting communication between Max applications and modules written in other languages like C/C++. Experienced developers and third parties can write code to extend Max functionality using MEDLL.

As MEDLL is completely wrapped in a self-documented C++ class, it is very easy to create DLLs that can perform a handful of tasks: access OS services, handle specific devices, etc.

You cannot directly call routines from ordinary DLLs - you need a way to convert Xbase parameters to C and vice-versa.

The MEDLL templates provided allow you to write routines callable by MSP/Max and provide all the support you need to receive and return values to Max and MSP applications.

Suppose you have a static library called PRINTMNG. This library exposes the C functions print_string() and get_printer_status().

Using the MEDLL templates, you could create a MEDLL called MyPRINTMNG. Then you would define the MEDLL functions PRINT_STRING() and GET_PRINTER_STATUS().

These newly defined functions would call the PRINTMNG library functions print_string() and get_printer_status(). This includes converting Xbase parameters to C types and then returning the C value to Xbase.

After creating your MEDLL, you can load it dynamically using LoadMEDLL(). Once the library is loaded by Max/MSP, you can call any function defined in that library seamlessly, just the way you would call any native Xbase functions or UDF.

The MEDLL Structure

MEDLL exposes an interface allowing Max developers to create functions in external languages that can be called from within their applications.

MEDLL also provides the necessary functions to:

- retrieve parameters passed by the Max application
- accept returning values to the application.

Where to find MEDLL files

Source code and samples are located in the **`../max/medll`** directory of the Max SDK installation.

Please refer to the modules in that directory for illustrations of MEDLL design approaches.

Example:

```
// Load a MEDLL that allows some interesting Windows API calls
dllHandle := loadMEDLL("Max-WinApi.dll")
if dllHandle == 0
   FatalError("Application Module not found: Max-WinApi.dll")
endif

// Call MessageBox() function defined in Max-WinApi.dll
RetVal := MessageBox("Title", MyMessage)
if RetVal == _OK_BUTTON_
. . .
endif
. . .
// Free the MEDLL: no longer used
freeMEDLL(dllHandle)
```

More Web Development Issues

Query Strings
Javascript Overview
CGI Environment Variables
Exposing Data: Are You Giving Away The Keys?
Privacy Matters
Useful Links

Migrating Apps: Getting From Xbase To MSP

Overview
Isolate Your User Interface Code
Add Validation

Using The Max Extension DLL (MEDLL) Interface

What Is MEDLL?