Advanced Web Server Toolkit

Part number/version: 90000684_H
Release date: January 2007
www.digi.com

Contents

Using This Guide

Review this section for basic information about the guide you are using, as well as for general support and contact information.

About this guide

This document describes the NET+OS Web Application Toolkit for the Advanced Web Server. It is based on the *RomPager Advanced Web Server Programming Reference* from Allegro Software Development Corporation.

NET+OS, a network software suite optimized for the NET+ARM processor, is part of the NET+OS integrated product family.

Conventions used in this guide

This table describes the typographic conventions used in this guide:

This convention	Is used for
italic type	Emphasis, new terms, variables, and document titles.
Select **menu** → **menu option** or **options**.	Menu selections. The first word is the menu name; the words that follow are menu selections.
`monospaced type`	File names, path names, and code examples.

Related documentation

For additional documentation, see the Documentation folder in the NET+OS Start menu.

Support

To get help with a question or technical problem with this product, or to make comments and recommendations about our products or documentation, use this contact information:

- Digi Web site: http://www.digi.com
- Digi support: http://www.digiembedded.com
- United States telephone: +1 877 912-3444
- International telephone: +1 952 912-3444

Development Process Overview

CHAPTER 1

This chapter provides an overview of the process of building Web pages for an embedded device running the Advanced Web Server.

Overview

Building Web pages for an embedded device that is running the Advanced Web Server involves these general steps:

1. Design the Web pages for the device.
2. Write the HTML code for these Web pages.
3. Identify the variables to access.
4. Define the HTML links to the variables.
5. Generate C source code from the HTML, using the PBuilder compiler.
6. Flesh out the C source code for variable access.
7. Compile and link the C source code for the target device.

The next sections describe each step in detail.

Step 1. Design the Web pages

The first step in developing Web pages for an embedded device is laying out the actual Web pages. What will the pages look like? Will a page have forms for data entry or will it be just a display page? What user interface elements will be used? Which parts will be static and which will be dynamic? The dynamic components are particularly important because they need to be interfaced to the device.

Page design is beyond the scope of this guide, but many reference books are available on the subject.

Step 2. Write the HTML code

You can use any text editor to create HTML pages because the storage format is a standard text file. Tools such as Adobe PageMill and Microsoft FrontPage speed up the page creation process a great deal because they provide WYSIWYG editing, so you don't need to work directly with HTML tags. You should know some HTML to port the pages efficiently. Most browsers have a `View Source` command, which you can use to look at the HTML of pages from other servers.

Before you integrate your pages, preview them. Place them in a single directory on your disk, and select `Open→ File` to preview the pages.

For pages with forms, check the HTML pages to make sure the form actions (URLs) for each page are specified and unique, and that the form item tags have unique names. Also, make sure that the form method (Get or POST) is the one you want to use:

- **POST method**. Passes the form arguments in the HTTP request body; they are invisible to the user.
- **Get method**. Appends the form arguments to the URL; they are visible.

You might want to use the Get method for repeated (or bookmarked) queries, but usually, you want the forms to be submitted with the POST method.

Step 3. Identify the variables to be accessed

After you design and code your HTML pages, the key engineering task is to identify the device variables used in these pages. Each HTML page to be displayed can contain dynamic data retrieved from the device. Examples of dynamic data display are the status of ports on an Ethernet switch or the level of toner remaining in a laser printer. In addition, the device can be controlled by using HTML forms, which pass data from the browser to the device. Examples of controlling a device are specifying the initial configuration of a router or changing the temperature setting of a building HVAC system.

You need to define access to each variable to be displayed or written. The RomPager engine uses a variable access structure for each variable. The variable access structure contains pointers to the direct memory location of the variable or to an access routine, as well as type information about the variable.

The access routines used to read/write the variables are referred to as `Get/Set` routines throughout this guide because of their resemblance to SNMP `Get/Set` calls. These calls can be the same `Get/Set` routines used by SNMP if the device is SNMP manageable; they also can be new calls written just to support the Web-based management of the device. The variable access structures provide the link between the HTML pages and forms and the `Get/Set` routines.

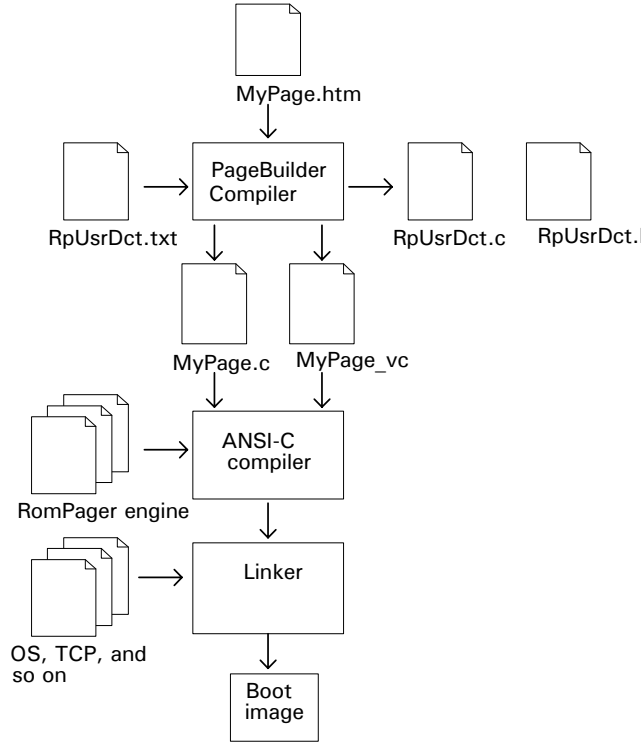
Step 4. Define the HTML links to the variables

Special RomPager HTML comment tags define the variable access structures that can be incorporated into the HTML file that defines a page. The comment tags, which are integrated with the HTML code of a page, pass information to the PBuilder compiler about how the variables will be accessed. The compiler generates the ANSI C source code version of the HTML code.

The combined HTML and comment tags describe the page layout and the access structures to the variables, while keeping the actual code that accesses the variables in separately compiled modules.

Step 5. Generate C source code using the PBuilder compiler

The next step in the process is to pass the HTML pages through the PBuilder compiler to produce C source code. This figure shows the flow of the modules that are developed and passed through compile and linking steps:



The PBuilder compiler uses the HTML input file to build two output files:

- The C source code that represents the compressed internal format of the HTML page with structures for both static text and variable access
- A set of stub routines that can be fleshed out to access the variables

The compiler uses a built-in system dictionary for HTML and an optional user-provided dictionary to compress commonly used character strings and phrases. For more information, see Chapter 4, "Phrase Dictionaries."

If you have more than HTML file, put the files in a batch file and process the batch file (`.pbb`) using PBuilder, instead of processing them individually. Using a batch file ensures that the correct `RpPages.c` file is generated for your application.

Step 6. Flesh out the C source code for variable access

The stub routines produced by the PBuilder compiler need to be fleshed out to access the variables, or they can be discarded if the variable access structures point to existing access routines.

Step 7. Compile and link the C source code

The final step in building the Web pages for the device is compiling the C source code. Because the PBuilder compiler produces standard ANSI C code, you can use any compliant compiler to generate the object code. The object code is then linked into the same ROM image as the engine and the `Get/Set` routines.

Be aware that some PBuilder functions require calling `RpHSGetServerData()`, which returns a handle to the internal data structure maintained on the server. *Do not modify this call until after the Advanced Web Server has been started.*

HTML Comment Tags

CHAPTER 2

This chapter describes how to use comment tags, a special form of standard HTML tags that provide the compiler with instructions for creating the RomPager internal structures

Overview

The PBuilder compiler prepares the internal format of an HTML page by examining the page and storing the information in internal C-format structures. To serve the page, the RomPager engine uses internal structures that consist of:

- *Static* elements for storing the regular HTML
- *Dynamic* elements for storing information about how to create the dynamic HTML at runtime

RomPager *comment tags* are a special form of standard HTML comment tags that provide the compiler with instructions for creating the RomPager internal structures. Because most HTML editors support HTML comment tags, the RomPager internal format information can be stored in the HTML file along with the standard HTML for displaying a page.

HTML comment tag formats

This is the form of an HTML comment tag:

```
<!-- the comment -->
```

The RomPager comment tags are used in this way:

```
<!-- rompagercommenttag keyword1=value1 keyword2=value2 …-->
<P>Some Sample HTML</P>
<!-- RpEnd -->
```

Typically, sample HTML follows this tag, so when you preview a page with an offline browser or HTML page editor, you can see what the page will look like when it is served. The page editor ignores the RomPager comment tags and displays the sample HTML. The `RpEnd` tag tells the compiler where the regular HTML starts up again.

In a case in which there is no sample HTML, you can insert the `RpNoSample` keyword into any comment tag. In this case, the statement:

```
<!-- rompagercommenttag keyword1=value1 keyword2=value2...RpNoSample -->
```

is equivalent to:

```
<!-- rompagercommenttag keyword1=value1 keyword2=value2...--> <!-- RpEnd -->
```

Variable access structures

Most of the dynamic RomPager elements have a variable access structure that contains pointers to access the dynamic information. The pointers are qualified with a type to indicate how the pointer is used to access the variable. The pointers can point to either a memory location or a function routine.

Most variable access structures have methods for:

- Retrieving a value (the `Get` pointer)
- Setting a value (the `Set` pointer)

The comment tag *keywords* specify the values of the variable access structures. You indicate the pointer types with the `RpGetType` and `RpSetType` keywords. The allowed values for these keywords are:

- `Direct`
- `Function`
- `Complex`
- `Null`

If the pointer type is `Direct`, either `RpGetPtr` or `RpSetPtr` specifies the name of the memory location of the variable. If the pointer type is `Function` or `Complex`, either the `RpGetPtr` or `RpSetPtr` keyword specifies the name of the routine to the variable. If the pointer type is `RpGetPtr` or `RpSetPtr`, customer-provided routines are used to access the variable.

If the comment tag describes a form element that is used only to set a value and has no `Get` function, you can specify a keyword value of `Null` for `RpGetType`. If you do not specify the `RpGetType` or `RpSetType` keywords, a value of `Direct` is used. If you do not specify `RpGetPtr` or `RpSetPtr`, and the comment tag contains a `Name` or `RpName` keyword, the pointer name is created from the `Name` or `RpName` keyword.

```
[RpGetType, RpSetType] = [Direct, Function, Complex, Null]
```

When the RomPager engine either serves a page or processes a form, it provides conversion between the HTML ASCII string format and the device internal format. You indicate the type of conversion requested by using the `RpTextType` keyword in the RomPager comment tags.

Here are the allowed values:

`ASCII`	`Signed16`
`ASCIIExtended`	`Signed32`
`ASCIIFixed`	`Unsigned32`
`Hex`	`Unsigned8`
`HexColonForm`	`Unsigned16`
`DotForm`	

If you do not specify the `RpTextType` keyword, `ASCII` is used, as in this example:

```
RpTextType = [ASCII, ASCIIExtended, ASCIIFixed, Hex, HexColonForm, DotForm,
    Signed8, Signed16, Signed32, Unsigned8, Unsigned16, Unsigned32]
```

Internal elements

The internal elements that PBuilder creates have default identifiers. You can use the `RpIdentifier` keyword with most RomPager comment tags to define a specific internal identifier. To save memory, use common elements among more than one page. If an element is to be used more than once, you can provide a specific identifier using the `RpIdentifier` keyword. The `RpUseIdentifier` comment tag is used to refer to the additional uses of the common element. Usually, the internal elements that are defined are both inserted into a page/form element list and individually defined. If the elements being defined are common elements that are included independently, you can set the `Independent` keyword to prevent the element from being included in the page/form element list.

Use the `Name` keyword in any RomPager comment tag to define the HTML name that refers to the internal element. The RomPager comment tags for defining `<INPUT>` tags or form elements all require the `Name` keyword, but you can use this keyword with any RomPager comment tag to set up a name association.

The name takes storage in device memory, so if you have limited resources, keep these recommendations in mind:

- Use name associations only when needed.
- Use short names instead of long names.

If you are using JavaScript in form elements, you must configure both PBuilder and the RomPager engine to store and display the JavaScript with the other internal elements. For PBuilder, you must enable the `UseJavaScript` keyword in the `PbSetUp.txt` file. PBuilder then assumes that all unrecognized keywords are JavaScript and assigns storage in the structure for the JavaScript. The RomPager engine can recreate the JavaScript when the element is displayed.

For an example of using JavaScript in RomPager comment tags, see the `<INPUT TYPE=Button>` example.

Repeat groups

You can define groups of items – such as those that define table rows and columns – once, and repeat them to generate a page. The advantage of using repeat groups is the significant savings in page storage.

The maximum number of nested repeat groups you can use per page is eight.

Static text

Static text, the building block used for the bulk of an HTML page, consists of all text that does not vary as pages are served. PBuilder analyzes static text elements and compresses them using the built-in system phrase dictionary and a user-supplied phrase dictionary.

HTML comment tags

```
<!-- RpDataZeroTerminated -->
<!-- RpDZT -->
<!-- RpEnd -->
```

This table describes the tags:

Tag	Description
`RpDataZeroTerminated` and `RpEnd`	Bracket the HTML static text and define the internal elements that generate the text when the page is served. Not normally needed on a page because PBuilder processes all unrecognized text as if it were bracketed by these tags. You can use them for clarity to define the HTML tool not specified by the other RomPager comment tags.
`RpDZT`	An alias for `RpDataZeroTerminated`.

Function prototypes

None.

Commented HTML

```
<!-- RpDataZeroTerminated --><HTML><HEAD>
<TITLE>HTML Page Title</TITLE>
</HEAD><BODY><!-- RpEnd -->
<!-- RpDZT --><P><H2><CENTER>Visible Page Title</CENTER>
</H2></P><!-- RpEnd -->
```

Internal structures

```
/* Element List Items */
          { eRpItemType_DataZeroTerminated, &PgSample_Item_1 },
          { eRpItemType_DataZeroTerminated, &PgSample_Item_2 },
/* Structures */
static char PgSample_Item_1[] =
          C_oHTML_oHEAD_oTITLE "HTML Page Title" C_xTITLE_xHEAD_oBODY;
static char PgSample_Item_2[] =
          C_oP C_oH2 C_oCENTER "Visible Page Title"
          C_xCENTER C_xH2 C_xP;
```

Sample generated HTML

```
<HTML><HEAD><TITLE>HTML Page Title</TITLE></HEAD><BODY>
<P><H2><CENTER>Visible Page Title</CENTER></H2></P>
```

Dynamic values

You can insert a dynamic value anywhere in a page by pointing to the variable location or a function to provide the value.

HTML comment tags

```
<!-- RpDisplayText RpGetType=gt RpGetPtr=gp RpTextType=t Size=s -->
<!-- RpNamedDisplayText Name=n RpGetType=gt RpGetPtr=gp
          RpTextType=t Size=s -->
```

These tags either define the location of the variable to display or point to a function to retrieve the variable. The value of the variable is dynamically inserted into the HTML page that is being displayed.

This table describes the associated keywords:

Keyword	Description
`Size`	Used for conversions when the `RpTextType` is `ASCIIFixed`, `Hex`, `HecColorNorm`, or Dot `Form` to specify the number of characters that should appear on the page for the field.
`Name`	Used with `RpNamedDisplayText` to specify the HTML name to be passed into complex `Get` routines.

These tags give you the equivalent of "server-side includes;" that is, if you want to have C functions that are called to insert arbitrary display data into the middle of a page, you can use these tags with the default `RpTextType ASCII`.

Function prototypes

```
GetType = Function - TextType = ASCII, ASCIIFixed, Hex, HexColonForm,
                                          DotForm
typedef char * (*rpFetchBytesFuncPtr)(void);
GetType = Complex - TextType = ASCII, ASCIIFixed, Hex, HexColonForm,
DotForm
typedef char * (*rpFetchBytesComplexPtr)(void *theServerDataPtr,
                              char *theNamePtr, Signed16Ptr
theIndexValuesPtr);
GetType = Function - TextType = Unsigned8, Unsigned16, Unsigned32

typedef Unsigned8 (*rpFetchUnsigned8FuncPtr) (void);
```

```
typedef Unsigned16 (*rpFetchUnsigned16FuncPtr)(void);
typedef Unsigned32 (*rpFetchUnsigned32FuncPtr)(void);
GetType = Complex - TextType = Unsigned8, Unsigned16, Unsigned32


typedef Unsigned8 (*rpFetchUnsigned8ComplexPtr)(void *theServerDataPtr,
                                        char *theNamePtr, Signed16Ptr
theIndexValuesPtr);
typedef Unsigned16 (*rpFetchUnsigned16ComplexPtr)(void *theServerDataPtr,
                                        char *theNamePtr, Signed16Ptr
theIndexValuesPtr);
typedef Unsigned32 (*rpFetchUnsigned32ComplexPtr)(void *theServerDataPtr,
                                        char *theNamePtr, Signed16Ptr
theIndexValuesPtr);
GetType = Function - TextType = Signed8, Signed16, Signed32,

typedef Signed8             (*rpFetchSigned8FuncPtr)(void);
typedef Signed16 (*rpFetchSigned16FuncPtr)(void);
typedef Signed32 (*rpFetchSigned32FuncPtr)(void);
GetType = Complex - TextType = Signed8, Signed16, Signed32

typedef Signed8 (*rpFetchSigned8ComplexPtr)(void *theServerDataPtr,
                                        char *theNamePtr, Signed16Ptr
theIndexValuesPtr);
typedef Signed16 (*rpFetchSigned16ComplexPtr)(void *theServerDataPtr,
                                        char *theNamePtr, Signed16Ptr
theIndexValuesPtr);
typedef Signed32 (*rpFetchSigned32ComplexPtr)(void *theServerDataPtr,
                                        char *theNamePtr, Signed16Ptr
theIndexValuesPtr);
```

Commented HTML

```
<!-- RpDisplayText RpGetType=Function RpGetPtr=SomeFunction -->
The IP Address is: <!-- RpEnd -->
<!-- RpDisplayText RpGetType=Direct RpGetPtr=theIpAddress
RpTextType=DotForm Size=15 -->199.200.100.50<!-- RpEnd -->
<!-- RpNamedDisplayText RpName=ThisName RpGetType=Complex
RpGetPtr=AnotherFunction --> on the Computer Room router.<!-- RpEnd -->
```

Internal structures

```
/* Element List Items */
           { eRpItemType_DisplayText,&PgSample_Item_1 },
           { eRpItemType_DisplayText,&PgSample_Item_2 },
           { eRpItemType_NamedDisplayText,&PgSample_Item_3 },
/* Structures */
rpTextDisplayItem PgSample_Item_1 = {
           SomeFunction,
           eRpVarType_Function
           eRpTextType_ASCII,
           0
};
rpTextDisplayItem PgSample_Item_2 = {
           &theIpAddress,
           eRpVarType_Direct,
           eRpTextType_DotForm,
           15
};
```

Sample generated HTML

```
<P>The IP Address is: 199.200.100.50 on the Computer Room router.</P>
```

HTML <INPUT> elements

HTML forms use `<INPUT>` tags to describe information that can be entered in a form. Use the `RpFormInput` tag to provide PBuilder with the information it needs to prepare the internal structures to support the `<INPUT>` tags.

HTML comment tag

```
<!-- RpFormInput TYPE=t NAME=n VALUE=v RpGetType=gt RpGetPtr=gp
                                    RpSetType=st RpSetPtr=sp RpTextType=tt
MaxLength=m
                                    Size=s RpItemNumber=in Dynamic -->
TYPE = [text, password, hidden, checkbox, radio, file, button, reset,
submit]
```

The RpFormInput tag contains the information for both generating an <INPUT> tag and processing the form value. This table describes the keywords associated with RpFormInput:

Keyword	Description
TYPE	Can be text, password, hidden, check box, radio, file, button, reset, or submit for the value for the <INPUT> tag Type keyword. If this keyword is not specified, text is used.
Size	Used for the <INPUT> tag SIZE keyword and for data type conversions when RpTextType is ASCIIFixed, Hex, HexColonForm, or DotForm to specify the number of characters that should appear on the page for the field.
MaxLength	Used for the <INPUT> tag MAXLENGTH keyword.
VALUE	Used for the <INPUT> tag Value keyword when TYPE is radio, reset, or submit.
RpItemNumber	If TYPE is radio, the RpItemNumber keyword can supply a value to which the radio button group variable is set if this radio button is chosen. If RpItemNumber is not specified, the value for the radio button becomes one greater than the last radio button.
Dynamic	Used if the form element will be used inside a repeat group. For more information, see the section “Repeat Groups.”

<INPUT TYPE=TEXT>

HTML comment tag

```
<!-- RpFormInput TYPE=text NAME=n RpGetType=gt RpGetPtr=gp
                                RpSetType=st RpSetPtr=sp RpTextType=tt
MaxLength=m Size=s -->
```

Use this tag in a form to define text entry fields. The next table describes the associated keywords:

Keyword	Description
NAME	Specifies the HTML name of the check box
SIZE	Determines the display size of the field

Keyword	Description
MAXLENGTH	Determines the maximum number of characters that can be entered in the field
RpTextType	Depending on the value of this keyword, the RomPager engine provides automatic data conversion from the HTML/form string format

The RomPager engine provides automatic data conversion from the HTML/form string format depending on the value of the RpTextType keyword.

The function prototypes used to access and store the values are shown next.

Function prototypes

```
GetType = Function - TextType = ASCII, ASCIIFixed, Hex, HexColonForm,
DotForm
typedef char * (*rpFetchBytesFuncPtr)(void);

GetType = Complex - TextType = ASCII, ASCIIFixed, Hex, HexColonForm,
DotForm

typedef char * (*rpFetchBytesComplexPtr)(void *theServerDataPtr,
char *theNamePtr, Signed16Ptr theIndexValuesPtr);
GetType = Function - TextType = Unsigned8, Unsigned16, Unsigned32

typedef Unsigned8 (*rpFetchUnsigned8FuncPtr)(void);
typedef Unsigned16 (*rpFetchUnsigned16FuncPtr)(void);
GetType = Complex - TextType = Unsigned8, Unsigned16, Unsigned32

typedef Unsigned8 (*rpFetchUnsigned8ComplexPtr)(void *theServerDataPtr,
char *theNamePtr, Signed16Ptr theIndexValuesPtr);
typedef Unsigned16 (*rpFetchUnsigned16ComplexPtr)(void *theServerDataPtr,
char *theNamePtr, Signed16Ptr theIndexValuesPtr);
typedef Unsigned32 (*rpFetchUnsigned32ComplexPtr)(void *theServerDataPtr,
char *theNamePtr, Signed16Ptr theIndexValuesPtr);

GetType = Function - TextType = Signed8, Signed16, Signed32

typedef Signed8 (*rpFetchSigned8FuncPtr)(void);
typedef Signed16 (*rpFetchSigned16FuncPtr)(void);
typedef Signed32 (*rpFetchSigned32FuncPtr)(void);
```

```
GetType = Complex - TextType = Signed8, Signed16, Signed32

typedef Signed8 (*rpFetchSigned8ComplexPtr)(void *theServerDataPtr,
char *theNamePtr, Signed16Ptr theIndexValuesPtr);
typedef Signed16 (*rpFetchSigned16ComplexPtr)(void *theServerDataPtr,
char *theNamePtr, Signed16Ptr theIndexValuesPtr);
typedef Signed32 (*rpFetchSigned32ComplexPtr)(void *theServerDataPtr,
char *theNamePtr, Signed16Ptr theIndexValuesPtr);

SetType = Function - TextType = ASCII, ASCIIFixed, Hex, HexColonForm,
DotForm

typedef void (*rpStoreAsciiTextFuncPtr)(char *theValuePtr);

SetType = Complex - TextType = ASCII, ASCIIFixed, Hex, HexColonForm,
DotForm

typedef void (*rpStoreAsciiTextComplexPtr)(void *theServerDataPtr,
char *theValuePtr, char * theNamePtr,
Signed16Ptr theIndexValuesPtr);

SetType = Function - TextType = Unsigned8, Unsigned16, Unsigned32

typedef void (*rpStoreUnsigned8FuncPtr)(Unsigned8 theValue);
typedef void (*rpStoreUnsigned16FuncPtr)(Unsigned16 theValue);
typedef void (*rpStoreUnsigned32FuncPtr)(Unsigned32 theValue);

SetType = Complex - TextType = Unsigned8, Unsigned16, Unsigned32

typedef void (*rpStoreUnsigned8ComplexPtr)(void *theServerDataPtr,
Unsigned8 theValue, char *theNamePtr,
Signed16Ptr theIndexValuesPtr);
typedef void (*rpStoreUnsigned16ComplexPtr)(void *theServerDataPtr,
Unsigned16 theValue, char *theNamePtr,
Signed16Ptr theIndexValuesPtr);
typedef void (*rpStoreUnsigned32ComplexPtr)(void *theServerDataPtr,
Unsigned32 theValue, char *theNamePtr,
Signed16Ptr theIndexValuesPtr);

SetType = Function - TextType = Signed8, Signed16, Signed32
```

```
typedef void (*rpStoreSigned8FuncPtr)(Signed8 theValue);
typedef void (*rpStoreSigned16FuncPtr)(Signed16 theValue);
typedef void (*rpStoreSigned32FuncPtr)(Signed32 theValue);

SetType = Complex - TextType = Signed8, Signed16, Signed32, Signed64

typedef void (*rpStoreSigned8ComplexPtr)(void *theServerDataPtr,
Signed8 theValue, char *theNamePtr,
Signed16Ptr theIndexValuesPtr);
typedef void (*rpStoreSigned16ComplexPtr)(void *theServerDataPtr,
Signed16 theValue, char *theNamePtr,
Signed16Ptr theIndexValuesPtr);

typedef void (*rpStoreSigned32ComplexPtr)(void *theServerDataPtr,
Signed32 theValue, char *theNamePtr,
Signed16Ptr theIndexValuesPtr);
```

Commented HTML

```
<!-- RpFormInput TYPE="TEXT" NAME="TextValue" SIZE="20" MAXLENGTH="20" -->
<INPUT TYPE="TEXT" NAME="TextValue" VALUE="Initial Text"
           SIZE="20" MAXLENGTH="20" >
<!-- RpEnd -->
<!-- RpFormInput TYPE="TEXT" NAME="DefaultGateway" SIZE="15"
MAXLENGTH="15"
           RpGetType=Complex RpGetPtr=GetDefaultGateway
           RpSetType=Complex RpSetPtr=SetDefaultGateway
RpTextType="DotForm" -->
<INPUT TYPE="TEXT" NAME="DefaultGateway" VALUE="0.0.0.0"
           SIZE="15" MAXLENGTH="15" >
<!-- RpEnd -->
```

Internal structures

```
/* Element List Items */
           { eRpItemType_FormAsciiText, &PgSample_Item_1 },
           { eRpItemType_FormAsciiText, &PgSample_Item_2 },

/* Structures */
```

```
static rpTextFormItem PgSample_Item_1 = {
          "TextValue",
          &TextValue,
          &TextValue,
          eRpVarType_Direct,
          eRpVarType_Direct,
          eRpTextType_ASCII,
          20,
          20
};

static rpTextFormItem PgSample_Item_2 = {
          "DefaultGateway",
          GetDefaultGateway,
          SetDefaultGateway,
          eRpVarType_Complex,
          eRpVarType_Complex,
          eRpTextType_DotForm,
          15,
          15
};

/* Variables and Functions */

char      TextValue[21] = "Initial Text";
char      theGatewayAddress[4] = { 0, 0, 0, 0 };

char *GetDefaultGateway(void *theServerDataPtr, char *theNamePtr,
Signed16Ptr theIndexValuesPtr) {

          return theGatewayAddress;
}

void SetDefaultGateway(void *theServerDataPtr, char *theValuePtr,
char *theNamePtr, Signed16Ptr theIndexValuesPtr) {

memcpy(theGatewayAddress, theValuePtr, 4);
}
```

Sample generated HTML

```
<INPUT TYPE="TEXT" NAME="TextValue" SIZE="20" MAXLENGTH="20"
          VALUE="Initial Text">
<INPUT TYPE="TEXT" NAME="DefaultGateway" SIZE="15" MAXLENGTH="15"
          VALUE="0.0.0.0">
```

<INPUT TYPE=PASSWORD>
<INPUT TYPE=HIDDEN>

HTML comment tag

```
<!-- RpFormInput TYPE=password NAME=n RpGetType=gt RpGetPtr=gp
          RpSetType=st RpSetPtr=sp RpTextType=tt MaxLength=m Size=s -->

<!-- RpFormInput TYPE=hidden NAME=n RpGetType=gt RpGetPtr=gp
          RpSetType=st RpSetPtr=sp RpTextType=tt MaxLength=m Size=s -->
```

These tags are used in a form to define password and hidden fields. Password fields behave the same way as text entry fields, except that the browser does not display the value that is keyed in. The browser does not display hidden fields, but these fields are still submitted with other fields in a form. Applications can use them for state passing.

The keywords for these tags work the same way as the `TYPE=text` tags do.

Function prototypes

These comment tags use the same functions for accessing variables as the `TYPE=text` tags.

Commented HTML

```
<!-- RpDataZeroTerminated --><P>Password Field:  <!-- RpEnd -->
<!-- RpFormInput TYPE=PASSWORD NAME=Password SIZE=25 MAXLENGTH=25
RpTextType=ASCII
          RpGetType=Direct RpGetPtr=gPassword
          RpSetType=Direct RpSetPtr=gPassword -->
<INPUT TYPE="PASSWORD" NAME="Password" VALUE="Initial Password Text"
          SIZE="25" MAXLENGTH="25" >
```

```
<!-- RpEnd -->
<!-- RpDataZeroTerminated --><BR>Hidden Field:<!-- RpEnd -->
<!-- RpFormInput TYPE=HIDDEN NAME=Hidden RpTextType=ASCII
          RpGetType=Direct RpGetPtr=gHidden
          RpSetType=Direct RpSetPtr=gHidden -->
<INPUT TYPE="HIDDEN" NAME="Hidden" VALUE="Initial Hidden Text">
<!-- RpEnd -->
```

Internal structures

```
/* Element List Items */
          { eRpItemType_DataZeroTerminated,&PgSample_Item_1 },
          { eRpItemType_FormPasswordText,&PgSample_Item_2 },
          { eRpItemType_DataZeroTerminated,&PgSample_Item_3 },
          { eRpItemType_FormHiddenText,&PgSample_Item_4 },
/* Structures */
static char PgSample_Item_1[] =
          C_oP "Password Field:" C_NBSP C_NBSP;
static rpTextFormItem PgSample_Item_2 = {
          "Password",
          &gPassword,
          &gPassword,
          eRpVarType_Direct,
          eRpVarType_Direct,
          eRpTextType_ASCII,
          25,
          25,
};

static char PgSample_Item_3[] =
          C_oBR "Hidden Field: ";

static rpTextFormItem PgSample_Item_4 = {
          "Hidden",
          &gHidden,
          &gHidden,
          eRpVarType_Direct,
          eRpVarType_Direct,
          eRpTextType_ASCII,
          20,
```

```
            20,
};

/* Variables and Functions */

static char                         gPassword[26] = "Initial Password
Text";
static char                         gHidden[26]= "Initial Hidden Text";
```

Sample generated HTML

```
<P>Password Field:  <INPUT TYPE="PASSWORD" NAME="Password"
SIZE="25" MAXLENGTH="25" VALUE="Initial Password Text"><BR>
Hidden Field:<INPUT TYPE="HIDDEN" NAME="Hidden" VALUE="Initial Hidden
Text">
```

<INPUT TYPE=CHECKBOX>

HTML comment tag

```
<!-- RpFormInput TYPE=checkbox NAME=n RpGetType=gt RpGetPtr=gp
                                RpSetType=st RpSetPtr=sp -->
```

This tag defines check box elements in a form. The HTML name of the check box is specified by the `NAME` keyword. The functions described next can be used to access and store a Boolean value that determines whether the box is checked.

Because HTML forms submit only the check box items that are checked on a form, the RomPager engine turns off all check boxes on a form at the beginning of form processing. As the form is processed, the checked individual check box items are set to on. Both the `reset` and `set` operations use the `Set` function specified by the comment tag.

Function prototypes

```
GetType = Function
typedef Boolean (*rpFetchBooleanFuncPtr)(void);
GetType = Complex
typedef Boolean (*rpFetchBooleanComplexPtr)(void *theServerDataPtr,
char *theNamePtr, Signed16Ptr theIndexValuesPtr);
```

```
SetType = Function

typedef void (*rpStoreBooleanFuncPtr)(Boolean theValue);

SetType = Complex

typedef void (*rpStoreBooleanComplexPtr)(void *theServerDataPtr,
Boolean theValue, char *theNamePtr,
Signed16Ptr theIndexValuesPtr);
```

Commented HTML

```
<!-- RpFormInput TYPE=CHECKBOX NAME=CheckBox
          RpGetType=Function RpGetPtr=GetCheckBox
          RpSetType=Function RpSetPtr=SetCheckBox -->
<INPUT TYPE="CHECKBOX" NAME="CheckBox">
<!-- RpEnd -->
<!-- RpDataZeroTerminated -->A Checkbox<BR><!-- RpEnd -->
```

Internal structures

```
/* Element List Items */
          { eRpItemType_FormCheckbox,        &PgSample_Item_1 },
          { eRpItemType_DataZeroTerminated,  &PgSample_Item_2 },
/* Structures */
static rpCheckboxFormItem PgSample_Item_1 = {
          "CheckBox",
          GetCheckBox,
          SetCheckBox,
          eRpVarType_Function,
          eRpVarType_Function
};

static char PgSample_Item_2[] =
          "A Checkbox" C_oBR;

/* Variables and Functions */
Boolean theCheckBoxValue = True;
Boolean GetCheckBox(void) {
          return theCheckBoxValue;
```

```
}

void SetCheckBox(Boolean theValue) {

          theCheckBoxValue = theValue;
}
```

Sample generated HTML

```
<INPUT TYPE="CHECKBOX" NAME="CheckBox" CHECKED>A Checkbox<BR>
```

<INPUT TYPE=RADIO>

HTML comment tag

```
<!-- RpRadioButtonGroup NAME=n RpGetType=gt RpGetPtr=gp
          RpSetType=st RpSetPtr=sp -->
<!-- RpFormInput TYPE=radio NAME=n VALUE=v RpItemNumber=in -->
```

These tags are used in a form to define radio button groups. Radio buttons specify groups of values, only one of which can be selected at a time.

This table describes the tags:

Tag	Description
`RpRadioButtonGroup`	Contains the information common to all radio buttons of a radio button group, including that for generating the button display values and processing the form value. Must precede the `<!-- RpFormInput TYPE=radio -->` tag for a given radio button group. If `RpRadioButtonGroup` is missing, PBuilder creates a radio group structure with default values.
`RpFormInput`	For a radio button, uses these keywords. ■ `Name`. Specifies the radio group the radio button belongs to. ■ `Value`. Specifies the HTML value ■ `RpItemNumber`. Specifies the item number used for internal processing. If RpItemNumber keywords are not specified, PBuilder supplies default values starting at 0.

This table describes the keywords associated with the `RpRadioButtonGroup` and `RpFormInput` tags:

Keyword	Description
`Name`	Specifies the radio button group.
`Value`	Specifies the HTML value.
`RpItemNumber`	Specifies the item number. If this keyword is missing, PBuilder supplies default values starting at 0.

When a page is displayed, the RomPager engine uses the access functions described next to retrieve a value. The RomPager matches the retrieved value against the item number of each button in the group to determine which button to highlight. When a form is submitted, the RomPager engine matches the HTML value of the submitted button against the buttons in the radio group to determine which item number to pass to the storage functions.

Function prototypes

```
GetType = Function
typedef rpOneOfSeveral (*rpFetchRadioGroupFuncPtr)(void);

GetType = Complex
typedef rpOneOfSeveral (*rpFetchRadioGroupComplexPtr)
(void *theServerDataPtr, char *theNamePtr,
Signed16Ptr theIndexValuesPtr);
SetType = Function

typedef void (*rpStoreRadioGroupFuncPtr)(rpOneOfSeveral theValue);

SetType = Complex

typedef void (*rpStoreRadioGroupComplexPtr)(void *theServerDataPtr,
rpOneOfSeveral theValue, char *theNamePtr,
Signed16Ptr theIndexValuesPtr);
```

Commented HTML

```
<!-- RpRadioButtonGroup NAME=RadioButtonGroup
        RpGetType=Direct RpGetPtr=gRadioButtonGroup RpSetType=Direct
RpSetPtr=gRadioButtonGroup -->
<!-- RpFormInput TYPE=RADIO NAME=RadioButtonGroup VALUE=One RpItemNumber=0
-->
    <INPUT TYPE="RADIO" NAME="RadioButtonGroup" VALUE="One">
<!-- RpEnd -->
<!-- RpDZT -->One Radio Button<!-- RpEnd -->
<!-- RpFormInput TYPE=RADIO NAME=RadioButtonGroup VALUE=Another
RpItemNumber=1 -->
    <INPUT TYPE="RADIO" NAME="RadioButtonGroup" VALUE="Another" CHECKED>
<!-- RpEnd -->
<!-- RpDZT -->Another Radio Button<BR><!-- RpEnd -->
```

Internal structures

```
/* Element List Items */
           { eRpItemType_FormRadioButton, &PgSample_Item_1 },
           { eRpItemType_DataZeroTerminated, &PgSample_Item_2 },
           { eRpItemType_FormRadioButton, &PgSample_Item_3 },
           { eRpItemType_DataZeroTerminated, &PgSample_Item_4 },

/* Structures */

static rpRadioGroupInfo PgSample_Item_0 = {
           "RadioButtonGroup",
           &gRadioButtonGroup,
           &gRadioButtonGroup,
           eRpVarType_Direct,
           eRpVarType_Direct
};

static rpRadioButtonFormItem PgSample_Item_1 = {
           &PgSample_Item_0,
           "One",
           0
};

static char PgSample_Item_2[] =
```

```
            "One Radio Button\n";

static rpRadioButtonFormItem PgSample_Item_3 = {
            &PgSample_Item_0,
            "Another",
            1
};

static char PgSample_Item_4[] =
            "Another Radio Button" C_oBR;


/* Variables and Functions */

rpOneOfSeveral gRadioButtonGroup = 1;
```

Sample generated HTML

```
<INPUT TYPE="RADIO" NAME="RadioButtonGroup" VALUE="One">
One Radio Button
<INPUT TYPE="RADIO" NAME="RadioButtonGroup" VALUE="Another" CHECKED>
Another Radio Button<BR>
```

<INPUT TYPE=FILE>

HTML comment tag

```
<!-- RpFormInput TYPE=file NAME=n MaxLength=m Size=s -->
```

This tag is used in a form to define the field for selecting a file to upload. For HTTP file uploads to work, the `Enctype` keyword in the `RpFormHeader` tag must be specified as `MULTIPART/FORM-DATA`.

Function prototype

This element type has no access functions.

Commented HTML

```
<!-- RpFormInput TYPE="file" NAME="FileUpload" SIZE="15" MAXLENGTH="15" -->
<INPUT TYPE="FILE" NAME="FileUpload" SIZE="15" MAXLENGTH="15">
<!-- RpEnd -->
```

Internal structures

```
/* Element List Items */
          { eRpItemType_FormFile, &PgSample_Item_1 },
/* Structures */
static rpFileFormItem PgSample_Item_1 = {
          "FileUpload",
          15,
          15
};
```

Sample generated HTML

```
<INPUT TYPE="FILE" NAME="FileUpload" SIZE="15" MAXLENGTH="15">
```

<INPUT TYPE=BUTTON>

HTML comment tag

```
<!-- RpFormInput TYPE=button NAME=n RpGetType=gt RpGetPtr=gp
                            RpSetType=st RpSetPtr=sp -->
```

These input elements are usually used as place holders for JavaScript commands. The next table describes the associated keywords:

Keyword	Description
`Name`	Specifies the HTML name of the button. The value on the button is determined by the text functions described next.
`UseJavaScript`	If this keyword is enabled in the `PbSetUp.txt` file, PBuilder assumes that all unrecognized keywords are JavaScript and assigns storage in the structure for the JavaScript. The RomPager engine recreates the JavaScript when the element is displayed.

Function prototypes

```
GetType = Function
typedef char * (*rpFetchBytesFuncPtr)(void);
GetType = Complex

typedef char * (*rpFetchBytesComplexPtr)(void *theServerDataPtr,
char *theNamePtr, Signed16Ptr theIndexValuesPtr);
SetType = Function

typedef void (*rpStoreAsciiTextFuncPtr)(char *theValuePtr);

SetType = Complex

typedef void (*rpStoreAsciiTextComplexPtr)(void *theServerDataPtr,
char *theValuePtr, char * theNamePtr,
Signed16Ptr theIndexValuesPtr);
```

Commented HTML

```
<!-- RpFormInput TYPE=BUTTON NAME=MyButton RpGetType=Direct
RpGetPtr=gMyButton
          RpSetType=Direct RpSetPtr=gMyButton
onclick=ButtonClick(this.form) -->
<INPUT TYPE="BUTTON" NAME="MyButton" VALUE="Click Here"
          onclick=ButtonClick(this.form)>
<!-- RpEnd -->
```

Internal structures

```
/* Element List Items */
          { eRpItemType_FormButton, &PgSample_Item_1 },

/* Structures */

static rpCheckboxFormItem PgSample_Item_1 = {
          "MyButton",
          &gMyButton,
          &gMyButton,
          eRpVarType_Direct,
          eRpVarType_Direct,
```

```
            gJavaScript
};

/* Variables and Functions */

static char gMyButton[] = "Click Here";
static char gJavaScript[] = " onclick=ButtonClick(this.form)";
```

Sample generated HTML

```
<INPUT TYPE="BUTTON" NAME="MyButton" VALUE="Click Here"
            onclick=ButtonClick(this.form)>
```

<INPUT TYPE=RESET>
<INPUT TYPE=SUBMIT>

HTML comment tag

```
<!-- RpFormInput TYPE=submit VALUE=v -->
<!-- RpFormInput TYPE=reset VALUE=v -->
```

These tags define HTML Submit and Reset buttons. The `VALUE` keyword defines the label displayed on the button. Reset buttons are used for functions local to the browser; they never trigger server actions. The RomPager engine recognizes submit buttons, and the value is stored in the engine's data structure. If the device needs to know which of multiple Submit buttons was pressed, it uses the `RpGetSubmitButtonValue` callback routine to retrieve the stored value.

Function prototypes

There are no function calls with these elements.

Commented HTML

```
<!-- RpFormInput TYPE=SUBMIT VALUE="Submit Form"-->
    <INPUT TYPE="SUBMIT" NAME="Submit" VALUE="Submit Form">
<!-- RpEnd -->
<!-- RpFormInput TYPE=RESET VALUE="Reset Form" -->
    <INPUT TYPE="RESET" VALUE="Reset Form">
<!-- RpEnd -->
```

Internal structures

```
/* Element List Items */
          { eRpItemType_FormSubmit,&PgSample_Item_1 },
          { eRpItemType_FormReset,&PgSample_Item_2 },
/* Structures */

rpButtonFormItem PgSample_Item_1 = {
          "Submit Form"
};
rpButtonFormItem PgSample_Item_2 = {
          "Reset Form"
};
```

Sample generated HTML

```
<INPUT TYPE="SUBMIT" NAME="Submit" VALUE="Submit Form">
<INPUT TYPE="RESET" VALUE="Reset Form">
```

<INPUT TYPE=SUBMIT>

HTML comment tag

```
<!-- RpFormNamedSubmit NAME=n RpGetType=gt RpGetPtr=gp
                                    RpSetType=st RpSetPtr=sp -->
```

This tag defines HTML Submit buttons where the label displayed on the button is determined at run time. This tag also is useful for defining individually named buttons that can be referred to by JavaScript.

The HTML name of the button is specified by the `NAME` keyword. The value that is displayed on the button is determined by the text functions described next.

Function prototypes

```
GetType = Function
typedef char * (*rpFetchBytesFuncPtr)(void);

GetType = Complex
typedef char * (*rpFetchBytesComplexPtr)(void *theServerDataPtr,
 char *theNamePtr, Signed16Ptr theIndexValuesPtr);
```

```
SetType = Function
typedef void (*rpStoreAsciiTextFuncPtr)(char *theValuePtr);

SetType = Complex
typedef void (*rpStoreAsciiTextComplexPtr)(void *theServerDataPtr,
 char *theValuePtr, char * theNamePtr,
 Signed16Ptr theIndexValuesPtr);
```

Commented HTML

```
<!-- RpFormNamedSubmit NAME=Submit1 RpGetType=Direct RpGetPtr=gAddName
           RpSetType=Direct RpSetPtr=gAddName -->
    <INPUT TYPE="SUBMIT" NAME="Submit1" VALUE="Add">
<!-- RpEnd -->
<!-- RpFormNamedSubmit NAME=Submit2 RpGetType=Function
RpGetPtr=GetChangeName
           RpSetType=Function RpSetPtr=SetChangeName -->
    <INPUT TYPE="SUBMIT" NAME="Submit2" VALUE="Change">
<!-- RpEnd -->
<!-- RpFormNamedSubmit NAME=Submit3 RpGetType=Complex
RpGetPtr=GetDeleteName
           RpSetType=Complex RpSetPtr=SetDeleteName -->
    <INPUT TYPE="SUBMIT" NAME="Submit3" VALUE="Delete">
<!-- RpEnd -->
```

Internal structures

```
/* Element List Items */
           { eRpItemType_FormNamedSubmit, &PgSample_Item_1 },
           { eRpItemType_FormNamedSubmit, &PgSample_Item_2 },
           { eRpItemType_FormNamedSubmit, &PgSample_Item_3 },

/* Structures */

static rpCheckboxFormItem PgSample_Item_1[] = {
           "Submit1",
           &gAddName,
           &gAddName,
           eRpVarType_Direct,
           eRpVarType_Direct
};
```

```
static rpCheckboxFormItem PgSample_Item_2[] = {
          "Submit2",
          GetChangeName,
          SetChangeName,
          eRpVarType_Function,
          eRpVarType_Function
};
static rpCheckboxFormItem PgSample_Item_3[] = {
          "Submit3",
          GetDeleteName,
          SetDeleteName,
          eRpVarType_Complex,
          eRpVarType_Complex
};

/* Variables and Functions */
static char gAddName[15] = "Add";
static char gChangeName[15] = "Change";
static char gDeleteName[15] = "Delete";
char *GetChangeName(void) {
          return gChangeName;
}
void SetChangeName(char *theValuePtr) {
          strcpy(gChangeName, theValuePtr);
}
char *GetDeleteName(void *theServerDataPtr, char *theNamePtr,
                              Signed16Ptr theIndexValuesPtr) {
          return gDeleteName;
}
void SetDeleteName(void *theServerDataPtr, char *theValuePtr,
                              char *theNamePtr, Signed16Ptr
theIndexValuesPtr) {
          strcpy(gDeleteName, theValuePtr);
}
```

Sample generated HTML

```
<INPUT TYPE="SUBMIT" NAME="Submit1" VALUE="Add">
<INPUT TYPE="SUBMIT" NAME="Submit2" VALUE="Change">
<INPUT TYPE="SUBMIT" NAME="Submit3" VALUE="Delete">
```

<SELECT><OPTION> (Fixed list — single select)

HTML comment tag

```
<!-- RpFormSingleSelect NAME=n Size=s RpGetType=gt RpGetPtr=gp
                                RpSetType=st RpSetPtr=sp Dynamic -->
<!-- RpSingleSelectOption VALUE=v RpItemNumber=i -->
<!-- RpEndSelect -->
```

These tags are used for creating SELECT menus where the list of menu choices is known at compile time, and only a single choice can be made from the menu.

This table describes the tags:

Tag	Description
RpFormSingleSelect	Contains the information for generating the <SELECT> tag and processing the form value.
RpSingleSelectOption	Specifies options for the <SELECT> list. If the RpItemNumber keywords are not specified, PBuilder supplies default values starting at 1.
RpEndSelect	Terminates the options list of the <SELECT> menu.

This table describes the associated keywords:

Keyword	Description
Name	Specifies the HTML name of the item.
Size	Used for the <SELECT> tag SIZE keyword, which controls the size of the menu list. If Size is not specified, 1 is used. The access functions are used to retrieve and store a value that corresponds with a given option.
Dynamic	Used if the form element will be used in a repeat group

When a page is displayed, the RomPager engine uses the access functions described next to retrieve a value. It matches the retrieved value against the item number of each option in the SELECT list to determine which option to highlight. When a form is submitted, the RomPager engine uses the HTML name of the submitted option to determine the value to pass to the storage function.

Function prototypes

```
GetType = Function
typedef    Unsigned8 (*rpFetchRadioGroupFuncPtr)(void);

GetType = Complex
typedef    Unsigned8 (*rpFetchRadioGroupComplexPtr)(void
*theServerDataPtr,
char *theNamePtr, Signed16Ptr theIndexValuesPtr);

SetType = Function

typedef    void (*rpStoreRadioGroupFuncPtr)(Unsigned8 theValue);

SetType = Complex

typedef    void (*rpStoreRadioGroupComplexPtr)(void *theServerDataPtr,
Unsigned8 theValue, char *theNamePtr,
```

Commented HTML

```
<!-- RpFormSingleSelect NAME="Fixed Single Select" SIZE=4
           RpGetType=Direct RpGetPtr=gFixedSingleSelect
           RpSetType=Direct RpSetPtr=gFixedSingleSelect -->
           <SELECT NAME="Fixed Single Select" SIZE="4">
<!-- RpEnd -->
<!-- RpSingleSelectOption VALUE="Item 1" RpItemNumber=1 -->
           <OPTION>Item 1
<!-- RpEnd -->
<!-- RpSingleSelectOption VALUE="Item 2" RpItemNumber=2 -->
           <OPTION SELECTED>Item 2
<!-- RpEnd -->
<!-- RpSingleSelectOption VALUE="Item 3" RpItemNumber=3 -->
           <OPTION>Item 3
<!-- RpEnd -->
<!-- RpSingleSelectOption VALUE="Item 4" RpItemNumber=4 -->
           <OPTION>Item 4
<!-- RpEnd -->
<!-- RpSingleSelectOption VALUE="Item 5" RpItemNumber=5 -->
           <OPTION>Item 5
```

```
<!-- RpEnd -->
<!-- RpEndSelect -->
          </SELECT>
<!-- RpEnd -->
```

Internal structures

```
/* Element List Items */
          { eRpItemType_FormFixedSingleSelect, &PgSample_Item_1 },
/* Structures */

static rpOption_OneSelect PgSample_Item_1_Option5 = {
          (rpOption_OneSelectPtr) 0,
          "Item 5",
          5
};

static rpOption_OneSelect PgSample_Item_1_Option4 = {
          &PgSample_Item_1_Option5,
          "Item 4",
          4
};

static rpOption_OneSelect PgSample_Item_1_Option3 = {
          &PgSample_Item_1_Option4,
          "Item 3",
          3
};

static rpOption_OneSelect PgSample_Item_1_Option2 = {
          &PgSample_Item_1_Option3,
          "Item 2",
          2
};

static rpOption_OneSelect PgSample_Item_1_Option1 = {
          &PgSample_Item_1_Option2,
          "Item 1",
          1
};
```

```
static rpFixedSingleSelectFormItem PgSample_Item_1 = {
           "Fixed Single Select",
           &PgSample_Item_1_Option1,
           &gFixedSingleSelect,
           &gFixedSingleSelect,
           eRpVarType_Direct,
           eRpVarType_Direct,
           4
};

/* Variables and Functions */

rpOneOfSeveral gFixedSingleSelect = 2;
```

Sample generated HTML

```
<SELECT NAME="Fixed Single Select" SIZE="4">
<OPTION VALUE=1>Item 1
<OPTION VALUE=2 SELECTED>Item 2
<OPTION VALUE=3>Item 3
<OPTION VALUE=4>Item 4
<OPTION VALUE=5>Item 5
</SELECT>
```

<SELECT><OPTION> (Fixed list — multiple select)

HTML comment tag

```
<!-- RpFormMultiSelect NAME=n Size=s Dynamic -->
<!-- RpMultiSelectOption VALUE=v RpGetType=gt RpGetPtr=gp
                                 RpSetType=st RpSetPtr=sp -->
<!-- RpEndSelect -->
```

Use these tags to create SELECT menus where the list of menu choices is known at compile time and multiple choices can be made from the menu. This table describes the tags:

Tag	Description
RpFormMultiSelect	Contains the information for generating the <SELECT MULTIPLE> tag and processing the form value.
RpMultiSelectOption	Specifies the options for the <SELECT> list. Uses the functions described in the next section to access a Boolean value that determines whether an option is selected.
RpEndSelect	Terminates the options list of the <SELECT> menu.

These are the associated keywords:

Keyword	Description
Name	Specifies the HTML name of the item.
Size	Used for the <SELECT> tag SIZE keyword, which controls the size of the menu list. If the keyword is not specified, 1 is used.
Dynamic	Used if the form element is to be used in a repeat group.
Value	Specifies the display string of the option.

Like a check box, an HTML form with a SELECT MULTIPLE item submits only the options selected in the list. So the RomPager engine sets all options in the list to off at the beginning of form processing. As the form is processed, the individual options that are selected are set to on. Both the reset and set operations use the set function by specifying RpTextType specified by the comment tag.

Function prototypes

```
GetType = Function
typedef    Boolean (*rpFetchBooleanFuncPtr)(void);

GetType = Complex
typedef    Boolean (*rpFetchBooleanComplexPtr)(void *theServerDataPtr,
char *theNamePtr, Signed16Ptr theIndexValuesPtr);

SetType = Function
typedef void (*rpStoreBooleanFuncPtr)(Boolean theValue);
```

```
SetType = Complex
typedef    void (*rpStoreBooleanComplexPtr)(void *theServerDataPtr,
Boolean theValue, char *theNamePtr,
Signed16Ptr theIndexValuesPtr);
```

Commented HTML

```
<!-- RpFormMultiSelect NAME=FixedMultiple SIZE=0 -->
    <SELECT MULTIPLE NAME="FixedMultiple">
<!-- RpEnd -->
<!-- RpMultiSelectOption VALUE="Item 1" RpGetType=Direct RpGetPtr=gItem1
RpSetType=Direct RpSetPtr=gItem1 -->
    <OPTION>Item 1
<!-- RpEnd -->
<!-- RpMultiSelectOption VALUE="Item 2" RpGetType=Direct RpGetPtr=gItem2
RpSetType=Direct RpSetPtr=gItem2 -->
    <OPTION SELECTED>Item 2
<!-- RpEnd -->
<!-- RpMultiSelectOption VALUE="Item 3" RpGetType=Direct RpGetPtr=gItem3
RpSetType=Direct RpSetPtr=gItem3 -->
    <OPTION SELECTED>Item 3
<!-- RpEnd -->
<!-- RpMultiSelectOption VALUE="Item 4" RpGetType=Direct RpGetPtr=gItem4
RpSetType=Direct RpSetPtr=gItem4 -->
    <OPTION>Item 4
<!-- RpEnd -->
<!-- RpMultiSelectOption VALUE="Item 5" RpGetType=Direct RpGetPtr=gItem5
RpSetType=Direct RpSetPtr=gItem5 -->
    <OPTION>Item 5
<!-- RpEnd -->
<!-- RpEndSelect -->
          </SELECT>
<!-- RpEnd -->
```

Internal structures

```
/* Element List Items */
          { eRpItemType_FormFixedMultiSelect, &PgSample_Item_1 },

/* Structures */
```

```
static rpOption_MultiSelect PgSample_Item_1_Option5 = {
          (rpOption_MultiSelectPtr) 0,
          "Item 5",
          &gItem5,
          &gItem5,
          eRpVarType_Direct,
          eRpVarType_Direct
};

static rpOption_MultiSelect PgSample_Item_1_Option4 = {
          &PgSample_Item_1_Option5,
          "Item 4",
          &gItem4,
          &gItem4,
          eRpVarType_Direct,
          eRpVarType_Direct
};

static rpOption_MultiSelect PgSample_Item_1_Option3 = {
          &PgSample_Item_1_Option4,
          "Item 3",
          &gItem3,
          &gItem3,
          eRpVarType_Direct,
          eRpVarType_Direct
};

static rpOption_MultiSelect PgSample_Item_1_Option2 = {
          &PgSample_Item_1_Option3,
          "Item 2",
          &gItem2,
          &gItem2,
          eRpVarType_Direct,
          eRpVarType_Direct
};

static rpOption_MultiSelect PgSample_Item_1_Option1 = {
          &PgSample_Item_1_Option2,
          "Item 1",
```

```
            &gItem1,
            &gItem1,
            eRpVarType_Direct,
            eRpVarType_Direct
};

static rpFixedMultiSelectFormItem PgSample_Item_1 = {
            "FixedMultiple",
            &PgSample_Item_1_Option1,
            0
};
/* Variables and Functions */

static Boolean gItem1 = False;
static Boolean gItem2 = True;
static Boolean gItem3 = True;
static Boolean gItem4 = False;
static Boolean gItem5 = False;
```

Sample generated HTML

```
<SELECT MULTIPLE NAME="FixedMultiple">
<OPTION>Item 1
<OPTION SELECTED>Item 2
<OPTION SELECTED>Item 3
<OPTION>Item 4
<OPTION>Item 5
</SELECT>
```

<SELECT><OPTION> (Variable list — string values)

HTML comment tag

```
<!-- RpFormVariableSelect NAME=n MULTIPLE RpResetPtr=r
                                RpGetType=gt RpGetPtr=gp RpSetType=st
RpSetPtr=sp
                                Size=s RpTextType=t RpFieldWidth=f Dynamic
-->
```

Use this tag to create <SELECT> menus where the list of menu choices is created at run time. The values retrieved by the access functions are used to create the list of choices, and the selected values are returned to the storage functions when the form is processed.

The RpFormVariableSelect tag contains the information for generating the <SELECT> and <OPTION> tags and for processing the form value.

This table describes the associated keywords:

Keyword	Description
NAME	Specifies the HTML name of the item.
Size	Used for the <SELECT> tag SIZE keyword, which controls the size of the menu list. If the keyword is not specified, 1 is used.
MULTIPLE	Specifies that the MULTIPLE keyword is to be generated for the <SELECT> tag, which allows multiple options to be selected from the list. If the MULTIPLE keyword is present, the RpResetPtr keyword also must be supplied to specify a pointer to a function that is called to reset the options.
RpFieldWidth	Used for conversions when RpTextType is ASCIIFixed, Hex, HexColonForm, or DotForm to specify the number of characters that should appear on the page for the field.
Dynamic	Used if the form element is used in a repeat group.

The engine calls the Get function once to get the value to display for each <OPTION> string until the function returns a null pointer to signal that there are no more <OPTION> strings.

The engine increments theItemNumber each time the Get function is called. The values returned to the Set function during form processing are the values that the Get function supplied for the selected options.

Function prototypes

```
GetType = Function
typedef    void * (*rpFetchOptionFuncPtr)(Unsigned8 theItemNumber,
Boolean *theOptionSelectedFlag);

GetType = Complex
typedef    void * (*rpFetchOptionComplexPtr)(void *theServerDataPtr,
```

```
char *theNamePtr,
Signed16Ptr theIndexValuesPtr,
Unsigned8 theItemNumber,
Boolean *theOptionSelectedFlag);
Reset Function

typedef    void (*rpResetVarSelectPtr)(void *theServerDataPtr);
```

The prototypes used for setting the value of the selected option are the same as the `SetType` prototypes used by the `<INPUT TYPE=TEXT>` tags. The actual prototype used depends on the value of the `RpSetType` and `RpTextType` keywords.

Commented HTML

```
<!-- RpFormVariableSelect NAME=VariableSingle SIZE=16 RpTextType=ASCII
           RpGetType=Function RpGetPtr=VariableSingleGet
           RpSetType=Function RpSetPtr=VariableSinglePut -->
    <SELECT NAME="VariableSingle" SIZE="16">
    <OPTION>One
    <OPTION SELECTED>Two
    <OPTION>Three
    <OPTION>Four
    <OPTION>Five
    <OPTION>Six
    <OPTION>Seven
    <OPTION>Eight
    </SELECT>
<!-- RpEnd -->

<!-- RpFormVariableSelect NAME=VariableMultiple MULTIPLE SIZE=0
RpFieldWidth=15 RpTextType=DotForm RpResetPtr=VariableMultipleReset
RpGetType=Function RpGetPtr=VariableMultipleGet
RpSetType=Function RpSetPtr=VariableMultiplePut -->
    <SELECT MULTIPLE NAME="VariableMultiple">
    <OPTION>0.0.0.0
    <OPTION>1.1.1.1
    <OPTION SELECTED>2.2.2.2
    <OPTION SELECTED>3.3.3.3
    <OPTION>4.4.4.4
    <OPTION SELECTED>7.7.7.7
```

```
    <OPTION SELECTED>8.8.8.8
    <OPTION>15.15.15.15
    <OPTION>16.16.16.16
    <OPTION>32.32.32.32
    <OPTION>32.32.32.32
    <OPTION>63.63.63.63
    <OPTION>64.64.64.64
    <OPTION>127.127.127.127
    <OPTION>128.128.128.128
    <OPTION>255.255.255.255
    </SELECT>
<!-- RpEnd -->
```

Internal structures

```
/* Element List Items */
           { eRpItemType_FormVariableSingleSelect,&PgSample_Item_1 },
           { eRpItemType_FormVariableMultiSelect,&PgSample_Item_2 },

/* Structures */

static rpVariableSelectFormItem PgSample_Item_1 = {
           "VariableSingle",
           (rpResetVarSelectPtr) 0,
           VariableSingleGet,
           VariableSinglePut,
           eRpVarType_Function,
           eRpVarType_Function,
           eRpTextType_ASCII,
           0,
           16
};
static rpVariableSelectFormItem PgSample_Item_2 = {
           "VariableMultiple",
           VariableMultipleReset,
           VariableMultipleGet,
           VariableMultiplePut,
           eRpVarType_Function,
           eRpVarType_Function,
           eRpTextType_DotForm,
```

```
            15,
            0
};

/* Variables and Functions */

#define kVariableSingleCount 8
#define kVariableMultipleCount16

static Unsigned8 gVariableSingleSelect = 2;
static char *gVariableSingleSelectItems[kVariableSingleCount] = {
            "One",
            "Two",
            "Three",
            "Four",
            "Five",
            "Six",
            "Seven",
            "Eight"
};

typedef    struct {
Boolean                          fSelected;
Unsigned8                     fValue[4];
} VariableMultiSelect, *VariableMultiSelectPtr;

static VariableMultiSelect gVariableMultiSelect[kVariableMultipleCount] =
{
            { False, {   0,   0,   0,   0 } },
            { False, {   1,   1,   1,   1 } },
            { True,  {   2,   2,   2,   2 } },
            { True,  {   3,   3,   3,   3 } },
            { False, {   4,   4,   4,   4 } },
            { True,  {   7,   7,   7,   7 } },
            { True,  {   8,   8,   8,   8 } },
            { False, {  15,  15,  15,  15 } },
            { False, {  16,  16,  16,  16 } },
            { False, {  32,  32,  32,  32 } },
            { False, {  32,  32,  32,  32 } },
```

```
            { False, {  63,  63,  63,  63 } },
            { False, {  64,  64,  64,  64 } },
            { False, { 127, 127, 127, 127 } },
            { False, { 128, 128, 128, 128 } },
            { False, { 255, 255, 255, 255 } }
};
static void *VariableSingleGet(Unsigned8 theItemNumber,
Boolean *theOptionSelectedFlag) {
*theOptionSelectedFlag = (theItemNumber + 1) == gVariableSingleSelect;
return (theItemNumber < kVariableSingleCount) ?
gVariableSingleSelectItems[theItemNumber] : (void *) 0;
}
static voidVariableSinglePut(char *theValuePtr) {
int theItemsIndex;
theItemsIndex = 0;
while (theItemsIndex < kVariableSingleCount &&
strcmp(theValuePtr, gVariableSingleSelectItems[theItemsIndex])) {
theItemsIndex += 1;
}
gVariableSingleSelect = theItemsIndex + 1;
return;
}
static void *VariableMultipleGet(Unsigned8 theItemNumber,
Boolean *theOptionSelectedFlag) {
*theOptionSelectedFlag = gVariableMultiSelect[theItemNumber].fSelected;
return (theItemNumber < kVariableMultipleCount) ?
&gVariableMultiSelect[theItemNumber].fValue : (void *) 0;
}
static voidVariableMultipleReset(void *theServerDataPtr) {
            int              theIndex;
            VariableMultiSelectPtrtheVariableMultiSelectPtr;


            theVariableMultiSelectPtr = gVariableMultiSelect;
            for (theIndex = 0; theIndex < kVariableMultipleCount; theIndex
+= 1) {
                              theVariableMultiSelectPtr->fSelected = False;
                              theVariableMultiSelectPtr += 1;
            }
```

```
            return;

static voidVariableMultiplePut(char *theValuePtr) {
            int              theIndex;
            VariableMultiSelectPtrtheVariableMultiSelectPtr;
            Boolean          theFoundFlag;
            theVariableMultiSelectPtr = gVariableMultiSelect;
            theIndex = 0;
            theFoundFlag = False;
            while (!theFoundFlag && theIndex < kVariableMultipleCount) {
                             if (!memcmp(theValuePtr,
theVariableMultiSelectPtr->fValue,
                                        sizeof(theVariableMultiSelectPtr-
>fValue))) {
                                 theVariableMultiSelectPtr->fSelected =
True;
                                 theFoundFlag = True;
                             }
                             else {
                                 theVariableMultiSelectPtr += 1;
                                 theIndex                  += 1;
                             }
            }
            return;
}
```

Sample generated HTML

```
<SELECT NAME="VariableSingle" SIZE="16">
<OPTION>One
<OPTION SELECTED>Two
<OPTION>Three
<OPTION>Four
<OPTION>Five
<OPTION>Six
<OPTION>Seven
<OPTION>Eight
</SELECT>
<SELECT MULTIPLE NAME="VariableMultiple">
<OPTION>0.0.0.0
```

```
<OPTION>1.1.1.1
<OPTION SELECTED>2.2.2.2
<OPTION SELECTED>3.3.3.3
<OPTION>4.4.4.4
<OPTION SELECTED>7.7.7.7
<OPTION SELECTED>8.8.8.8
<OPTION>15.15.15.15
<OPTION>16.16.16.16
<OPTION>32.32.32.32
<OPTION>32.32.32.32
<OPTION>63.63.63.63
<OPTION>64.64.64.64
<OPTION>127.127.127.127
<OPTION>128.128.128.128
<OPTION>255.255.255.255
</SELECT>
```

<SELECT><OPTION> (Variable list — item number values)

HTML comment tag

```
<!-- RpFormVarValueSelect NAME=n MULTIPLE RpResetPtr=r
                                RpGetType=gt RpGetPtr=gp RpSetType=st
RpSetPtr=sp
                                Size=s RpTextType=t RpFieldWidth=f Dynamic
-->
```

Use this tag to create <SELECT> menus where the list of menu choices is created at run time. The values retrieved by the access functions are used to create the list of choices, and the item number of the choice is returned to the storage function when the form is processed.

The RpFormVarValueSelect tag contains the information for generating the <SELECT> and <OPTION> tags and for processing the form value.

This table describes the keywords:

Keyword	Description
Name	Specifies the HTML name of an item.
Size	Used for the <SELECT> tag SIZE keyword, which controls the size of the menu list. If the keyword is not specified, 1 is used.
MULTIPLE	Specifies that the MULTIPLE keyword is to be generated for the <SELECT> tag, which allows multiple options to be selected from the list. If the MULTIPLE keyword is present, the RpResetPtr keyword must also be supplied to specify a pointer to a function that is called to reset the options.
RpFieldWidth	Used for conversions when RpTextType is ASCIIFixed, Hex, HexColonForm, or DotForm to specify the number of characters that should appear on the page for the field
Dynamic	Used if the form element is used in a repeat group.

The engine calls the application's Get function to retrieve the label to display for the <OPTION> string. The Get function is called once for each <OPTION> string until the function returns a null pointer to signal that there are no more <OPTION> strings. The engine increments theItemNumber each time the Get function is called.

When the application's Get function returns the label to display for the <OPTION> string, it also returns an arbitrary unsigned 32-bit value that is associated with the display string. This unsigned 32-bit value can be a numeric, a structure pointer, or any other value that the application finds useful. The engine encodes the value using 8 hex digits in the VALUE keyword of the <OPTION> tag.

When the browser submits the form, the engine takes the eight hexadecimal digits and turns it back into the unsigned 32 bit value. In turn, this value is passed to the Set function during form processing for the application to handle. With some systems, the encoded value can appear to be byte-swapped, but the value is reassembled correctly at form processing time. In this way, the value returned to the Set function is the unsigned 32-bit value associated with the <OPTION> display string rather than the display string. This capability can be used to simplify the form processing routines used for variable list menus.

Function prototypes

```
GetType = Function
typedef void * (*rpFetchOptionValueFuncPtr)(Unsigned8 theItemNumber,
                                            Boolean *theOptionSelectedFlag,
                                            Unsigned32Ptr theValuePtr);

GetType = Complex

typedef void * (*rpFetchOptionValueComplexPtr)(void *theServerDataPtr,
                                            char *theNamePtr,
                                            Signed16Ptr theIndexValuesPtr,
                                            Unsigned8 theItemNumber,
                                            Boolean *theOptionSelectedFlag,
                                            Unsigned32Ptr theValuePtr);

Reset Function

typedef void (*rpResetVarSelectPtr)(void *theServerDataPtr);

SetType = Function

typedef void (*rpStoreUnsigned32FuncPtr)(Unsigned32 theValue);

SetType = Complex

typedef void (*rpStoreUnsigned32ComplexPtr)(void *theServerDataPtr,
        Unsigned32 theValue, char *theNamePtr, Signed16Ptr
theIndexValuesPtr);
```

Commented HTML

```
<!-- RpFormVarValueSelect NAME=VariableValueSingle SIZE=1 RpTextType=ASCII
                                RpGetType=Function
RpGetPtr=VariableValueSingleGet
                                RpSetType=Function
RpSetPtr=VariableValueSinglePut -->
    <SELECT NAME="VariableValueSingle" SIZE="1">
    <OPTION VALUE=00000001>One Hundred
    <OPTION VALUE=00000002 SELECTED>Two Hundred
    <OPTION VALUE=00000003>Three Hundred
```

```
    <OPTION VALUE=00000004>Four Hundred
    <OPTION VALUE=00000005>Five Hundred
    </SELECT>
<!-- RpEnd -->
<!-- RpFormVarValueSelect NAME=VariableValueMultiple MULTIPLE SIZE=0
           RpFieldWidth=15 RpTextType=DotForm
RpResetPtr=VarValueMultipleReset
           RpGetType=Function RpGetPtr=VarValueMultipleGet
           RpSetType=Function RpSetPtr=VarValueMultiplePut -->
    <SELECT MULTIPLE NAME="VariableValueMultiple">
    <OPTION VALUE=00000000>0.0.0.0
    <OPTION VALUE=00000001>1.1.1.1
    <OPTION VALUE=00000002 SELECTED>2.2.2.2
    <OPTION VALUE=00000003 SELECTED>3.3.3.3
    <OPTION VALUE=00000004>4.4.4.4
    <OPTION VALUE=00000005 SELECTED>7.7.7.7
    <OPTION VALUE=00000006 SELECTED>8.8.8.8
    <OPTION VALUE=00000007>15.15.15.15
    <OPTION VALUE=00000008>16.16.16.16
    <OPTION VALUE=00000009>32.32.32.32
    <OPTION VALUE=0000000a>32.32.32.32
    <OPTION VALUE=0000000b>63.63.63.63
    <OPTION VALUE=0000000c>64.64.64.64
    <OPTION VALUE=0000000d>127.127.127.127
    <OPTION VALUE=0000000e>128.128.128.128
    <OPTION VALUE=0000000f>255.255.255.255
    </SELECT>
<!-- RpEnd -->
```

Internal structures

```
/* Element List Items */
           { eRpItemType_FormVarValueSingleSelect,&PgSample_Item_1 },
           { eRpItemType_FormVarValueMultiSelect,&PgSample_Item_2 },

/* Structures */

static rpVariableSelectFormItem PgSample_Item_1 = {
           "VariableValueSingle",
           (rpResetVarSelectPtr) 0,
```

```
            VariableValueSingleGet,
            VariableValueSinglePut,
            eRpVarType_Function,
            eRpVarType_Function,
            eRpTextType_ASCII,
            0,
            1
};
static rpVariableSelectFormItem PgSample_Item_2 = {
            "VariableValueMultiple",
            VarValueMultipleReset,
            VarValueMultipleGet,
            VarValueMultiplePut,
            eRpVarType_Function,
            eRpVarType_Function,
            eRpTextType_DotForm,
            15,
            0
};

/* Variables and Functions */

#define kVariableValueSingleCount 5
#define kVariableMultipleCount16

static Unsigned8 gVarValueSingleSelect = 2;

static char *gVarValueSingleSelectItems[kVariableValueSingleCount] = {
            "One Hundred",
            "Two Hundred",
            "Three Hundred",
            "Four Hundred",
            "Five Hundred",
};

typedef struct {
            Boolean          fSelected;
            Unsigned8        fValue[4];
} VariableMultiSelect, *VariableMultiSelectPtr;
```

```
static VariableMultiSelect gVariableMultiSelect[kVariableMultipleCount] =
{
          { False, {   0,   0,   0,   0 } },
          { False, {   1,   1,   1,   1 } },
          { True,  {   2,   2,   2,   2 } },
          { True,  {   3,   3,   3,   3 } },
          { False, {   4,   4,   4,   4 } },
          { True,  {   7,   7,   7,   7 } },
          { True,  {   8,   8,   8,   8 } },
          { False, {  15,  15,  15,  15 } },
          { False, {  16,  16,  16,  16 } },
          { False, {  32,  32,  32,  32 } },
          { False, {  32,  32,  32,  32 } },
          { False, {  63,  63,  63,  63 } },
          { False, {  64,  64,  64,  64 } },
          { False, { 127, 127, 127, 127 } },
          { False, { 128, 128, 128, 128 } },
          { False, { 255, 255, 255, 255 } }
};

static void *VariableValueSingleGet(Unsigned8 theItemNumber,
                                    Boolean *theOptionSelectedFlag,
                                    Unsigned32Ptr theValuePtr) {
          *theOptionSelectedFlag = (theItemNumber + 1) ==
gVarValueSingleSelect;
          *theValuePtr = theItemNumber + 1;
          return (theItemNumber < kVariableSingleCount) ?
                         gVarValueSingleSelectItems[theItemNumber] :
(void *) 0;
}

static voidVariableValueSinglePut(Unsigned32 theValue) {

          gVarValueSingleSelect = (Unsigned8) theValue;
          return;
}

static void *VarValueMultipleGet(Unsigned8 theItemNumber,
                                    Boolean *theOptionSelectedFlag,
```

```
                                                    Unsigned32Ptr theValuePtr) {
            *theOptionSelectedFlag =
gVariableMultiSelect[theItemNumber].fSelected;
            *theValuePtr = theItemNumber;
            return (theItemNumber < kVariableMultipleCount) ?

&gVariableMultiSelect[theItemNumber].fValue : (void *) 0;
}
static voidVarValueMultipleReset(void *theServerDataPtr) {
            int                          theIndex;
            VariableMultiSelectPtrtheVariableMultiSelectPtr;

            theVariableMultiSelectPtr = gVariableMultiSelect;
            for (theIndex = 0; theIndex < kVariableMultipleCount; theIndex
+= 1) {
                              theVariableMultiSelectPtr->fSelected = False;
                              theVariableMultiSelectPtr += 1;
            }
            return;
}
static voidVarValueMultiplePut(Unsigned32 theValue) {
            gVariableMultiSelect[theValue].fSelected = True;
            return;
}
```

Sample generated HTML

```
<SELECT NAME="VariableValueSingle" SIZE="1">
<OPTION VALUE=00000001>One Hundred
<OPTION VALUE=00000002 SELECTED>Two Hundred
<OPTION VALUE=00000003>Three Hundred
<OPTION VALUE=00000004>Four Hundred
<OPTION VALUE=00000005>Five Hundred
</SELECT>
<SELECT MULTIPLE NAME="VariableValueMultiple">
<OPTION VALUE=00000000>0.0.0.0
<OPTION VALUE=00000001>1.1.1.1
<OPTION VALUE=00000002 SELECTED>2.2.2.2
<OPTION VALUE=00000003 SELECTED>3.3.3.3
<OPTION VALUE=00000004>4.4.4.4
```

```
<OPTION VALUE=00000005 SELECTED>7.7.7.7
<OPTION VALUE=00000006 SELECTED>8.8.8.8
<OPTION VALUE=00000007>15.15.15.15
<OPTION VALUE=00000008>16.16.16.16
<OPTION VALUE=00000009>32.32.32.32
<OPTION VALUE=0000000a>32.32.32.32
<OPTION VALUE=0000000b>63.63.63.63
<OPTION VALUE=0000000c>64.64.64.64
<OPTION VALUE=0000000d>127.127.127.127
<OPTION VALUE=0000000e>128.128.128.128
<OPTION VALUE=0000000f>255.255.255.255
</SELECT>
```

<TEXTAREA>

HTML comment tags

```
<!-- RpFormTextArea NAME=n RpGetType=gt RpGetPtr=gp ROWS=r COLS=c -->

<!-- RpFormTextAreaBuf NAME=n RpGetType=gt RpGetPtr=gp
                                RpSetType=gt RpSetPtr=gp ROWS=r COLS=c -->
```

Use these tags in a form to define <TEXTAREA> elements. This table describes the associated keywords:

Keyword	Description
NAME	Specifies the HTML name of the text area.
ROWS	Used for the <TEXTAREA> tag ROWS keyword. If the keyword is not specified, 1 is used.
COLS	Used for the <TEXTAREA> tag COLS keyword. If the keyword is not specified, 40 is used.

The RpFormTextArea tag is used to build a display-only text area. This table describes its associated keywords:

Keyword	Description
RpGetType	Must be either Function or Complex
RpGetPtr	Points to a function that retrieves one line at a time

The RomPager engine calls the `Get` function once to get each display line of the text area until the function returns a null pointer to signal that there are no more display lines. The engine increments `theItemNumber` each time the `Get` function is called.

The `RpFormTextAreaBuf` tag is used to build a text area by using a display buffer. This table describes the keywords associated with the `RpFormTextAreaBuf` tag:

Keyword	Description
`RpGetType` and `RpSetType`	Must be either `Function` or `Complex`
`RpGetPtr` and `RpSetPtr`	Point to functions that access and store the complete text area buffer

Function prototypes

```
GetType = Function - RpFormTextArea

typedef char * (*rpFetchTextFuncPtr)(Unsigned16 theItemNumber);

GetType = Complex - RpFormTextArea

typedef char * (*rpFetchTextComplexPtr)(void *theServerDataPtr,
char *theNamePtr,
Signed16Ptr theIndexValuesPtr,
Unsigned16 theItemNumber);
GetType = Function - RpFormTextAreaBuf

typedef char * (*rpFetchBytesFuncPtr)(void);

GetType = Complex - RpFormTextAreaBuf

typedef char * (*rpFetchBytesComplexPtr)(void *theServerDataPtr,
char *theNamePtr,
Signed16Ptr theIndexValuesPtr);
SetType = Function - RpFormTextAreaBuf

typedef void (*rpStoreAsciiTextFuncPtr)(char *theValuePtr);

SetType = Complex - RpFormTextAreaBuf
```

```
typedef void(*rpStoreAsciiTextComplexPtr)(void *theServerDataPtr,
char *theValuePtr,
char * theNamePtr,
Signed16Ptr theIndexValuesPtr);
```

Commented HTML

```
<!-- RpFormTextArea NAME=TextArea ROWS=4 COLS=60
          RpGetType=Function RpGetPtr=TextAreaGet -->
    <TEXTAREA NAME="TextArea" ROWS="4" COLS="60">
    Initial Text Line 1
    Initial Text Line 2
    Initial Text Line 3
    Initial Text Line 4
    Initial Text Line 5
    Initial Text Line 6
    </TEXTAREA>
<!-- RpEnd -->

<!-- RpFormTextAreaBuf NAME=TextArea ROWS=3 COLS=55
          RpGetType=Function RpGetPtr=TextAreaBufGet
          RpSetType=Function RpSetPtr=TextAreaBufPut -->
    <TEXTAREA NAME="TextArea" ROWS="3" COLS="55">
    This text is read and written from a single buffer. The
    buffer must be smaller than kHttpWorkSize and the write
    function needs to check for buffer overflow.
    </TEXTAREA>
<!-- RpEnd -->
```

Internal structures

```
/* Element List Items */
          { eRpItemType_FormTextArea, &PgSample_Item_1 },
          { eRpItemType_FormTextAreaBuf,&PgSample_Item_2 },

/* Structures */

static rpTextAreaFormItem PgSample_Item_1 = {
          "TextArea",
```

```
            TextAreaGet,
            (void *) 0,
            eRpVarType_Function,
            eRpVarType_Direct,
            4,
            60
};
static rpTextAreaFormItem PgSample_Item_2 = {
            "TextArea",
            TextAreaBufGet,
            TextAreaBufPut,
            eRpVarType_Function,
            eRpVarType_Function,
            3,
            55
};
/* Variables and Functions */

#define kTextAreaCount          7
#define kTextAreaBufferSize  200

static char *TextArea[kTextAreaCount] = {
            "Initial Text Line 1",
            "Initial Text Line 2",
            "Initial Text Line 3",
            "Initial Text Line 4",
            "Initial Text Line 5",
            "Initial Text Line 6"
};
static char TextAreaBufBuffer[kTextAreaBufferSize] =
    "This text is read and written from a single buffer. The\n"
    "buffer must be smaller than kHttpWorkSize and the write\n"
    "function needs to check for buffer overflow.";

char *TextAreaGet(Unsigned16 theItemNumber) {
            return TextArea[theItemNumber];
}

char *TextAreaBufGet(void) {
```

```
            return TextAreaBufBuffer;
}

void TextAreaBufPut(char *theValue) {
            Unsigned16       theInputSize;

            theInputSize = strlen(theValue);
            if (theInputSize > kTextAreaBufferSize) {
                        *(theValue + kTextAreaBufferSize - 1) = '\0';
            }
            strcpy(TextAreaBufBuffer, theValue);

}
```

Sample generated HTML

```
<TEXTAREA NAME="TextArea" ROWS="4" COLS="60">
Initial Text Line 1
Initial Text Line 2
Initial Text Line 3
Initial Text Line 4
Initial Text Line 5
Initial Text Line 6
</TEXTAREA>

<TEXTAREA NAME="TextArea" ROWS="3" COLS="55">
This text is read and written from a single buffer. The
buffer must be smaller than kHttpWorkSize and the write
function needs to check for buffer overflow.
</TEXTAREA>
```

Item groups

Sometimes it is useful to group a series of elements that are used on more than one page.

```
<!-- RpItemGroup RpIdentifier=i -->


<!-- RpLastItemInGroup -->
```

These tags define a group of items that are used more than once. By grouping common sets of items, you can save a significant amount of page storage memory. You can place any other page element, including static text, dynamic text, and form elements, in an element group.

The `RpItemGroup` tag defines the beginning of a common element group. The `RpLastItemInGroup` tag defines the end of the common group.

You can place element groups within element groups for complex structures. The nesting level is `5`. No checking for recursion is done, so it is possible to set up items groups that point to themselves, generating an infinite amount of HTML.

Commented HTML

```
<!-- RpItemGroup RpIdentifier=PgEndItems -->
           <!-- RpDataZeroTerminated -->
           <BR>
           <BR>
           <!-- RpEnd -->
           <!-- RpFormInput TYPE=SUBMIT NAME=Submit VALUE=Submit -->
           <INPUT TYPE="SUBMIT" NAME="Submit" VALUE="Submit">
           <!-- RpEnd -->
           <!-- RpFormInput TYPE=RESET VALUE=Reset -->
           <INPUT TYPE="RESET" VALUE="Reset">
           <!-- RpEnd -->

           <!-- RpDataZeroTerminated -->
           <P> </P>
           Return to: <A HREF="ValidationSuite">Validation Main Page</A>
           </FORM>
           </BODY>
           </HTML>
           <!-- RpEnd -->
<!-- RpLastItemInGroup -->
```

Internal structures

```
/* Element List Items */
          { eRpItemType_ItemGroup, &PgEndItems },

/* Structures */

static rpItem PgEndItems[] = {
          { eRpItemType_DataZeroTerminated,&PgItemGroup_1 },
          { eRpItemType_FormSubmit,&PgItemGroup_2 },
          { eRpItemType_FormReset,&PgItemGroup_3 },
          { eRpItemType_DataZeroTerminated,&PgItemGroup_4 },
          { eRpItemType_LastItemInList }
};

static char PgItemGroup_1[] =
          C_oBR C_oBR;

static rpButtonFormItem PgItemGroup_2 = {
          "Submit"
};

static rpButtonFormItem PgItemGroup_3 = {
          "Reset"
};

static char PgItemGroup_4[] =
          C_oP_NBSP_xP
          "Return to: C_oANCHOR_HREF "ValidationSuite\">Validation Main
Page" C_xANCHOR
          C_xFORM C_xBODY_xHTML;
```

Sample generated HTML

```
<BR>
<BR>
<INPUT TYPE="SUBMIT" NAME="Submit" VALUE="Submit">
<INPUT TYPE="RESET" VALUE="Reset">
<P> </P>
Return to: <A HREF="ValidationSuite">Validation Main Page</A>
```

```
</FORM>
</BODY>
</HTML>
```

Dynamic HTML

Using the RomPager dynamic HTML capability allows a page to display different HTML at different times, depending on the current values in the device. This can be useful for displaying status messages and creating dynamic tables.

HTML comment tag

```
<!-- RpDynamicDisplay RpItemCount=n RpGetType=gt RpGetPtr=gp RpGroupPtr=g -
->
<!-- RpLastItemInGroup -->
```

The `RpDynamicDisplay` tag generates different HTML depending on the state of an internal variable accessed by the `Get` function.

The `Get` function returns a variable that is used as an index into a group of items, thereby choosing which HTML to generate.

This table describes the associated keywords:

Keyword	Description
`RpItemCount`	Specifies the number of elements in the display list and is used internally to validate the value returned by the `Get` function. The group of display list items can be specified by items that follow the `RpDynamicDisplay` tag, with the `RpLastItemInGroup` tag ending the group.
`RpGroupPrt`	If present, specifies the identifier of a previously defined display group of items.

Function prototypes

```
GetType = Function
typedef Unsigned8 (*rpFetchIndexFuncPtr)(void);
GetType = Complex
typedef Unsigned8 (*rpFetchIndexComplexPtr)(void *theServerDataPtr,
                                            Signed16Ptr theIndexValuesPtr);
```

Commented HTML

```
<!-- RpDynamicDisplay RpItemCount=3
          RpGetType=Function RpGetPtr=GetPrintQueueStatus -->
<!-- RpDataZeroTerminated -->
    <BR>The print queue is empty.<BR>
<!-- RpEnd -->
<!-- RpDataZeroTerminated -->
    <BR>The print queue is full.<BR>
<!-- RpEnd -->
          <!-- RpItemGroup -->
          <!-- RpDataZeroTerminated -->
                          <BR>There are
          <!-- RpEnd -->
          <!-- RpDisplayText RpGetPtr=PrintQueueCount
RpTextType=Unsigned16 -->
          6
          <!-- RpEnd -->
          <!-- RpDataZeroTerminated -->
                          jobs in the print queue.<BR>
          <!-- RpEnd -->
          <!-- RpLastItemInGroup -->
<!-- RpLastItemInGroup -->
```

Internal structures

```
/* Element List Items */
          { eRpItemType_DynamicDisplay, &PgSample_Item_1 },
/* Structures */

static rpDynamicDisplayItem PgSample_Item_1 = {
          GetPrintQueueStatus,
          eRpVarType_Function,
          3,
          PgSample_Item_1_Group
};

static rpItem PgSample_Item_1_Group[] = {
          { eRpItemType_DataZeroTerminated,&PgSample_Item_2 },
          { eRpItemType_DataZeroTerminated,&PgSample_Item_3 },
```

```
            { eRpItemType_ItemGroup,&PgSample_Item_4 },
            { eRpItemType_LastItemInList }
};

static char PgSample_Item_2[] =
            C_oBR "The print queue is empty." C_oBR;

static char PgSample_Item_3[] =
            C_oBR "The print queue is full." C_oBR;

static rpItem PgSample_Item_4[] = {
            { eRpItemType_DataZeroTerminated,&PgSample_Item_5 },
            { eRpItemType_DisplayText,&PgSample_Item_6 },
            { eRpItemType_DataZeroTerminated,&PgSample_Item_7 },
            { eRpItemType_LastItemInList }
};

static char PgSample_Item_5[] =
            C_oBR "There are ";

static rpTextDisplayItem PgSample_Item_6 = {
            &PrintQueueCount,
            eRpVarType_Direct,
            eRpTextType_Unsigned16,
            20
};

static char PgSample_Item_7[] =
            " jobs in the print queue."
            C_oBR;

/* Variables and Functions */

#define kMaxPrintQueueCount  300

Unsigned8 GetPrintQueueStatus(void) {

            if (PrintQueueCount == 0) {
                        return 0;
```

```
            }
            else if (PrintQueueCount >= kMaxPrintQueueCount) {
                        return 1;
            }
            else {
                        return 2;
            }
}
```

Sample generated HTML

Any of these:

- `<BR>The print queue is empty.<BR>`
- `<BR>The print queue is full.<BR>`
- `<BR>There are 6 jobs in the print queue.<BR>`

Repeat groups

HTML comment tags

```
<!-- RpRepeatGroup RpStart=s RpLimit=l RpIncrement=inc RpGroupPtr=g -->

<!-- RpRepeatGroupDynamic RpFunctionPtr=fp RpGroupPtr=gp RpMaxItems=m -->

<!-- RpRepeatGroupWhile RpFunctionPtr=fp RpGroupPtr=gp RpMaxItems=m -->
```

This table describes the tags:

Tag	Description
`RpRepeatGroup`	Defines a repeating group of items where the parameters to define the repeat loop are known at compile time. This tag provides the equivalent of a “for” loop.
`RpRepeatGroupDynamic`	Defines a repeating group of items where the parameters to define the repeat loop are determined at run time at the beginning of the loop. This item provides the equivalent of a “for” loop with dynamic initialization.

Tag	Description
RpRepeatGroupWhile	Defines a repeating group of items in which the parameters to define the repeat loop are determined at run time. This tag provides the equivalent of a “while” loop.
RpLastItemInGroup	Specifies a group of items to repeat

Here are the associated keywords:

Keyword	Description
RpStart, RpLimit, and RpIncrement	Specify the parameters of the “for” loop. If the RpIncrement keyword is not specified, 1 is used.
RpFunctionPtr	Specifies a function that returns the start, limit, and increment values for the repeat group.
RpFunctionPtr	Specifies a function that indicates when the looping is to terminate. The function is passed a pointer to a location it can use as a loop index. The function also is passed a pointer to an arbitrary value that the device routine can use. Initially, the index value is passed as 0, and the arbitrary value is passed to the device routine as a (void *) 0. On later calls, the index and the arbitrary value passed to the device routine are whatever the device passed back on the previous call. When the device routine returns a value of a (void *) 0, it signals to the RomPager engine that the loop is complete.

The group of items to repeat can be specified by items that follow the repeat group tag, with the RpLastItemInGroup tag ending the group. If the RpGroupPtr keyword is present, it specifies the identifier of a previously defined group of repeat items. If the structure access form of variable retrieval is used, the RpMaxItems keyword is used to tell the PBuilder compiler how much room to reserve in the structures it creates.

Function prototypes

RpRepeatGroupDynamic

```
typedef void (*rpDynamicRepeatFuncPtr)(void *theServerDataPtr,
                                       Signed16Ptr theStart,
                                       Signed16Ptr theLimit,
                                       Signed16Ptr theIncrement);
```

```
RpRepeatGroupWhile

typedef void (*rpRepeatWhileFuncPtr)(void *theServerDataPtr,
                                     Signed16Ptr theIndexPtr,
                                     void **
theRepeatGroupValuePtr);
```

Commented HTML

```
<!-- RpDataZeroTerminated -->
          <TABLE BORDER="0" CELLPADDING = "0" CELLSPACING= "0"
BGCOLOR=#000000>
          <TR>
<!-- RpEnd -->
<!-- RpRepeatGroup RpStart=1 RpLimit=8 RpIncrement=1 -->
          <!-- RpDataZeroTerminated -->
                    <TD><IMG SRC="/Images/10BT_Port"></TD>
          <!-- RpEnd -->
<!-- RpLastItemInGroup -->
<!-- RpDataZeroTerminated -->
          </TR></TABLE>
<!-- RpEnd -->
```

Internal structures

```
/* Element List Items */
          { eRpItemType_DataZeroTerminated, &PgSample_Item_1 },
          { eRpItemType_RepeatGroup, &PgSample_Item_2 },
          { eRpItemType_DataZeroTerminated,&PgSample_Item_3 },

/* Structures */

static char PgSample_Item_1[] =
          C_oTABLE_BORDER "\"0\"" C_CELLPADDING " \"0\"" C_CELLSPACING "
\"0\" "
          "BGCOLOR=#000000>\n"
          C_oTR "\n";

static rpRepeatGroupItem PgSample_Item_2 = {
          1,
          8,
```

```
            1,
            PgSample_Item_2_Group
};

static char PgSample_Item_3[] =
            C_xTR
            C_xTABLE;

static rpItem PgSample_Item_2_Group[] = {
            { eRpItemType_DataZeroTerminated, &PgSample_Item_4 },
            { eRpItemType_LastItemInList }
};

static char PgSample_Item_4[] =
            C_oTD C_oIMG_SRC "/Images/10BT_Port\">"
            C_xTD;
```

Commented HTML

```
<!-- RpDataZeroTerminated -->
            <TABLE BORDER="0" CELLPADDING = "0" CELLSPACING= "0"
BGCOLOR=#000000>
            <TR>
<!-- RpEnd -->
<!-- RpRepeatGroupDynamic RpFunctionPtr=GetRepeatGroupLimits -->
            <!-- RpDataZeroTerminated -->
                              <TD><IMG SRC="/Images/10BT_Port"></TD>
            <!-- RpEnd -->
<!-- RpLastItemInGroup -->
<!-- RpDataZeroTerminated --></TR></TABLE><!-- RpEnd -->
```

Internal structures

```
/* Element List Items */
            { eRpItemType_DataZeroTerminated, &PgSample_Item_1 },
            { eRpItemType_RepeatGroupDynamic, &PgSample_Item_2 },
            { eRpItemType_DataZeroTerminated,&PgSample_Item_3 },

/* Structures */
```

```
static char PgSample_Item_1[] =
          C_oTABLE_BORDER "\"0\"" C_CELLPADDING " \"0\"" C_CELLSPACING "
\"0\" "
          "BGCOLOR=#000000>\n"
          C_oTR "\n";

static rpRepeatGroupDynItem PgSample_Item_2 = {
          GetRepeatGroupLimits,
          PgSample_Item_2_Group
};

static char PgSample_Item_3[] =
          C_xTR
          C_xTABLE;

static rpItem PgSample_Item_2_Group[] = {
          { eRpItemType_DataZeroTerminated, &PgSample_Item_4 },
          { eRpItemType_LastItemInList }
};

static char PgSample_Item_4[] =
          C_oTD C_oIMG_SRC "/Images/10BT_Port\">"
          C_xTD;

/* Variables and Functions */

void GetRepeatGroupLimits(void *theServerDataPtr, Signed16Ptr theStart,
                          Signed16Ptr theLimit, Signed16Ptr
theIncrement) {

          *theStart = 1;
          *theLimit = 8;
          *theIncrement = 1;
}
```

Commented HTML

```
<!-- RpDataZeroTerminated -->
          <TABLE BORDER="0" CELLPADDING = "0" CELLSPACING= "0"
BGCOLOR=#000000>
          <TR>
<!-- RpEnd -->
<!-- RpRepeatGroupWhile RpFunctionPtr=TestPortRepeatGroup -->
          <!-- RpDataZeroTerminated -->
                         <TD><IMG SRC="/Images/10BT_Port"></TD>
          <!-- RpEnd -->
<!-- RpLastItemInGroup -->
<!-- RpDataZeroTerminated -->
          </TR></TABLE>
<!-- RpEnd -->
```

Internal structures

```
/* Element List Items */
          { eRpItemType_DataZeroTerminated, &PgSample_Item_1 },
          { eRpItemType_RepeatGroupWhile, &PgSample_Item_2 },
          { eRpItemType_DataZeroTerminated,&PgSample_Item_3 },

/* Structures */

static char PgSample_Item_1[] =
          C_oTABLE_BORDER "\"0\"" C_CELLPADDING " \"0\"" C_CELLSPACING "
\"0\" "
          "BGCOLOR=#000000>\n"
          C_oTR "\n";

static rpRepeatGroupDynItem PgSample_Item_2 = {
          TestPortRepeatGroup,
          PgSample_Item_2_Group
};

static char PgSample_Item_3[] =
          C_xTR
          C_xTABLE;

static rpItem PgSample_Item_2_Group[] = {
```

```
            { eRpItemType_DataZeroTerminated, &PgSample_Item_4 },
            { eRpItemType_LastItemInList }
};

static char PgSample_Item_4[] =
            C_oTD C_oIMG_SRC "/Images/10BT_Port\">"
            C_xTD;

/* Variables and Functions */

#define kPortMax = 8

void TestPortRepeatGroup(void *theServerDataPtr,
                                        Signed16Ptr theIndexPtr,
                                        void **theRepeatGroupValuePtr) {
            Signed16          theIndex;

            theIndex = *theIndexPtr;
            if (theIndex < kPortMax) {
                        *theRepeatGroupValuePtr =
thePortName[theIndex];
            }
            else {
                        *theRepeatGroupValuePtr = (void *) 0;
            }
            theIndex += 1;
            *theIndexPtr = theIndex;
}
```

Sample generated HTML

```
<TABLE BORDER="0" CELLPADDING = "0" CELLSPACING= "0" BGCOLOR=#000000><TR>
<TD><IMG SRC="/Images/10BT_Port"></TD>
<TD><IMG SRC="/Images/10BT_Port"></TD>
<TD><IMG SRC="/Images/10BT_Port"></TD>
<TD><IMG SRC="/Images/10BT_Port"></TD>
<TD><IMG SRC="/Images/10BT_Port"></TD>
<TD><IMG SRC="/Images/10BT_Port"></TD>
<TD><IMG SRC="/Images/10BT_Port"></TD>
<TD><IMG SRC="/Images/10BT_Port"></TD>
</TR></TABLE>
```

Repeat groups — Index display items

Within repeat groups, it is sometimes useful to display HTML that uses the current index settings of the repeat group.

HTML comment tags

```
<!-- RpIndexDisplay_0 RpText=t -->
<!-- RpIndexDisplay_1 RpText=t -->
<!-- RpIndexDisplay_2 RpText=t -->
<!-- RpIndexDisplay_3 RpText=t -->
<!-- RpIndexDisplay_4 RpText=t -->
<!-- RpIndexDisplay_5 RpText=t -->
<!-- RpQueryDisplay RpText=t -->
```

Use these tags within repeat groups to display the current value of an index:

Tag	Description
`RpIndexDisplay_0`	References the current index for the current repeat group. When you use nested repeat groups, `RpIndexDisplay_1` references the current index for the repeat group that contains the current repeat group. If you have a three-level repeat group (x, y, z), and you are in the innermost group, `RpIndexDisplay_0` refers to the z value, `RpIndexDisplay_1` refers to the y value and `RpIndexDisplay_2` refers to the x value. The string value of the appropriate index is appended to the text that is defined by the `RpText` keyword. Index values are generated automatically within repeat group, and you can pass them to other pages using query strings. When the RomPager engine receives a URL with an appended query string of the form `?x,y,z`, it stores the incoming values in the current index array.
`RpQueryDisplay`	Used within repeat groups to display a query string that includes the current value of all indices. The query string is appended to the text that is defined by the `RpText` keyword.

Commented HTML

```
<!-- RpDataZeroTerminated -->
          <TABLE BORDER="1" CELLPADDING = "10" CELLSPACING= "0"><TR>
<!-- RpEnd -->
<!-- RpIndexDisplay_0 RpText="<TD><B>Slot " -->
          <TD><B>Slot 3
<!-- RpEnd -->
```

```
<!-- RpDataZeroTerminated -->
          </B></TD>
<!-- RpEnd -->
<!-- RpRepeatGroup RpStart=1 RpLimit=8 RpIncrement=1 -->
          <!-- RpQueryDisplay RpText="<TD><FONT SIZE=+2><A HREF=\"/
PortInfo" -->
                         <TD><FONT SIZE=+2><A HREF="/PortInfo?3,1
          <!-- RpEnd -->
          <!-- RpIndexDisplay_0 RpText="\">" -->
                         ">1
          <!-- RpEnd -->
          <!-- RpDataZeroTerminated -->
                         </A></FONT></TD>
          <!-- RpEnd -->
<!-- RpLastItemInGroup -->
<!-- RpDataZeroTerminated -->
          </TR></TABLE>
<!-- RpEnd -->
```

Internal structures

```
/* Element List Items */
          { eRpItemType_DataZeroTerminated, &PgSample_Item_1 },
          { eRpItemType_IndexDisplay_0, &PgSample_Item_2 },
          { eRpItemType_DataZeroTerminated, &PgSample_Item_3 },
          { eRpItemType_RepeatGroup, &PgSample_Item_4 },
          { eRpItemType_DataZeroTerminated,&PgSample_Item_5 },

/* Structures */

static char PgSample_Item_1[] =
          C_oTABLE_BORDER "\"1\"" C_CELLPADDING " \"10\"" C_CELLSPACING "
\"0\">"
          C_oTR "\n";

static char PgSample_Item_2[] =
          C_oTD C_oB "Slot ";

static char PgSample_Item_3[] =
          C_xB
```

```
            C_xTD;

static rpRepeatGroupItem PgSample_Item_4 = {
            1,
            8,
            1,
            PgSample_Item_4_Group
};
static char PgSample_Item_5[] =
            C_xTR
            C_xTABLE;

static rpItem PgSample_Item_4_Group[] = {
            { eRpItemType_QueryDisplay, &PgSample_Item_6 },
            { eRpItemType_IndexDisplay_0, &PgSample_Item_7 },
            { eRpItemType_DataZeroTerminated,&PgSample_Item_8 },
            { eRpItemType_LastItemInList }
};

static char PgSample_Item_6[] =
            C_oTD C_oFONT_SIZE "+2>" C_oANCHOR_HREF "/PortInfo";

static char PgSample_Item_7[] =
            "\">";

static char PgSample_Item_8[] =
            C_xANCHOR C_xFONT C_xTD;
```

Sample generated HTML

```
<TABLE BORDER="1" CELLPADDING = "10" CELLSPACING= "0"><TR>
<TD><B>Slot 3</B></TD>
<TD><FONT SIZE=+2><A HREF="/PortInfo?3,1">1</FONT></TD>
<TD><FONT SIZE=+2><A HREF="/PortInfo?3,2">2</FONT></TD>
<TD><FONT SIZE=+2><A HREF="/PortInfo?3,3">3</FONT></TD>
<TD><FONT SIZE=+2><A HREF="/PortInfo?3,4">4</FONT></TD>
<TD><FONT SIZE=+2><A HREF="/PortInfo?3,5">5</FONT></TD>
```

```
<TD><FONT SIZE=+2><A HREF="/PortInfo?3,6">6</FONT></TD>
<TD><FONT SIZE=+2><A HREF="/PortInfo?3,7">7</FONT></TD>
<TD><FONT SIZE=+2><A HREF="/PortInfo?3,8">8</FONT></TD>
</TR></TABLE>
```

Repeat groups — dynamic form items

Within a repeat group, it is often useful to use common internal routines for processing array values for form items such as the `<INPUT TYPE=xxxx>` items and `<SELECT>` items.

HTML comment tags

```
<!-- RpFormInput TYPE=t NAME=n VALUE=v RpGetPtr=gp RpSetPtr=sp
RpTextType=tt
          MaxLength=m Size=s RpItemNumber=in Dynamic RpResetPtr=rp -->

<!-- RpFormSingleSelect NAME=n Size=s RpGetPtr=gp RpSetPtr=sp Dynamic -->

<!-- RpFormMultiSelect NAME=n Size=s Dynamic RpResetPtr=rp -->

<!-- RpFormVariableSelect NAME=n MULTIPLE RpResetPtr=r
          RpGetPtr=gp RpSetPtr=sp Size=s RpTextType=t RpFieldWidth=f
Dynamic -->

<!-- RpFormVarValueSelect NAME=n MULTIPLE RpResetPtr=r
          RpGetPtr=gp RpSetPtr=sp Size=s RpTextType=t RpFieldWidth=f
Dynamic -->
```

If the `Dynamic` keyword is used with any of these tags, the PBuilder compiler generates different internal structures so that the array values can be accessed by common routines. The access functions are assumed to be of type `Complex`, allowing the functions to access the index values of the repeat group - so you do not need to specify the `RpGetType` and `RpSetType` keywords.

Function prototypes

The function prototypes for the repeat group versions of the form elements are the same as for the regular complex type form elements.

In the case of the check box and the multi-select items, an additional function is defined using the `RpResetPtr` keyword. This function is used to reset the entire array before the repeat group processing.

```
<!-- RpFormInput TYPE=checkbox -->
<!-- RpFormMultiSelect -->

typedef void (*rpResetCheckboxArrayPtr) (void *theServerDataPtr,
                                         char *theNamePtr,
                                         Signed16Ptr theIndexValuesPtr);
```

Commented HTML

```
<!-- RpDataZeroTerminated -->
          <TABLE BORDER="1" CELLPADDING = "5" CELLSPACING= "0">
          <TR ALIGN=CENTER><TD COLSPAN=3>
<!-- RpEnd -->
<!-- RpIndexDisplay_0 text="<B>Slot " -->
          <B>Slot 3
<!-- RpEnd -->
<!-- RpDataZeroTerminated -->
          </B></TD></TR>
          <TR ALIGN=CENTER><TD>Port Number</TD>
          <TD>Port Name</TD><TD>Active</TD></TR>
<!-- RpEnd -->
<!-- RpRepeatGroup RpStart=1 RpLimit=8 RpIncrement=1 -->
          <!-- RpIndexDisplay_0 text="<TR ALIGN=CENTER><TD>" -->
                    <TR ALIGN=CENTER><TD>1
          <!-- RpEnd -->
          <!-- RpDataZeroTerminated RpIdentifier=PgCellSeparator -->
                    </TD><TD>
          <!-- RpEnd -->
          <!-- RpformInput TYPE=text NAME=PortName Size=20
             RpGetPtr=GetPortName RpSetPtr=SetPortName Dynamic -->
             <INPUT TYPE="Text" NAME="PortName" Value="Port_1" Size=20>
          <!-- RpEnd -->
          <!-- RpUseId RpId=PgCellSeparator RpItemType=DZT -->
                    </TD><TD>
          <!-- RpEnd -->
          <!-- RpFormInput TYPE=checkbox NAME=PortActive
```

```
                                RpGetPtr=GetPortStatus RpSetPtr=SetPortStatus
                                Dynamic RpResetPtr=ResetPortStatus -->
                                <INPUT TYPE="CHECKBOX" NAME="PortActive">
          <!-- RpEnd -->
          <!-- RpDataZeroTerminated -->
                                </TD></TR>
          <!-- RpEnd -->
<!-- RpLastItemInGroup -->
<!-- RpDataZeroTerminated -->
          </TABLE>
<!-- RpEnd -->
```

Internal structures

```
/* Element List Items */
          { eRpItemType_DataZeroTerminated, &PgSample_Item_1 },
          { eRpItemType_IndexDisplay_0, &PgSample_Item_2 },
          { eRpItemType_DataZeroTerminated, &PgSample_Item_3 },
          { eRpItemType_RepeatGroup, &PgSample_Item_4 },
          { eRpItemType_DataZeroTerminated,&PgSample_Item_5 },

/* Structures */

static char PgSample_Item_1[] =
          C_oTABLE_BORDER "\"1\"" C_CELLPADDING " \"5\"" C_CELLSPACING "
\"0\">\n"
          C_oTR_ALIGN_CENTER ">" C_oTD_COLSPAN "3>\n";

static char PgSample_Item_2[] =
          C_oB "Slot ";

static char PgSample_Item_3[] =
          C_xB C_xTD C_xTR C_oTR_ALIGN_CENTER ">" C_oTD "Port Number"
C_xTD
          C_oTD "Port Name" C_xTD C_oTD "Active" C_xTD C_xTR;

static rpRepeatGroupItem PgSample_Item_4 = {
          1,
          8,
          1,
```

```
            PgSample_Item_4_Group
};

static char PgSample_Item_5[] =
            C_xTABLE;

static rpItem PgSample_Item_4_Group[] = {
            { eRpItemType_IndexDisplay_0, &PgSample_Item_6 },
            { eRpItemType_DataZeroTerminated,&PgCellSeparator },
            { eRpItemType_FormTextDyn, &PgSample_Item_7 },
            { eRpItemType_DataZeroTerminated,&PgCellSeparator },
            { eRpItemType_FormCheckboxDyn,&PgSample_Item_8 },
            { eRpItemType_DataZeroTerminated,&PgSample_Item_9 },
            { eRpItemType_LastItemInList }
};

static char PgSample_Item_6[] =
            C_oTR_ALIGN_CENTER ">\n"  C_oTD;

char PgCellSeparator[] =
            C_xTD C_oTD;

static rpTextFormItem PgSample_Item_7 = {
            "PortName",
            GetPortName,
            SetPortName,
            eRpVarType_Complex,
            eRpVarType_Complex,
            eRpTextType_ASCII,
            20,
            20
};

static char PgSample_Item_8[] =
            C_xANCHOR C_xFONT C_xTD;
            C_oBR;

static rpDynCheckboxFormItem PgSample_Item_8 = {
            "PortActive",
```

```
            GetPortStatus,
            SetPortStatus,
            eRpVarType_Complex,
            eRpVarType_Complex,
            ResetPortStatus
};

static char PgSample_Item_9[] =
            C_xTD C_xTR;
/* Variables and Functions */

Boolean thePortStatusArray[4][8];
char thePortNameArray[4][8][21];

char *GetPortName(void *theServerDataPtr,
                                        char *theNamePtr,
                                        Signed16Ptr theIndexValuesPtr) {
            Signed16          theSlotIndex = *theIndexValuesPtr;
            Signed16          thePortIndex = *(theIndexValuesPtr + 1);

            return thePortNameArray[theSlotIndex][thePortIndex];
}

extern void SetPortName(void *theServerDataPtr,
                                        char *theValuePtr,
                                        char *theNamePtr,
                                        Signed16Ptr theIndexValuesPtr) {
            Signed16          theSlotIndex = *theIndexValuesPtr;
            Signed16          thePortIndex = *(theIndexValuesPtr + 1);

            strcpy (thePortStatusArray[theSlotIndex][thePortIndex],
theValuePtr);
}

Boolean GetPortStatus(void *theServerDataPtr,
                                        char *theNamePtr,
                                        Signed16Ptr theIndexValuesPtr) {
            Signed16          theSlotIndex = *theIndexValuesPtr;
            Signed16          thePortIndex = *(theIndexValuesPtr + 1);
```

```
            return thePortStatusArray[theSlotIndex][thePortIndex];
}
void SetPortStatus(void *theServerDataPtr,
                                        Boolean theValue,
                                        char *theNamePtr,
                                        Signed16Ptr theIndexValuesPtr) {
            Signed16          theSlotIndex = *theIndexValuesPtr;
            Signed16          thePortIndex = *(theIndexValuesPtr + 1);

            thePortStatusArray[theSlotIndex][thePortIndex] = theValue;
}
void ResetPortStatus(void *theServerDataPtr,
                                        char *theNamePtr,
                                        Signed16Ptr theIndexValuesPtr) {
            Signed16          theSlotIndex;
            Signed16          thePortIndex;

            for (theSlotIndex = 0; theSlotIndex < 4; theSlotIndex += 1) {
                              for (thePortIndex = 0; thePortIndex < 8;
thePortIndex += 1) {

thePortStatusArray[theSlotIndex][thePortIndex] = False;
                              }
            }
}
```

Sample generated HTML

```
<TABLE BORDER="1" CELLPADDING = "5" CELLSPACING= "0">
<TR ALIGN=CENTER><TD COLSPAN=3><B>Slot 3</B></TD></TR>
<TR ALIGN=CENTER><TD>Port Number</TD><TD>Port Name</TD><TD>Active</TD></
TR>
<TR ALIGN=CENTER><TD>1</TD>
<TD><INPUT TYPE="Text" NAME="PortName?3,1" Value="Port_1" Size=20></TD>
<TD><INPUT TYPE="CHECKBOX" NAME="PortActive?3,1"></TD></TR>
<TR ALIGN=CENTER><TD>2</TD>
<TD><INPUT TYPE="Text" NAME="PortName?3,2" Value="Accounting" Size=20></
TD>
<TD><INPUT TYPE="CHECKBOX" NAME="PortActive?3,2" CHECKED></TD></TR>
<TR ALIGN=CENTER><TD>3</TD>
```

```
<TD><INPUT TYPE="Text" NAME="PortName?3,3" Value="Port_3" Size=20></TD>
<TD><INPUT TYPE="CHECKBOX" NAME="PortActive?3,3"></TD></TR>
<TR ALIGN=CENTER><TD>4</TD>
<TD><INPUT TYPE="Text" NAME="PortName?3,4" Value="Port_4" Size=20></TD>
<TD><INPUT TYPE="CHECKBOX" NAME="PortActive?3,4"></TD></TR>
<TR ALIGN=CENTER><TD>5</TD>
<TD><INPUT TYPE="Text" NAME="PortName?3,5" Value="Marketing" Size=20></TD>
<TD><INPUT TYPE="CHECKBOX" NAME="PortActive?3,5" CHECKED></TD></TR>
<TR ALIGN=CENTER><TD>6</TD>
<TD><INPUT TYPE="Text" NAME="PortName?3,6" Value="Port_6" Size=20></TD>
<TD><INPUT TYPE="CHECKBOX" NAME="PortActive?3,6"></TD></TR>
<TR ALIGN=CENTER><TD>7</TD>
<TD><INPUT TYPE="Text" NAME="PortName?3,7" Value="Port_7" Size=20></TD>
<TD><INPUT TYPE="CHECKBOX" NAME="PortActive?3,7" CHECKED></TD></TR>
<TR ALIGN=CENTER><TD>8</TD>
<TD><INPUT TYPE="Text" NAME="PortName?3,8" Value="Port_8" Size=20></TD>
<TD><INPUT TYPE="CHECKBOX" NAME="PortActive?3,8"></TD></TR>
</TABLE>
```

Repeat groups — radio buttons

Within a repeat group, you might want to have a radio button group with one button for each member of the group. This capability can be useful for selecting an item within a repeat group.

HTML comment tag

```
<!-- RpFormRadioGroupDyn NAME=n RpGetType=gt RpGetPtr=gp
          RpSetType=st RpSetPtr=sp -->
```

The `RpFormRadioGroupDyn` tag specifies a radio button group that spans the range of the repeat group with one button associated with each index value of the repeat group. The `NAME` keyword specifies the HTML name of the radio button group.

When a page is displayed, the RomPager engine can use the access functions described next to retrieve a value. If the value matches the current index of the repeat group, the button is highlighted. When a form is submitted, the RomPager engine passes the value of the submitted button (the repeat group index) to the storage functions.

Function prototypes

```
GetType = Function

typedef rpOneOfSeveral (*rpFetchRadioGroupFuncPtr)(void);

GetType = Complex

typedef rpOneOfSeveral (*rpFetchRadioGroupComplexPtr)(void
*theServerDataPtr,
                                    char *theNamePtr, Signed16Ptr
theIndexValuesPtr);

SetType = Function

typedef void (*rpStoreRadioGroupFuncPtr)(Unsigned8 theValue);

SetType = Complex

typedef void (*rpStoreRadioGroupComplexPtr)(void *theServerDataPtr,
                                    Unsigned8 theValue, char *theNamePtr,
                                    Signed16Ptr theIndexValuesPtr);
```

Commented HTML

```
<!-- RpDataZeroTerminated -->
          <TABLE BORDER="1" CELLPADDING = "5" CELLSPACING= "0">
          <TR ALIGN=CENTER><TD COLSPAN=2>Select Port to Configure</TD></
TR>
          <TR ALIGN=CENTER><TD>Port Number</TD>
          <TD>Port Name</TD></TR>
<!-- RpEnd -->
<!-- RpRepeatGroup RpStart=1 RpLimit=8 RpIncrement=1 -->
          <!-- RpDataZeroTerminated -->
                        <TR ALIGN=CENTER><TD>
          <!-- RpEnd -->
          <!-- RpFormRadioGroupDyn NAME=SelPort
                        RpGetPtr=SelectedPort RpSetPtr=SelectedPort --
>
          <!-- RpIndexDisplay_0 text=" "-->
                         
```

```
            <!-- RpEnd -->

            <!-- RpDataZeroTerminated -->
                        </TD><TD>
            <!-- RpEnd -->
            <!-- RpDisplayText RpGetType=Complex RpGetPtr=GetPortName -->
                        Port_1
            <!-- RpEnd -->
            <!-- RpDataZeroTerminated -->
                        </TD></TR>
            <!-- RpEnd -->
<!-- RpLastItemInGroup -->
<!-- RpDataZeroTerminated -->
            <TR ALIGN=CENTER><TD COLSPAN=2>
<!-- RpEnd -->
            <INPUT TYPE="Submit" VALUE="Configure">
<!-- RpDataZeroTerminated -->
            </TD></TR>
            </TABLE>
<!-- RpEnd -->
```

Internal structures

```
/* Element List Items */
            { eRpItemType_DataZeroTerminated, &PgSample_Item_1 },
            { eRpItemType_RepeatGroup, &PgSample_Item_2 },
            { eRpItemType_DataZeroTerminated, &PgSample_Item_3 },
            { eRpItemType_FormSubmit, &PgSample_Item_4 },
            { eRpItemType_DataZeroTerminated,&PgSample_Item_5 },

/* Structures */

static char PgSample_Item_1[] =
            C_oTABLE_BORDER "\"1\"" C_CELLPADDING " \"5\"" C_CELLSPACING "
\"0\">\n"
            C_oTR_ALIGN_CENTER ">" C_oTD_COLSPAN "2>Select Port to
Configure"
            C_xTD C_xTR C_oTR_ALIGN_CENTER ">" C_oTD "Port Number" C_xTD
C_oTD
            "Port Name" C_xTD
```

```
            C_xTR;

static rpRepeatGroupItem PgSample_Item_2 = {
            1,
            8,
            1,
            PgSample_Item_2_Group
};

static char PgSample_Item_3[] =
            C_oTR_ALIGN_CENTER ">" C_oTD_COLSPAN "2>\n";

static rpButtonFormItem PgSample_Item_4 = {
            "Configure"
};

static char PgSample_Item_5[] =
            C_xTD C_xTR C_xTABLE;

static rpItem PgSample_Item_2_Group[] = {
            { eRpItemType_DataZeroTerminated,&PgSample_Item_6 },
            { eRpItemType_FormRadioGroupDyn,&PgSample_Item_7 },
            { eRpItemType_IndexDisplay_0, &PgSample_Item_8 },
            { eRpItemType_DataZeroTerminated,&PgSample_Item_9 },
            { eRpItemType_DisplayText, &PgSample_Item_10 },
            { eRpItemType_DataZeroTerminated,&PgSample_Item_11 },
            { eRpItemType_LastItemInList }
};

static char PgSample_Item_6[] =
            C_oTR_ALIGN_CENTER ">" C_oTD;

static rpRadioGroupInfo PgSample_Item_7 = {
            "SelPort",
            &SelectedPort,
            &SelectedPort,
            eRpVarType_Direct,
            eRpVarType_Direct
};
```

```
static char PgSample_Item_8[] =
          C_NBSP;

static char PgSample_Item_9[] =
          C_xTD C_oTD;

static rpTextDisplayItem PgSample_Item_10 = {
          GetPortName,
          eRpVarType_Complex,
          eRpTextType_ASCII,
          20
};
static char PgSample_Item_11[] =
          C_xTD C_xTR;

/* Variables and Functions */

rpOneOfSeveral SelectedPort = 7;

char thePortNameArray[4][8][21];

char *GetPortName(void *theServerDataPtr,
                                    char *theNamePtr,
                                    Signed16Ptr theIndexValuesPtr) {
          Signed16         theSlotIndex = *theIndexValuesPtr;
          Signed16         thePortIndex = *(theIndexValuesPtr + 1);

          return thePortNameArray[theSlotIndex][thePortIndex];
}
```

Sample generated HTML

```
<TABLE BORDER="1" CELLPADDING = "6" CELLSPACING= "0">
<TR ALIGN=CENTER><TD COLSPAN=2>Select Port to Configure</TD></TR>
<TR ALIGN=CENTER><TD>Port Number</TD><TD>Port Name</TD></TR>
<TR ALIGN=CENTER>
<TD><INPUT TYPE="RADIO" NAME="SelPort" VALUE="1"> 1</TD>
<TD>Port_1</TD></TR>
<TR ALIGN=CENTER>
```

```
<TD><INPUT TYPE="RADIO" NAME="SelPort" VALUE="2"> 2</TD>
<TD>Accounting</TD></TR>
<TR ALIGN=CENTER>
<TD><INPUT TYPE="RADIO" NAME="SelPort" VALUE="3"> 3</TD>
<TD>Port_3</TD></TR>
<TR ALIGN=CENTER>
<TD><INPUT TYPE="RADIO" NAME="SelPort" VALUE="4"> 4</TD>
<TD>Port_4</TD></TR>
<TR ALIGN=CENTER>
<TD><INPUT TYPE="RADIO" NAME="SelPort" VALUE="5"> 5</TD>
<TD>Marketing</TD></TR>
<TR ALIGN=CENTER>
<TD><INPUT TYPE="RADIO" NAME="SelPort" VALUE="6"> 6</TD>
<TD>Port_6</TD></TR>
<TR ALIGN=CENTER>
<TD><INPUT TYPE="RADIO" NAME="SelPort" VALUE="7" CHECKED> 7</TD>
<TD>Port_7</TD></TR>
<TR ALIGN=CENTER>
<TD><INPUT TYPE="RADIO" NAME="SelPort" VALUE="8"> 8</TD>
<TD>Port_8</TD></TR>
<TR ALIGN=CENTER>
<TD COLSPAN=2><INPUT TYPE="Submit" VALUE="Configure"></TD></TR>
</TABLE>
```

Page and form objects

The RomPager engine uses object structures to manage ROM-based pages. When a browser requests a URL, the server finds the object in a master object list and determines how to process the request. The PBuilder compiler creates object structures as part of compiling an HTML page:

- A page without a form creates a single object structure.
- A page with a form creates two object structures — one for the page and one for the form.

The optional headers described next are used for overriding the default values that the compiler creates. Certain types of special processing require these tags so that the appropriate flags are set in the object structure.

HTML comment tag

```
<!-- RpPageHeader RpUrl=url RpObjectType=ot
                              RpRefreshTime=rt RpRefreshPage=pgid
RpServerPush RpAccess=aa
                              RpFunctionPtr=fp RpStructuredAccess DebugFlow
Disposition -->

<!-- RpFormHeader Action=url Enctype=et Method=m RpNextPage=pgid
RpAccess=aa
                              RpFunctionPtr=fp RpDirect -->

<! --RpEndForm -->

</form>

<! -RpEnd -->
```

Use `RpEndForm` on all AWS forms to balance the `RpFormHeader` tag. Do not use `RpFormHeader` *only.* If you don't use `RpEndForm`, PBuilder issues an error.

The `RpPageHeader` tag supplies information that is stored in the page object structure. This table describes the keywords associated with `RpPageHeader`:

Keyword	Description
`RpUrl`	Specifies the URL to use to match a request from a browser. If you do not specify the `RpUrl` keyword, the PBuilder compiler creates the URL in the form */xxxxx* where *xxxxx* is the name of the page (*xxxxx*`.html`).
`RpObjectType`	Specifies how the page should be cached. The allowed values are `Static` and `Dynamic`. If you do not specify the keyword, `Dynamic` is used. Dynamic pages are served to the browser with HTTP headers to indicate that the page should not be cached. Static pages are served to the browser with HTTP headers that indicate that the page was created on the ROM image date of the device and can be cached.
`RpRefreshTime`	Specifies that dynamic page techniques should be used on this page and the time to wait before a page is served again. If you specify the `RpRefreshPage` keyword, it identifies the page to be served when the time expires. If you do not specify the `RpRefreshPage` keyword, the current page is redisplayed when the time expires. The `pgid` parameter for the `RpRefreshPage` keyword specifies the internal PBuilder generated name of the next page to be served. This name is of the form `Pg`*xxxxx* where *xxxxx* is the name of the page.

Keyword	Description
RpStructuredAccess	If you define this keyword, the PBuilder compiler creates structured access routines and storage for accessing and storing the variables. For details, see Chapter 5, "Stub routines" and example 2 in the Appendix.
Disposition	If you define this keyword, the compiler sets the kRpObjFlag_Disposition flag for the page object structure. This flag identifies a page object that should be treated as an attachment. If the flag is set, the RomPager engine serves the page with a Content-Disposition: attachment header. Some browsers recognize this header and save the page to the user's disk, rather than interpreting the HTML and displaying the page

The RpFormHeader tag supplies information that is stored in the form object structure. These are the associated keywords:

Keyword	Description
Action	Specifies the URL that is used with the Action keyword of the <FORM> tag. The browser sends this URL when a form is submitted, and the server uses it to find the object that handles the form items. If you do not specify the Action keyword, the PBuilder compiler creates the URL in the form /Forms/*xxxxx* where *xxxxx* is the name of the page (*xxxxx*.html).
Enctype	Specifies the encoding of the form data. The allowed values are APPLICATION/X-WWW-FORM-URLENCODED (the default) and MULTIPART/FORM-DATA.
Method	Specifies the HTTP method for sending the form data. The allowed values are POST (the default) and GET.
RpNextPage	Specifies the page to be served after the form is processed. If the RpNextPage keyword is not specified, the page that the form is on is served again. The pgid parameter for the RpNextPage keyword specifies the internal PBuilder generated name of the next page to be served. This name is of the form Pg*xxxxx* where *xxxxx* is the name of the page. Normally the RomPager engine uses HTTP redirection techniques (using the 302 Moved Temporarily header) to serve the page after a form is processed. The redirection techniques allow browsers to maintain more accurate history lists. The optional RpDirect keyword specifies that the next page is to be served directly and not redirected.

Keyword	Description
RpAccess	Specifies the security realm for both page and form objects. The allowed values are: Unprotected, Realm1, Realm2, … Realm8. You can specify the RpAccess keyword multiple times to indicate the page is allowed in multiple realms. If you do not specify RpAccess, a value of Unprotected is used. If an object belongs to more than one realm, RomPager uses the security method of the the lowest numbered realm in the challenge.
RpFunctionPtr	Specifies a pre-processing function to be called before the page items are displayed or a post-processing function to be called after the form items are processed.

Object structure definitions and prototypes

```
/* Required for Page and Form objects */

typedef struct rpObjectDescription {
          char *                  fURL;
          rpItem *                fItemsArrayPtr;
          rpObjectExtensionPtrfExtensionPtr;
          Unsigned32              fLength;
          rpAccess                fObjectAccess;
          rpDataType              fMimeDataType;
          rpObjectType         fCacheObjectType;
}         rpObjectDescription, *rpObjectDescriptionPtr;

/* Required for Form objects, optional for Page Objects */

typedef struct rpObjectExtension {
          rpProcessDataFuncPtrfProcessDataFuncPtr;
          rpObjectDescription *fPagePtr;
          rpObjectDescription *fRefreshPagePtr;
          Unsigned16              fRefreshSeconds;
          rpObjectFlags        fFlags;
}         rpObjectExtension, *rpObjectExtensionPtr;
```

```
/*         Page object pre-processing, Form object post-processing */

typedef void (*rpProcessDataFuncPtr)(void *theServerDataPtr,
                              Signed16Ptr theIndexValuesPtr);
```

Commented HTML

```
<!-- RpPageHeader RpRefreshTime=17 RpFunctionPtr=SamplePagePreProcessing
           RpAccess=Realm1 RpAccess=Realm2 -->
<!-- RpDataZeroTerminated -->
           <HTML>
           <HEAD>
           <TITLE>Sample Page</TITLE>
           </HEAD>
           <BODY>
<!-- RpEnd -->
<!-- RpFormHeader RpNextPage=PgSample2
RpFunctionPtr=SampleFormPostProcessing
           RpAccess=Realm1 -->
           <FORM METHOD="POST" ACTION="/Forms/Sample">
<!-- RpEnd -->
<!-- RpDataZeroTerminated -->
           <P><CENTER>Empty Page and Form</CENTER></P>
           </FORM>
           </BODY>
           </HTML>
<!-- RpEnd -->
```

Internal structures

```
rpObjectDescription PgSample = {
           "/Sample",
           PgSample_Items,
           &PgSample_ObjectExtension,
           (Unsigned32) 0,
           kRpPageAccess_Realm1 | kRpPageAccess_Realm2,
           eRpDataTypeHtml,
           eRpObjectTypeDynamic
};
static rpObjectExtension PgSample_ObjectExtension = {
```

```
            SamplePagePreProcessing,
            (rpObjectDescriptionPtr) 0,
            (rpObjectDescriptionPtr) 0,
            17,
            kRpObjFlags_None
};
static rpItem PgSample_Items[] = {
            { eRpItemType_DataZeroTerminated,&PgSample_Item_1 },
            { eRpItemType_FormHeader, &PgSample_Form },
            { eRpItemType_DataZeroTerminated,&PgSample_Item_2 },
            { eRpItemType_LastItemInList }
};
rpObjectDescription PgSample_Form = {
            "/Forms/Sample",
            PgSample_FormItems,
            &PgSample_FormObjectExtension,
            (Unsigned32) 0,
            kRpPageAccess_Realm1,
            eRpDataTypeForm,
            eRpObjectTypeDynamic
};

rpObjectExtension PgSample_FormObjectExtension = {
            SampleFormPostProcessing,
            &PgSample2,
            (rpObjectDescriptionPtr) 0,
            0,
            kRpObjFlags_None
};

static rpItem PgSample_FormItems[] = {
            { eRpItemType_LastItemInList }
};

static char PgSample_Item_1[] =
            C_oHTML_oHEAD_oTITLE "Sample Page"
            C_xTITLE_xHEAD_oBODY;

static char PgSample_Item_2[] =
```

```
            C_oP C_oCENTER "Empty Page and Form" C_xCENTER C_xP C_xFORM
            C_xBODY_xHTML;

/* Variables and Functions */

void SamplePagePreProcessing(void *theServerDataPtr,
                                        Signed16Ptr theIndexValuesPtr) {
}
void SampleFormPostProcessing(void *theServerDataPtr,
                                        Signed16Ptr theIndexValuesPtr) {
}
```

Sample generated HTML

```
<HTML>
<HEAD>
<TITLE>Sample Page</TITLE>
</HEAD>
<BODY>
<FORM METHOD="POST" ACTION="/Forms/Sample">
<P><CENTER>Empty Page and Form</CENTER></P>
</FORM></BODY></HTML>
```

Server-side image maps

Image maps support the selection of URL links from an image. With client-side image maps, the coordinate processing is done on the browser, and the browser determines which URL to request next. Because you can specify client-side image maps with standard HTML, you do not need special RomPager tags.

With server-side image maps, when you selects point by clicking the image, the browser sends the coordinates of the point to the server. The server uses an internal map to determine which part of the image has been selected and which URL link to serve. These tags are used to describe the image map and the selection coordinates.

HTML comment tags

```
<!-- RpFormImageMap RpUrl=u RpImgKeywords=ik RpDefaultPage=pgid
          RpSetType=st RpSetPtr=sp -->

<!-- RpImageMapCircle RpItemPage=pgid RpItemNumber=i
          RpCenter=(X,Y) RpRadius=r -->

<!-- RpImageMapRectangle RpItemPage=pgid RpItemNumber=i
          RpCorners=(X1,Y1),(X2,Y2) -->

<!-- RpImageMapPolygon RpItemPage=pgid RpItemNumber=i
          RpPoints=(X1,Y1),(X2,Y2),…, (Xn,Yn) -->

<!-- RpEndImageMapAreas -->
```

The `RpFormImageMap` tag specifies the information used to generate the image map HTML and the processing of the server-side image map.

The `RpUrl` keyword specifies the URL that is the `ACTION` parameter used to process the map. If the `RpUrl` keyword is not specified, the PBuilder compiler creates the URL in the form `/Forms/`*xxxxx* where *xxxxx* is the name of the page (*xxxxx*`.html`). When the RomPager engine builds an image map, it generates an `<IMG ISMAP>` tag. The `RpImgKeywords` keyword specifies a string that contains the keyword attributes to associate with the `<IMG>` tag. The string must contain the URL of the image and can contain other attributes such as `WIDTH` or `HEIGHT`. Individual selection areas within the map are specified by the `RpImageMapCircle`, `RpImageMapRectangle`, and `RpImageMapPolygon` tags.

The `RpDefaultPage` keyword specifies the page to serve if a point is clicked outside the specified selection areas. If the `RpDefaultPage` keyword is not specified, the default page is set to the page that the image map is contained on.

The `Set` function specified by the `RpSetType` and `RpSetPtr` keywords stores a value associated with the selected map area. PBuilder creates a special RomPager form object that is used to process the map.

The image area tags specify the page to serve with the `RpItemPage` keyword and use the `RpItemNumber` keyword to specify the value to be passed to the `Set` function when the area is selected. If the `RpItemNumber` keyword is not specified, PBuilder generates a value for each map area starting at 1.

The RpImageMapCircle tag specifies the information for processing an image map circle. The RpCenter keyword specifies the pixel coordinates of the center of the circle. The RpRadius keyword specifies the radius of the circle.

The RpImageMapRectangle tag specifies the information for processing an image map rectangle. The RpCorners keyword specifies the left, top, right, and bottom coordinates of the rectangle.

The RpImageMapPolygon tag specifies information for processing an image map polygon. The RpPoints keyword specifies the coordinates of the points of the polygon.

The list of image map area tags that are associated with an RpFormImageMap tag is terminated when the RpEndImageMapAreas tag is encountered.

The pgid parameter for the RpDefaultPage and RpItemPage keywords specifies the internal PBuilder-generated name of the page to be served in response to an image map query. This name is of the form Pg*xxxxx* where *xxxxx* is the name of the page.

Function prototypes

```
SetType = Function

typedef void (*rpStoreLocationPtr)(rpMapLocation theValue);

SetType = Complex

typedef void(*rpStoreLocationComplexPtr)(void *theServerDataPtr,
                                         rpMapLocation theValue,
                                         char *theNamePtr,
                                         Signed16Ptr
theIndexValuesPtr);
```

Commented HTML

```
<!-- RpFormImageMap RpDefaultPage=PgSample
          RpSetType=Direct RpSetPtr=ImageMapValue
          RpImgKeywords="SRC=\"/Images/ABImageMap\" WIDTH=109 HEIGHT=55"
-->
<A HREF="/Forms/ImageMapValidation">
<IMG SRC="/Images/ABImageMap" WIDTH=109 HEIGHT=55 ISMAP></A>
```

```
<!-- RpEnd -->
<!-- RpImageMapRectangle RpItemPage=PgSample RpItemNumber=1
          RpCorners=(9, 9),(45,45) -->
<!-- RpImageMapRectangle RpItemPage=PgSample RpItemNumber=2
          RpCorners=(63,9),(96,45) -->
<!-- RpEndImageMapAreas -->
```

Internal structures

```
/* Structures */

rpObjectDescription PgSample = {
          "/Sample",
          PgSample_Items,
          (rpObjectExtensionPtr) 0,
          (Unsigned32) 0,
          kRpPageAccess_Unprotected,
          eRpDataTypeHtml,
          eRpObjectTypeDynamic
};

static rpItem PgSample_Items[] = {
          { eRpItemType_DataZeroTerminated, &PgSample_Item_1 },
          { eRpItemType_LastItemInList }

rpObjectDescription PgSample_Form = {
          "/Forms/Sample",
          PgSample_FormItems,
          &PgSample_FormObjectExtension,
          (Unsigned32) 0,
          kRpPageAccess_Unprotected,
          eRpDataTypeMap,
          eRpObjectTypeDynamic
};

rpObjectExtension PgSample_FormObjectExtension = {
          (rpProcessDataFuncPtr) 0,
          &PgSample,
          (rpObjectDescriptionPtr) 0,
          0,
```

```
            kRpObjFlags_None
};

static rpItem PgSample_FormItems[] = {
            { eRpItemType_FormImageMap, &PgSample_Item_2 },
            { eRpItemType_LastItemInList }
};

static char PgSample_Item_1[] =
            C_oANCHOR_HREF "/Forms/Sample\">" C_oIMG_SRC "/Images/
ABImageMap\""
            C_WIDTH "109" C_HEIGHT "55 ISMAP>"
            C_xANCHOR;
static rpImageMapFormItem PgSample_Item_2 = {
            &PgSample,
            &PgSample_Item_2_Option_1,
            &ImageMapValue,
            eRpVarType_Direct
};
rpImageMapLocation PgSample_Item_2_Option_1 = {
            &PgSample_Item_2_Option_2,
            &PgSample,
            eRpLocationType_Rectangle,
            (rpPointPtr) 0,
            9,
            45,
            9,
            45,
            1
};

rpImageMapLocation PgSample_Item_2_Option_2 = {
            (rpImageMapLocationPtr) 0,
            &PgSample,
            eRpLocationType_Rectangle,
            (rpPointPtr) 0,
            9,
            45,
            63,
```

```
            96,
            2
};

/* Variables and Functions */

rpImageMapLocation ImageMapValue;
```

Sample generated HTML

```
<A HREF="/Forms/Sample"><IMG SRC="/Images/ABImageMap"
                              WIDTH=109 HEIGHT=55 ISMAP></A><BR>
```

HTML referer tag

HTML comment tag

```
<!-- RpHtmlReferer RpText=t -->
```

When a browser sends in a request, it usually sends a HTTP header (referer) that indicates the page from which the request was made.

The `RpHtmlReferer` tag is used to generate a dynamic HTML link to the referring page. PBuilder generates the link using the format `<A HREF="`*`xxxxxx`*`">Link_Text</A>`. The `RpText` keyword defines the text to be used to replace the `Link_Text` portion of the format. The *`xxxxxx`* portion of the format is replaced at runtime by the value of the HTTP referer header.

Commented HTML

```
<!-- RpDataZeroTerminated -->
            <BR>Return to:
<!-- RpEnd -->
<!-- RpHtmlReferer RpText="last page"-->
            <A HREF="http://www.device.com/Sample">last page</A>
<!-- RpEnd -->
<!-- RpDataZeroTerminated --><BR><!-- RpEnd -->
```

Internal structures

```
/* Element List Items */
           { eRpItemType_DataZeroTerminated,&PgSample_Item_1 },
           { eRpItemType_HtmlReferer,&PgSample_Item_2 },
           { eRpItemType_DataZeroTerminated,&PgSample_Item_3 },
/* Structures */

static char PgSample_Item_1[] =
           C_oBR "Return to: "

static char PgSample_Item_2[] =
           "last page";

static char PgSample_Item_3[] =
           C_oBR;
```

Sample generated HTML

```
<BR>Return to: <A HREF="http://www.device.com/Sample">last page</A><BR>
```

URL state tag

HTML comment tag

```
<!-- RpUrlState RpText=t -->
```

The `RpUrlState` tag generates a link with the current URL state. The URL state is appended to the text defined by the `RpText` keyword. For more information, see Chapter 8, “State Management”.

Commented HTML

```
<!-- RpDataZeroTerminated -->
           <BR>Proceed to:
<!-- RpEnd -->
<!-- RpUrlState RpText="<A HREF=\"" -->
           <A HREF="/US/12345”
<!-- RpEnd -->
<!-- RpDataZeroTerminated -->/NextPage">the next page</A>.<BR><!-- RpEnd -
->
```

Internal structures

```
/* Element List Items */
          { eRpItemType_DataZeroTerminated,&PgSample_Item_1 },
          { eRpItemType_UrlState,  &PgSample_Item_2 },
          { eRpItemType_DataZeroTerminated,&PgSample_Item_3 },

/* Structures */

static char PgSample_Item_1[] =
          C_oBR "Proceed to: ";

static char PgSample_Item_2[] =
          C_oANCHOR_HREF;

static char PgSample_Item_3[] =
          "/NextPage\">the next page" C_xANCHOR "."
          C_oBR;
```

Sample generated HTML

```
<BR>Proceed to: <A HREF="/US/12345/NextPage">the next page</A>.<BR>
```

File insert HTML tag

HTML comment tag

```
<!-- RpFile RpFileName=fn -->
```

Use the `RpFile` tag to specify a file whose contents will be dynamically inserted into the HTML stream. This can be useful for creating a page to display contents that are dynamically created elsewhere in the device, such as logging streams.

Commented HTML

```
<!-- RpDataZeroTerminated -->
          <BR>The data log:<BR><PRE>
<!-- RpEnd -->
<!-- RpFile RpFileName="DataLog" -->
          00:00:05  Page Served           /Main
```

```
<!-- RpEnd -->
<!-- RpDataZeroTerminated -->
          </PRE><BR>
<!-- RpEnd -->
```

Internal structures

```
/* Element List Items */
          { eRpItemType_DataZeroTerminated,&PgSample_Item_1 },
          { eRpItemType_File,      &PgSample_Item_2 },
          { eRpItemType_DataZeroTerminated,&PgSample_Item_3 },

/* Structures */

static char PgSample_Item_1[] =
          C_oBR "The data log:" C_oBR "<PRE>";

static char PgSample_Item_2[] =
          "DataLog";

static char PgSample_Item_3[] =
          "</PRE>" C_oBR;
```

Sample generated HTML

```
<BR>The data log:<BR><PRE>
00:02:25  Page Served          /ServerStatus
00:02:03  Page Served          /ServerStatus
00:01:01  Page Served          /ServerStatus
00:00:22  Page Served          /ServerStatus
00:00:14  Page Served          /ServerStatus
00:00:09  Image Served         /Images/RedDot
00:00:09  Page Served          /Menu
00:00:06  Image Served         /Images/Main
00:00:06  Image Served         /Images/BlueDot
00:00:05  Page Served          /Main
</PRE><BR>
```

Image source tag

HTML comment tag

```
<!-- RpImageSource RpObjectPtr=op -->
```

The `RpImageSource` tag creates the beginning portion of the HTML `<IMG>` tag. The `RpObjectPtr` keyword provides the name of a RomPager internal image object created with PBuilder. When the RomPager engine creates the HTML it creates `<IMG SRC="`*`objecturl`*`"` where *`objecturl`* is the URL of the image object.

This tag can provide some compression of an HTML page by eliminating the need to store the URL name of the image multiple times.

Commented HTML

```
<!-- RpImageSource RpObjectPtr=PgBlueDot -->
          <IMG SRC="/Images/BlueDot">
<!-- RpEnd -->
<!-- RpDataZeroTerminated -->
           This is a blue dot.<BR>
<!-- RpEnd -->
```

Internal structures

```
/* Element List Items */
          { eRpItemType_ImageSource,&PgBlueDot },
          { eRpItemType_DataZeroTerminated,&PgSample_Item_2 },

/* Structures */

static char PgSample_Item_2[] =
          C_NBSP "This is a blue dot." C_oBR;
```

Sample generated HTML

```
<IMG SRC="/Images/BlueDot"> This is a blue dot.<BR>
```

Structure sharing tags

The PBuilder compiler uses standard HTML pages with special RomPager comment tags to create the internal C structures for the ROM-based pages. A few tags are used to share structures within a page and across pages.

HTML comment tag

```
<!-- RpUseIdentifier RpIdentifier=i RpItemType=t -->
<!-- RpUseId RpId=i RpItemType=t -->
<!-- RpFormItem RpIdentifier=i RpItemType=t -->
```

When PBuilder compiles an HTML page, it generates a set of C structures and creates default identifiers for the structures based on the name of the page. Because substantial memory savings can be achieved by sharing common structures, it is necessary to specify an identifier for the structures to share them. The `RpIdentifier` keyword is used in the RomPager comment tags so PBuilder uses the supplied identifier instead of creating one.

The `RpUseIdentifier` tag indicates that an item with the given `RpIdentifier` and `RpItemType` is defined elsewhere, but should be used here. `RpUseId` is an alias for `RpUseIdentifier` and `RpId` is an alias for `RpIdentifier`.

Normally, when PBuilder encounters a comment tag that describes a form item, it adds an item to the current form item list and to the current page item list. If a `RpUseIdentifier` tag is used that specifies a group, PBuilder cannot determine the types of items contained within the group. So if the group has form items, PBuilder does not have enough information to add the form item pointers to the current form item list. The `RpFormItem` tag is used to tell PBuilder to add a form item with the given `RpIdentifier` and `RpItemType` to the current form item list.

The `RpItemType` keyword specifies the internal object type used in the element list. The keyword value is formed by removing the leading constant from the RomPager internal item type. So, for example, you specify the `eRpItemType_DisplayText` item type with a keyword value of `DisplayText`.

Here are the possible keyword values:

```
RpItemType = [DataZeroTerminated, DZT, DataLengthEncoded, DisplayText,
DynamicDisplay, ExtendedAscii, File, FormAsciiText, FormButton,
FormCheckbox, FormCheckboxDyn, FormFile, FormFixedMultiDyn,
FormFixedMultiSelect, FormFixedSingleDyn, FormFixedSingleSelect,
FormHeader, FormHiddenText, FormHiddenDyn, FormImageMap, FormNamedSubmit,
FormPasswordText, FormPasswordDyn, FormRadioGroupDyn, FormRadioButton,
FormRadioButtonDyn, FormReset, FormSubmit, FormTextArea, FormTextAreaBuf,
FormTextDyn, FormVariableMultiDyn, FormVariableMultiSelect,
FormVariableSingleDyn, FormVariableSingleSelect, FormVarValueMultiDyn,
FormVarValueMultiSelect, FormVarValueSingleDyn, FormVarValueSingleSelect,
HtmlReferer, ImageSource, IndexDisplay_0, IndexDisplay_1, IndexDisplay_2,
IndexDisplay_3, IndexDisplay_4, IndexDisplay_5, ItemGroup,
NamedDisplayText, QueryDisplay, RepeatGroup, RepeatGroupDynamic,
RepeatGroupWhile, SlaveIdentity, UrlState]
```

Miscellaneous formatting tags

The PBuilder compiler uses standard HTML pages with special RomPager comment tags to create the internal C structures for the ROM-based pages. To ensure that the pages can be viewed with browsers or image editors, you can use some formatting tags.

HTML comment tag

```
<!-- RpHiddenDataZeroTerminated RpText=h -->
<!-- RpHDZT RpText=h -->
<!-- RpSample -->
```

This table describes the tags:

Tag	Description
`RpHiddenDataZeroTerminated`	Used for HTML text that is to be included in the internal C structures, but that you want the browser to ignore when it displays the sample page. Usually this tag is used with the `RpDynamicDisplay` tag, when multiple choices could be displayed, but only one is shown on the sample page. `RpHDZT` is an alias for `RpHiddenDataZeroTerminated`.

Tag	Description
`RpText` keyword	Supplies the HTML to process.
`RpSample`	Used when a browser or page editor should show some HTML during the page design process that PBuilder is to ignore because other tags will dynamically create HTML. The sample HTML follows this tag and is terminated by the `<!-- RpEnd -->` tag.

The `RpText` keyword supplies the HTML to process.

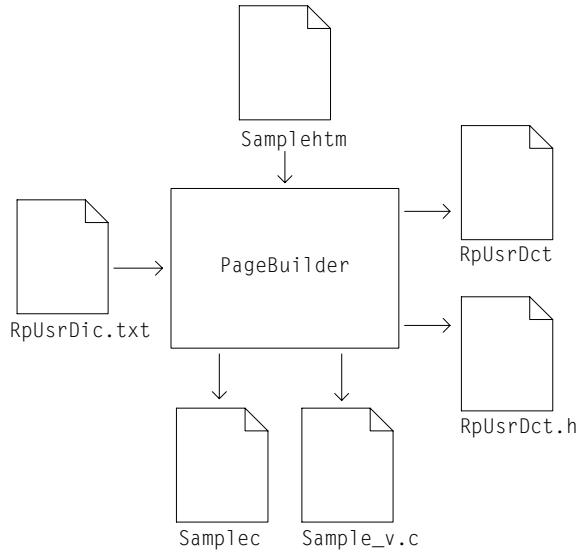
Phrase Dictionaries

CHAPTER 3

This chapter describes the PBuilder's two phrase dictionaries.

Overview

The RomPager engine uses a phrase dictionary technique to provide compression for static ASCII text strings. The PBuilder compiler searches the HTML input files for strings that match the phrase dictionaries and replaces the strings with tokens as it stores the static text elements.



Compression

The PBuilder compiler recognizes standard HTML phrases and stores them as single-byte tokens in the compressed text. In addition, the PBuilder recognizes user-specified phrases and stores them as two-byte tokens in the compressed text. For example, the input phrase:

```
<TITLE>The AS Group Ltd. C3PO Control Page</TITLE>
```

would be stored internally as:

```
static char PgTitle[] = C_oTITLE “The AS Group Ltd. C3PO Control Page”
C_xTITLE;
```

The RpUsrDct.txt file specifies the user-defined strings that the compiler should look for when compiling HTML pages. Each entry is of the form:

```
MacroName = “Search Phrase”
```

which tells PBuilder to search for phrases that match the search phrase, and replace them with the specified macro definition that represents a two-byte token.

If you want user dictionary definitions to contain system dictionary definitions, specify the system dictionary macro name in the user dictionary definition. For example, in the provided RpUsrDct.txt, this definition:

```
C_S_AllegroLogo = C_oHR C_oP C_oCENTER C_oIMG_SRC “/Images/Main\”“ C_WIDTH
“162”
C_HEIGHT “83>“ C_xCENTER
```

tells the PBuilder compiler to look for all phrases of the form

```
<HR><P><CENTER><IMG SRC=/Images/Main WIDTH=162 HEIGHT=83></CENTER>
```

and replace them with the two-byte token represented with the name C_S_AllegroLogo.

If company name and device phrases were defined, this input phrase:

```
<TITLE>The AS Group Ltd. C3PO Control Page</TITLE>
```

would be stored internally as:

```
static char PgTitle[] = C_oTITLE “The “ C_S_Company “ “ C_S_Device
 “ Control Page” C_xTITLE;
```

The compiler generates two C source files — RpUsrDct.c and RpUsrDct.h — for the user dictionary, using the RpUsrDct.txt file as input. The individual entries in the dictionary are created in the order in which they are specified in RpUsrDct.txt. The definitions of the user dictionary are in RpUsrDct.h with the data entries in RpUsrDct.c. You must compile and link these files with your project along with the RomPager engine and the page files.

The RpUsrDct.txt file provided with RomPager contains definitions for two sets of entries. The first 64 definitions are for defining error message strings that the RomPager engine uses. These entries are reserved. The definitions that follow the error message definitions are used in the demo pages that are provided with the RomPager development environment and can be replaced in the production environment.

International support with dynamic user dictionaries

You can dynamically select user dictionaries at runtime using the `RpSetUserPhraseDictionary` callback routine. One use of this feature is to support international HTML pages. By defining all the phrases on an HTML page that are to be internationalized as user dictionary phrases, you can change the display language of the page by selecting a different user dictionary while maintaining the graphical layout and access to device variables.

For an application with six languages, the HTML page structure would be stored only once, with a different user-phrase dictionary for each language. Because the `RpSetUserPhraseDictionary` specifies a pointer to the table of string pointers at run-time, the user-phrase dictionaries can be stored in either ROM or a file system, depending on product needs.

Typically, the development process is to define the HTML pages and phrases to be internationalized in a single language. For example, the English phrases to be translated are defined as user dictionary phrases in the `RpUsrDct.txt` file, and the HTML that defines the page layout and variable access using the English phrases is defined in `Sample.htm`.

After you run the PBuilder compiler, four files are created:

File	Description
`Sample.c`	Contains the compressed and tokenized HTML
`Sample_v.c`	Contains the stub variable access routines
`RpUsrDct.h`	Contains the index to the user dictionary
`RpUsrDct.c`	Contains the English version of the phrases (without the HTML)

The files can then be given to translation teams to create alternate language versions of the phrases, and the translation teams do not need to be concerned with HTML.

Expansion

As the static text elements of type `eRpItemType_DataZeroTerminated` are served, they are examined for characters with the high-bit set, and replacement phrases are substituted. Standard ASCII characters are in the range `<000>` to `<177>` and pass through the dictionary expansion process unchanged. For international pages that use the high-bit characters (such as the SJIS form of Japanese), the international extended characters are stored by PBuilder using the `eRpItemType_ExtendedAscii` element type, and are not passed through the dictionary expansion process.

The PBuilder has two phrase dictionaries:

- **System dictionary**. Uses single byte tokens
- **User-specified dictionary**. Uses double byte tokens

The Extended ASCII characters `<200>` to `<372>` are used by the built-in system dictionary. Each of these characters is a direct index to a commonly-used HTML tag or other phrase. Of the 123 possible system dictionary entries, 94 are currently used.

By default, 1024 available two-byte tokens can be used for the user-specified phrase dictionary. These tokens are stored in four sets of 256 entries of two-byte tokens:

- The first 256 entries are selected with characters in the range <377><000> to <377><377>.
- The next 256 entries are selected with characters in the range <376><000> to <376><377>.
- The fourth set of 256 entries are selected are selected with characters in the range <374><000> to <374><377>.
- The first 64 entries (<377><000> to <377><077>) are reserved by the RomPager engine for error messages.

The Extended ASCII character defined by `RpCompressionEscape` is used to signal that the character that follows should not be passed through the dictionary expansion process. This is useful for storing a single high-bit international character in a phrase that is otherwise all standard ASCII.

If a phrase dictionary will contain many phrases that use high-bit characters such as a dictionary with many Japanese or Chinese double-byte phrases, set the `theCompressionFlag` variable in the `RpSetUserPhraseDictionary` call to False, so that no further expansion is performed on phrases in the dictionary being set.

Using the PBuilder Compiler

CHAPTER 4

This chapter describes how to convert your HTML pages into source code that you can compile.

Overview

After you build HTML pages, you need to turn them into source code that you can compile. The internal format of a page consists of an object header structure, a list of object elements, and the object elements with the variable access structures. PBuilder converts HTML pages, `.gif`, `.jpeg`, `.pict`, and `.png` images, and Java applets into C source code files.

Copy the version of the application to the directory that contains your HTML pages. Start PBuilder, and enter the name of a file to be converted. PBuilder finds the file and creates the internal format C source file with a `.c` extension. PBuilder also creates a C source file with a `_v.c` extension that contains stub routines for `Set`/`Get` routines.

When you specify a file, but not a path, Pbuilder assumes the file is in the current directory or folder. If you specify a path, you must use a slash (/) as the path separator character. If you specify a file without specifying a file extension, PBuilder tries to open files with extensions that it knows about. PBuilder first tries `.pbb` or `.txt`, for a batch file, then tries the extensions for HTML, image, and applet files.

If a file has an HTML extension (`.htm`, `.html`, `.js`, or `.css`), PBuilder requires a user dictionary file, `RpUsrDct.txt`, which contains the list of items for the RomPager user dictionary. PBuilder uses the contents of `RpUsrDct.txt` to compress the output of the converted HTML. PBuilder also creates two files from `RpUsrDct.txt` that must be part of the RomPager project: `RpUsrDict.h` and `RpUsrDict.c`.

This table lists the file extensions for image files:

Image file	File extension
GIF	`.gif`
JPEG	`.jpeg` or `.jpg`
TIFF	`.tiff` or `.tif`
PICT	`.pict` or `.pct`
PNG	`.png`

The file extensions for Java files can be:

- .class
- .jar
- .cla or .cls

To process a batch of files in PBuilder, set up a batch file with the .pbb or .txt extension. Make sure each line of the batch file contains a separate name of a file to parse. When you run PBuilder, specify the batch file as the file name to process.

For example, to process the files Main.html, Logo1.gif, and Logo2.gif in one pass, the batch file would contain:

```
Main
Logo1
Logo2
```

If a batch file is processed, the CreateSingleSourceFile flag in PbSetUp.txt controls whether the batch output goes to one combined C source files or to separate files. In addition, PBuilder creates a master object list in a file named RpPages.c. RomPager requires this list so it can find the objects that are stored in ROM. On startup, PBuilder looks for a default batch file named PBuilder.pbb. If this file exists, PBuilder does not display a prompt, and it processes the contents of the file.

Controlling the PBuilder compiler

Two input files modify the behavior of the PBuilder compiler:

- PbSetUp.txt
- RpUsrDct.txt

The next sections describe these files.

PbSetup.txt

You use the `PbSetUp.txt` file to modify the PBuilder output for the requirements of the C compiler. (Some C compilers have different requirements for the format of the page items. Other modifications may be needed for different editors.)

You can modify PBuilder output by placing the `PbSetUp.txt` file in the same directory as PBuilder. If this file does not exist, the default behaviors are used. Each line of the `PbSetUp.txt` file is either empty, a comment line (the first non-white space character is a slash (/), or a command line. Command lines are used to modify the behavior of Page Builder. A command line contains a keyword and a value.

The format of a command line is: *xxxxx = yyyyy*

where *xxxxx* is the keyword and *yyyyy* is the value. A value is a string, integer, or flag depending on the keyword. The value is placed in quotes if it is a string.

This table shows the keywords and their descriptions, types, and default values:

Keyword	Type	Default value	Description
`AllowedCharactersPerLine`	Integer	80	Determines the length of lines for `eRpItemType_ExtendedAscii` and `eRpItemType_DataZeroTerminated` items.
`AllowMacroNameForNumbers`	Flag	False	If the flag is specified, no numeric checks are performed on numeric constants. This allows macro names to be passed through PBuilder to the C compilation that follows.
`BufferSizeMultiplier`	Integer	1	Determines the amount of memory to allocate for working buffers. Large files or files with many form items may need a larger value.
`CreateSingleSourceFile`	Flag	False	If the flag specified, the output from all conversions is written to a single file; otherwise, a file is created for each conversion of a file.

Keyword	Type	Default value	Description
DestinationPath	String	""	If the string is specified, the output from the conversions is written to the directory specified. Otherwise, the conversions are written to the current directory. The URL path separator (/) is used in the path name to separate directories.
DontCreateVariablesFile	Flag	False	If the flag is specified, the xxx_v.c file with stub or structured access routines is not created.
ExtendedAsciiEscapeChar	Integer	251 ('\373')	Used to quote extended characters that are not in an eRpItemType_ExtendedAscii item.
ExtendedAsciiGroupSize	Integer	10	Determines the number of standard ASCII characters between extended ASCII characters before terminating an eRpItemType_ExtendedAscii item.
IgnoreWhitespaceAfterEol	Flag	False	If the flag is specified, white space is ignored after an end of line is detected.
JustGenerateObjectList	Flag	False	If the flag is specified and a batch file is input, only the RpPages.c file with the object list is created. This flag can be useful in an automated make file environment.
LeadInChar	String	"char"	The string value is the type of eRpItemType_ExtendedAscii and eRpItemType_DataZeroTerminated items. It may be useful to change this to "const char" in some environments.
LeadInUnsignedChar	String	"unsigned char"	The string value is the type of eRpItemType_DataLengthEncoded items. It may be useful to change this to "const unsigned char" in some environments.

Keyword	Type	Default value	Description
MaxBytesInItem	Integer	498	Determines the maximum number of bytes in the initializer for eRpItemType_ExtendedAscii, eRpItemType_DataZeroTerminated, and eRpItemType_DataLengthEncodedit items.
MaxBytesPerLine	Integer	16	Determines the number of bytes to generate on a line for eRpItemType_DataLengthEncoded items.
MinimizeWhiteSpace	Flag	False	If the flag is specified, white space that will be provided by the system dictionary is removed.
TabLength	Integer	4	Set to the number of characters a tab replaces in your source code editor.
UseFileNameForUrl	Flag	False	If the flag is specified, the URL for an object includes the file name extension of the source file.
UseFilePathForUrl	Flag	False	If the flag is specified, the URL for an object includes the path of the file.
UseJavaScript	Flag	False	Specifies that structures are to be generated with pointers to JavaScript.

RpUsrDct.txt

The user dictionary file contains frequently-used phrases that are in the HTML but not in the system dictionary.

The format of the user dictionary file is similar to that of the PBSetUp.txt file. Each line of the file is either a:

- User-dictionary element
- Comment line (indicated by a slash as the first non-white space character)
- Blank line

A user-dictionary element contains a macro name and a phrase and uses this format:

```
C_S_RomPager = “RomPager”
```

where `C_S_RomPager` is the macro name and “`RomPager`” is the phrase. PBuilder searches for the phrases in the HTML and replaces them with the macro names.

In addition, PBuilder generates two files:

- `RpUsrDct.h`. Required to compile the pages that PBuilder builds; it contains lines like this:
  ```
  #define C_S_RomPager“\377\000”/* “RomPager” */
  ```
- `RpUsrDct.c`. Must be compiled and linked with the RomPager engine; it contains a structure like this:
  ```
  const char *gUserPhrases[] = {
      /* \377\000 -- C_S_RomPager */ “RomPager”,
                                .
                                .
                                .
  };
  ```

URL names created by PBuilder

When PBuilder creates the URL that is stored with an object, it uses several ways to determine the URL name. First, if the object is a page defined with HTML comments, a URL specified with the `RpUrl` keyword in the `RpPageHeader` tag is used (if it exists). If the source file is not an HTML file or does not use `RpPageHeader` with `RpUrl`, the URL is created from the file name of the source. The URL that is created by PBuilder takes different forms, depending on the settings of the `UseFileNameForUrl` and `UseFilePathForUrl` flags in the `PbSetup.txt` file.

This table shows how PBuilder creates the URL name for an HTML file with an input name of `/MyDirectory/MyPage.htm`:

		UseFilePathForUrl	
		True	**False**
UseFileNameForUrl	**True**	/MyDirectory/MyPage.htm	/MyPage.htm
	False	/MyDirectory/MyPage	/MyPage

This table shows the generated URL for an image object with an input file name of /MyDirectory/MyGif.gif:

		UseFilePathForUrl	
		True	**False**
UseFileNameForUrl	**True**	/MyDirectory/MyGif.gif	/Images/MyGif.gif
	False	/MyDirectory/MyGif	/Images/MyGif

Java applets — file extension .cla, .cls, or .class — are treated the same as image objects with this exception: if the UseFileNameForUrl flag is not set, the URL has a .class extension.

The Action keyword in the RpFormHeader tag specifies the URL that's used to find form objects. If the RpFormHeader tag or the Action keyword is not specified, the URL is created from the source file name without using the UseFileNameForUrl and UseFilePathForUrl flags.

If the previous HTML example had a single form, the created URL would be /Forms/MyPage. If the HTML file had two forms, the created URLs would be /Forms/MyPage_1 and /Forms/MyPage_2.

Avoid using special characters (alphanumeric) in URL names. Different browsers handle the characters differently when they present a request to the Advanced Web Server, and special characters can prevent a URL object from being found.

Stub Routines

CHAPTER 5

This chapter describes the `Set` and `Get` routines for accessing device variables.

Overview

The PBuilder compiler produces two files for a Web page input file:

- A file that contains the page data and variable access structures
- A file that contains stub routines for the variable access structures

For an input file name, the PBuilder output files is named `MyPage.c` for the page data and `MyPage_v.c` for the stub routines.

The stub routines are the `Get` and `Set` routines for accessing the device variables. When you build your Web application, you must complete the stub routines so they store and retrieve the device variables. Each variable on a page has a `Get` routine and a `Set` routine if the variable is part of a form. For examples of the stub routines that PBuilder generates, see the Appendix.

After you complete the stub routines, you can set the `DontCreateVariablesFile` flag in `PbSetup.txt` to prevent PBuilder from regenerating the stub routines file and overwriting your completed stub routines.

Structured access

In some cases, you might want to `Get` and `Set` all the variables for a page in one routine. For example, if each variable requires another task to retrieve information, it can be more efficient to retrieve or store the variables with one routine, rather than access the dynamic information one variable at a time. The RomPager environment provides structured access routines to support this capability.

When a page is displayed, the RomPager engine calls the structured access pre-processing routine. This routine accesses all the variables for a page and stores them in a temporary structure that PBuilder defined. The structure is stored in the user data area of the request control block. PBuilder creates a stub version of the structured access routine that you complete.

When the RomPager engine executes the `Get` functions, it calls the routines that PBuilder generates to retrieve the variables from the structure. You do not need to modify these routines, because PBuilder creates them specifically for the individual page structure.

When a form is processed, the engine uses the PBuilder generated `Set` routines to store the variables in the page structure. When the form post-processing structured access routine is called, it takes the information from the structure and stores it in the device variables. PBuilder creates the stub version of this routine that you complete.

Engine Exit Functions and Callback Routines

CHAPTER 6

This chapter describes the variable access structures for exiting to the device and the callback routines that are used to control the internal states of the Web server engine.

Overview

The RomPager engine makes calls to the device code at various points to build pages or process forms. Each form item on a page contains a variable access structure for exiting to the device to get or set the parameter values

Many callback routines control internal states of the Web server engine. These routines can be called from either the device `Get`/`Set` routines or the optional page/form processing routines. The callback routines pass back to the engine the pointer to the engine global data and other variables as appropriate.

The next sections describe the exit functions and callback routines.

Engine exit functions

```
typedef    Signed16 (*rpFetchSigned16FuncPtr)(void);
typedef    Signed16 (*rpFetchSigned16ComplexPtr)(void *theServerDataPtr,
                                    char *theNamePtr, Signed16Ptr
theIndexValuesPtr);
typedef    void (*rpStoreSigned16FuncPtr)(Signed16 theValue);
typedef    void (*rpStoreSigned16ComplexPtr)(void *theServerDataPtr,
                                    Signed16 theValue, char *theNamePtr,
                                           Signed16Ptr theIndexValuesPtr);
```

These prototypes show sample `Set` and `Get` functions for a signed 16-bit variable with simple and complex functions. The simple functions provide a simple way to pass back a variable value for a `Get` or to receive a value for a `Set`. The complex functions are passed additional information that can be useful in the function. A void pointer, `theServerDataPt`, points to the engine internal data structures and is passed back to the engine in the callback routines described in the next section. This variable is retrieved by calling `RpHSGetServerData()`. Where applicable, the HTML name of a form item is passed in a complex function using a char pointer (`theNamePtr`). The current state of the repeat group index values is passed as a pointer (`theIndexValuesPtr`) to an array of signed 16-bit values.

The first item of the array is index level 0, the next array item is index level 1, and so on.

```
typedef    void (*rpResetCheckboxArrayPtr) (void *theServerDataPtr,
                                  char *theNamePtr, Signed16Ptr
                                  theIndexValuesPtr);
```

If a repeating check box array (`eRpItemType_FormCheckboxDyn`) is used, an exit function is called at the beginning of the form processing. Use this function to reset all the check boxes in the array; the `Set` function is called only for the check boxes that have been checked on the form.

The function of type `rpResetCheckboxArrayPtr` is called with the server data pointer, a pointer to the name of the form item, and a pointer to the index variables array. This function also is called if the `eRpItemType_FormFixedMultiDyn` item is used, so that the list of menu selections can be reset.

```
typedef void (*rpResetVarSelectPtr)(void *theServerDataPtr);
```

If a `<SELECT MULTIPLE>` HTML form item with a variable list of menu items is used (`eRpItemType_FormVariableMultiSelect`, `eRpItemType_FormVariableMultiDyn`, `eRpItemType_FormVarValueMultiSelect`, or `eRpItemType_FormVarValueMultiDyn`, for example), an exit function is called at the beginning of the form processing. Use this function to deselect all the items in the list, because the `Set` function is called only for items that have been selected in the list.

```
typedef    void (*rpDynamicRepeatFuncPtr)(void *theServerDataPtr,
                                  Signed16Ptr theStart,
                                        Signed16Ptr theLimit,
                                        Signed16Ptr theIncrement);
```

The `repeat group dynamic page` element (`eRpItemType_RepeatGroupDynamic`) allows the device to set up an HTML table display based on current internal values. The device is called with the `RpDynamicRepeatFuncPtr` function, which passes in the server data pointer and pointers to the starting value, limiting value, and increment value of the table.

```
typedef    void (*rpRepeatWhileFuncPtr)(void *theServerDataPtr,
                                     Signed16Ptr theIndexPtr, void **
                                     theRepeatGroupValuePtr);
```

The `repeat group while` page element (`eRpItemType_RepeatGroupWhile`) also allows the device to set up an HTML table display based on current internal values. The device is called with a function (`rpRepeatWhileFuncPtr`) that passes in the server data pointer and a pointer to the index variables array.

The function also passes a pointer to an arbitrary value that the `repeat while` function can use. Initially, the pointer points to the value (`void *`) 0. On later calls, the pointer points to whatever the device stored on the previous call. The `repeat while` function continues to build the table until the device sets the pointer to point to a value of (`void *`) 0.

In addition to exits for each individual form element, the engine provides optional exits for page pre-processing and form post-processing. If this exit is used, the address of the function of type `rpProcessDataFuncPtr` is placed in the `fProcessDataFuncPtr` field of the `rpObjectExtension` structure. This function is passed to the server data pointer and the index values pointer.

The optional functions are for setting up a collection of data before the page is served, or for validating form elements after all the individual form items have been processed.

```
typedef void (*rpProcessCloseFuncPtr)(void *theServerDataPtr);
```

If the `RpSetRequestCloseFunction` routine is used to set up a close function, a function of type `rpProcessCloseFuncPtr` is called after the TCP connection has been closed.

```
extern rpPasswordState SpwdGetExternalPassword(void *theServerDataPtr,
Unsigned8 theConnectionId,
char *theRealmNamePtr,
char *theUsernamePtr,
char *thePasswordPtr,
Unsigned32 theIpAddress);

typedef    void (*rpSessionCloseFuncPtr) (void *theServerDataPtr,
                                          void *theUserCookie);
```

If the Web application provides URL authentication, the `SpwdGetExternalPassword` routine is called when the engine has a URL access to authenticate. A function of type `rpSessionCloseFuncPtr` is called for external password session timeouts.

Engine callback routines

A variety of callback routines are used to control the internal states of the Web server engine, including those for:

- Page flow and connection control
- ROM master object list
- Browser request
- User data
- Time access
- Query index
- Phrase dictionary
- HTTP event logging
- Form item handling
- Number to string conversion and string to number conversion
- Delayed functions – external tasks

The next sections describe these callback routines.

Page flow and connection control

```
extern char *RpGetSubmitButtonValue(void *theServerDataPtr);
extern char * RpGetCurrentUrl(void *theServerDataPtr);
extern void RpSetNextPage(void *theServerDataPtr,
rpObjectDescriptionPtr theNextPagePtr);
extern void RpSetNextFilePage(void *theServerDataPtr, char
*theNextPagePtr);
extern void RpSetRedirect(void *theServerDataPtr);
extern void RpSetRefreshTime(void *theServerDataPtr,
Unsigned16 theRefreshSeconds);
extern void RpSetRefreshPage(void *theServerDataPtr,
rpObjectDescriptionPtr theRefreshPagePtr);
extern void RpSetRequestCloseFunction(void *theServerDataPtr,
rpProcessCloseFuncPtr theFunctionPtr);
extern void RpSetConnectionClose(void *theServerDataPtr);
```

The page flow callback routines control the page state. This table describes the page flow callback routines:

Routine	Description
`RpGetSubmitButtonValue`	Returns a character pointer to the ASCII value of the `Submit` button. This routine is useful with forms that have more than one `Submit` button so the management application can determine which button was pressed.
`RpGetCurrentUrl`	Returns a character pointer to the URL that invoked the page or form. This routine is useful for processing routines that are common to multiple pages or forms.
`RpSetNextPage`	Passes in an object pointer to the ROM-based page to be served next after redirection. This routine is typically called in the optional form post-processing routine to override the value in the `fPagePtr` of the `rpObjectExtension` structure.
`RpSetNextFilePage`	Works the same way as `RpSetNextPage`, but passes a pointer to a null-terminated string that contains the next URL to be served. In this way, the next page to be served can be located in ROM, in the file system, or on a remote host. If the `RpSetNextPage` or `RpSetNextFilePage` function is called in the page pre-processing routine, normally the page item pointed to by the "Next" page is served directly. If it is useful to have the page served by browser redirection, use the `RpSetRedirect` call.
`RpSetRefreshTime` and `RpSetRefreshPage`	Can be used to dynamically set up the time in seconds and the page pointer to the next page. Setting the refresh time to 0 ends the page refresh cycle. The information also can be set up statically in the object extension fields. If these callback routine are used, they are effective only if called during page setup pre-processing. They are not effective if: ■ They are called during page item processing, because the HTTP headers have already been sent. ■ During form processing, because forms handling provides redirection to a new page that has its own refresh values.

Routine	Description
RpSetRequestCloseFunction	Used to set up a function that is called after the HTTP request has been completely processed and the TCP connection has been closed. The primary use for this function is to trigger device reset functions after being assured that a page with a Reset about to start message has been delivered.
RpSetConnectionClose	Used to force the TCP connection that is being used for the current HTTP request to close after the current HTTP request is completed. Both Netscape *keep alive* support and HTTP 1.1 persistent connections attempt to keep the TCP connection open for multiple HTTP requests from a single browser. In some cases, it can be useful for the server to free up a TCP connection for other users or other TCP applications.

ROM master object list

```
extern void RpSetRomObjectList(void *theServerDataPtr,
rpObjectDescPtrPtr
*theMasterObjectListPtr);
```

The ROM object master list consists of a list of pointers to object lists, each of which points to a set of ROM objects. The lists are searched linearly, so you can achieve some small gains by putting frequently used pages toward the front. The home or root page is the first object in the first list (index offset 0). No other order dependencies exist in the lists.

The RpSetRomObjectList callback routine can be used to tell the RomPager engine to use an alternative ROM object master list. Because the object master list is a set of pointers to individual ROM object lists, you can control the entire search list at runtime by varying the contents of the object master list. This can be useful for dynamically enabling/disabling feature sets.

Browser request

```
extern char * RpGetUserAgent(void *theServerDataPtr);
extern char * RpGetHostName(void *theServerDataPtr);
```

This table describes the browser request routines:

Routine	Description
RpGetUserAgent	Returns the information from the User-Agent HTTP header that the browser sends in with the request. This can be useful in distinguishing among browser types to decide the type of HTML to serve
RpGetHostName	Passes in the host name used by the browser to access the page. This can be useful for constructing fully-qualified URL references.

User data

```
extern void RpSetCookie(void *theServerDataPtr, void *theCookie);
extern void * RpGetCookie(void *theServerDataPtr);
extern void RpSetRequestCookie(void *theServerDataPtr, void *theCookie);
extern void * RpGetRequestCookie(void *theServerDataPtr);
extern void * RpGetRepeatWhileValue(void *theServerDataPtr);
```

This table describes the user data routines:

Routine	Description
RpSetCookie	Saves a pointer to arbitrary user data in the engine data structure for retrieval later with the RpGetCookie routine. The RpSetRequestCookie routine saves a pointer to arbitrary user data in the current request for retrieval later with the RpGetRequestCookie routine.
RpGetRepeatWhileValue	Can be used in individual Get functions that are called during processing of an eRpItemType_RepeatGroupWhile item. The routine retrieves the current value that was returned by the Repeat While function. This allows the individual Get functions to use the current repeat value as an index or in any other way to modify the value the Get function returns.

Time access

```
extern Unsigned32 RpGetMonthDayYearInSeconds(Unsigned32 theMonth,
          Unsigned32 theDay,Unsigned32 theYear);
extern Unsigned32 RpGetSysTimeInSeconds(void);
```

This table describes the time access routines:

Routine	Description
RpGetSysTimeInSeconds	Used internally by the RomPager engine and also can be used by the device page or form routines. This routine also returns the system time in internal format, which is seconds since 1/1/1901.
RpGetMonthDayYearInSeconds	Translates an external date into internal format.

Query index

```
extern Signed16 RpPopQueryIndex(void *theServerDataPtr);
extern void     RpPushQueryIndex void *theServerDataPtr,
                                 Signed16 theQueryValue);
extern Signed8  RpGetQueryIndexLevel(void *theServerDataPtr);
```

The query index callback routines are used to modify the query index state. This table describes the routines:

Routine	Description
RpPopQueryIndex	Pops a value off the stack and returns it to the caller. These routines can be used to control the index values that are passed to other pages. The values used in these routines are internal index values, so they are 0-relative.
RpGetQueryIndexLevel	Reports back the current index level in use starting at 0. If no index levels are in use, the value returned is -1.

Phrase dictionary

```
extern void RpSetUserPhraseDictionary(void *theServerDataPtr,
                                      char **theUserDictionaryPtr,
                                      Boolean theCompressionFlag);
extern void RpSetRequestUserPhraseDictionary(void *theServerDataPtr,
                                      char **theUserDictionaryPtr,
                                      Boolean theCompressionFlag);
```

A *phrase dictionary* is an array of char pointers indexed by characters used in the HTML text. The `RpSetUserPhraseDictionary` routine lets you to change the current user phrase dictionary. Using alternate user phrase dictionaries can be helpful in setting up pages that will have alternate appearances depending on the dictionary.

The *`theCompressionFlag`* variable is used to signal whether the phrases in the dictionary can contain other compressed phrases. If *`theCompressionFlag`* is True, the phrases in the dictionary being set can contain additional compressed phrases, from either the system or user dictionaries. This can reduce the size of a phrase dictionary if there are many common phrase fragments.

If *`theCompressionFlag`* is False, no further expansion is performed on phrases in the dictionary being set. As a result, any characters with high bits set are passed directly to the browser. If a phrase dictionary is being set up with many double-byte phrases (for example, Japanese or Chinese), setting `theCompressionFlag` to False reduces the overall size of the dictionary by eliminating the need for `RpCompressionEscape` characters.

To reset the current phrase dictionary to the default user phrase dictionary, use this call:

```
RpSetUserPhraseDictionary(theServerDataPtr, &gUserPhrases, True);
```

This routine changes the default user dictionary for the entire system. The `RpSetRequestUserPhraseDictionary` routine changes the user dictionary for a single page or form request. This routine can be called from within either the page pre-processing routine or the first item on a page or form to set the dictionary to be used for processing the rest of the items of that page or form.

Form item handling

```
extern char * RpGetFormBufferPtr(void *theServerDataPtr);
extern void RpGetFormItem(char ** theBufferPtr,
                                char * theNamePtr,
                                char * theValuePtr);
extern void RpSetFormObject(void *theServerDataPtr,
                            rpObjectDescriptionPtr theObjectPtr);
extern voidRpReceiveItem(void *theServerDataPtr, char *theNamePtr,
                            char *theValuePtr);
```

RomPager normally handles forms for the user application by processing items against a form item list pointed to by a form object. In some cases, however, an application may either want to handle its own form items or use some of the internal RomPager routines to do partial form processing.

This table describes the form item handling routines:

Routine	Description
RpGetFormBufferPtr	Returns a pointer to the form arguments passed in from the browser with either: ■ A GET Query command. Valid only in the pre-processing function of the GET Query command. ■ A POST command. Request arguments from HTML forms are passed as a group of name/value pairs encoded in the Form URL Encoding format.
RpGetFormItem	Used to decode and retrieve a name/value pair from the form buffer. The buffer pointer is updated to point to the next name/value pair, and the values of the current name/value pair are copied into the buffers passed as input to the call.
RpReceiveItem	Passes the name/value pair against the current ROM-based form object to use the standard RomPager processing to drive calls to the Set functions.
RpSetFormObject	Sets up the current form object.

Number to string conversion

```
extern Unsigned16 RpConvertSigned32ToAscii(Signed32 theNumber,
          char *theBufferPtr);
extern Unsigned16 RpConvertUnsigned32ToAscii(Unsigned32 theNumber,
          char *theBufferPtr);
```

This table describes the number to string conversion routines:

Function	Description
RpConvertUnsigned32ToAscii and RpConvertSigned32ToAscii	Used internally by the RomPager engine; the device page display routines also can use them. These routines take parameters of a signed or unsigned 32-bit number and a char pointer to the output buffer. The routines also return the number of characters in the converted string.

String to number conversion

```
extern void RpConvertHexString(void *theServerDataPtr,
                               char *theHexPtr, char *theValuePtr,
                               Unsigned8 theOutputCharCount, char
theSeparator);
extern void RpConvertDotFormString(void *theServerDataPtr,
                               char *theDotFormPtr, char *theValuePtr,
                               Unsigned8 theOutputCharCount);
extern Signed8 RpConvertStringToSigned8(void *theServerDataPtr,
                               char *theValuePtr);
extern Signed16 RpConvertStringToSigned16(void *theServerDataPtr,
                               char *theValuePtr);
extern Signed32 RpConvertStringToSigned32(void *theServerDataPtr,
                               char *theValuePtr);
extern Unsigned8 RpConvertStringToUnsigned8(void *theServerDataPtr,
                               char *theValuePtr);
extern Unsigned16 RpConvertStringToUnsigned16(void *theServerDataPtr,
                               char *theValuePtr);
extern Unsigned32 RpConvertStringToUnsigned32(void *theServerDataPtr,
                               char *theValuePtr);
extern rpItemError RpGetConversionErrorCode(void *theServerDataPtr);
extern void RpSetUserErrorMessage(void *theServerDataPtr,
                               char *theMessagePtr);
```

The RomPager engine uses the string to number routines internally during the forms item handling; the device form item routines also can use them.

The engine normally uses these routines to perform the conversion according to the numeric type set in the `fTextType` field:

- `RpConvertHexString`
- `RpConvertDotFormString`
- `RpConvertStringToSigned8`
- `RpConvertStringToSigned16`
- `RpConvertStringToSigned32`
- `RpConvertStringToUnsigned8`
- `RpConvertStringToUnsigned16`
- `RpConvertStringToUnsigned32`

If the device form item handling routine wants to either examine the string before the conversion or look at the number after the conversion, the `fTextType` field should be set to `eRpTextType_ASCII`; the function pointed to by the `fSetPtr` field can call the conversion routine.

The `RpGetConversionErrorCode` callback can be used to check the results of the conversion routine. The `RpSetUserErrorMessage` callback can be used to trigger an error page display with a custom message.

Delayed functions — external tasks

```
extern unsigned char RpInitiateDelayedFunction(void *theServerDataPtr);
extern void RpCompleteDelayedFunction(void *theServerDataPtr,
          unsigned char theConnection);
extern unsigned char RpGetCurrentConnection(void *theServerDataPtr);
extern unsigned char RpSetCurrentConnection(void *theServerDataPtr,
          unsigned char theConnection);
```

Preparing a page or processing form results can require access to an external task, which can incur a delay. Such a task might be a read or write from a database, or a request for information over an internal network.

To allow communication with these external tasks without suspending processing for other HTTP requests, the RomPager engine provides these callback routines for controlling the state of a RomPager internal task or connection:

Callback routine	Description
`RpInitiateDelayedFunction`	Called when a delayed function is started and returns the connection number of the internal RomPager task that will be suspended.The connection number should be saved away for use by the completion routine of the delayed function.
`RpCompleteDelayedFunction`	When the completion routine of the delayed function is ready to allow the RomPager internal task to resume processing, it should issue the `RpCompleteDelayedFunction` call with the saved connection number.

Callback routine	Description
RpSetCurrentConnection	If the completion routine of the delayed function issues any other RomPager callback routines such as RpSetNextPage, it needs to issue a RpSetCurrentConnection call to set up RomPager for the correct RomPager internal connection. The RpSetCurrentConnection call returns the old current connection that needs to be restored with another RpSetCurrentConnection call before the completion routine finishes.
RpGetCurrentConnection	Used to determine which connection is being used by the internal RomPager task.
RpInitiateDelayedFunction	Can be called from within the optional page pre-processing routine to set up local variables that will be accessed when the page is served or from the optional form post-processing routine to store away variables that were entered.

Here is an example of the processing flow. The first section of code is a RomPager task, the second section is a spawned external task, and the third section is a RomPager task.

```
/* begin page pre-processing */
/* spawn host OS task to do delayed processing */
/* suspend RomPager processing for the current connection */

theSuspendedConnectionId=RpInitiateDelayedFunction(theDataPtr);

/* allow RomPager task to process other connections */

return;



/* begin external task execution */
/* do external tasks */
/* wait for external task completion */
/* finish external tasks */
/* set the RomPager current connection to the suspended connection */
theCurrentConnectionId=RpSetCurrentConnection(theDataPtr,
theSuspendedConnectionId);

/* issue engine control calls for the suspended connection */

RpSetNextPage(theDataPtr, &PgResultsPage);
```

```
/* the suspended connection */

RpCompleteDelayedFunction(theDataPtr, theSuspendedConnectionId);

/* restore the RomPager current connection */

theCurrentConnectionId=RpSetCurrentConnection(theDataPtr,
theCurrentConnectionId);

/* end external task execution */

return;


          /* finish page pre-processing */
          /* start page display */
```

Dynamic Pages

CHAPTER 7

This chapter describes how to use the client pull method to work with dynamic pages.

Overview

The Web Application Toolkit supports the *client pull* method developed by Netscape (and fully supported by Microsoft Internet Explorer). With this method, an object that is served to the browser is sent with an HTTP Refresh header that tells the browser when to ask for the next page to be served.

The browser maintains a timer. If the user has not clicked another link, the browser automatically requests the next page after the timer has expired. The next page can be either:

- The same as the previous page
- A different page specified in the Refresh header

In a Web-based management application, this method can be useful for setting up a series of pages that each points to the next one. For example

- Status page A would be displayed for 15 seconds.
- Status page B would be displayed for 10 seconds.
- Status page C would be displayed for 30 seconds.
- Status page A would be displayed again, and the whole process would repeat.

Using the client pull method

To use the client pull method, you must specify a refresh time in seconds for the page. You can specify the refresh time in either of these ways:

- When you create the page, by specifying the `RpRefreshTime` keyword in the `RpPageHeader` tag
- At runtime, by specifying the `RpSetRefreshPage` callback routine

If a page other than the current page is to be refreshed, you can specify the new page in either of these ways:

- At page creation, by using the `RpRefreshPage` keyword
- At runtime by using the `RpSetRefreshPage` callback routine

You can end the page refresh cycle by either:

- Serving a page that has a refresh time of `0`
- Calling the `RpSetRefreshTime` routine with a value of `0`

Another way to use the client pull refresh method is to use the `<META HTTP-EQUIV>` tag in the page that is sent to the browser. This tag sends a header in the HTML page. The server ignores the header, but the browser treats the header as if it received the header with the rest of the HTTP headers from the server.

To use this technique, place `<META HTTP-EQUIV>` in the `<HEAD>` section of the page, as shown in the next examples:

```
<HTML>
<HEAD>
<TITLE>Refresh this page in 10 seconds</TITLE>
<META HTTP-EQUIV=“Refresh” CONTENT=“10”>
</HEAD>
<BODY>
Refresh this page in 10 seconds
</BODY>
</HTML>

<HTML>
<HEAD>
<TITLE>Refresh another page in 5 seconds</TITLE>
<META HTTP-EQUIV=“Refresh” CONTENT=“5;URL=http://www.allegrosoft.com/
index.html”>
</HEAD>
<BODY>
Refresh another page in 5 seconds
</BODY>
</HTML>
```

State Management

CHAPTER 8

This chapter describes the techniques the Advanced Web Server uses for state management.

Overview

The HTTP protocol is a stateless protocol. From the point of view of the protocol, each request from a client browser to a Web server is an individual request. From the point of view of a Web application written on the server, it can be useful to identify multiple requests from the same user and group them into a session.

The server creates a session by assigning a *state* or a *session ID* and sending it with the response. If the browser supports the technique the server is using, the browser returns the state on subsequent requests, allowing the server to recognize subsequent requests from the same user.

The Advanced Web Server supports two techniques for state management:

- **Hiding state information in the URL**. This low-overhead technique works with any browser but is limited in the amount of state information that can be passed.
- **Using HTTP cookies**. The HTTP cookies technique, developed by Netscape, is supported by most recent browsers.

  Although HTTP cookies have somewhat higher overhead than URL hiding, they allow greater amounts of state information to be passed back and forth.

The next sections describe the two methods.

Hiding technique

The RomPager URL hiding technique works by prepending the URL in a link with a signal sequence and a state value. When the Web application wants to set a state for a user, it uses the `RpSetUrlState` call to store a string into the Web server. When a link or path name is generated with `RpUrlState`, it has this form:

`http://deviceaddress/US/`*`SSSSS`*`/pathname`

where *`SSSSS`* is the value of the state string.

On later requests, the URL is examined to see if it is in this format. If it is, the state information is stripped off from the URL and stored where the application can retrieve it using `RpGetUrlState`. The rest of the URL pathname is passed to the engine for action.

You can clear the URL state at the server with the `RpSetUrlState` call to set the state to a null string. When the browser quits, it stops using the special URLs, and the session is terminated.

HTTP cookie technique

The HTTP cookies technique works by sending state information in special HTTP headers. While these headers are non-standard, most recent browsers support them. The HTTP cookie headers allow multiple states to be passed between the server and the browser.

In the Advanced Web Server, if HTTP cookies are enabled, an array of character strings is available for storing and retrieving state information. The size of the array is 10, and the cookie size is 40.

The `RpSetHttpCookie` call is used to set a cookie value that will be passed to the browser and the `RpGetHttpCookie` call is used to retrieve the cookie that the browser has sent in. These calls use `theIndex` as a parameter to choose the string in the array to reference. After a cookie has been sent to a browser, it will send it in with each subsequent request. When the user quits the browser, it deletes the stored cookies, and the session is completed.

With both techniques, a session is a series of independent requests that the server has asked the browser to tag with state information. The browser does not notify the server when the session completes. The server can deduce that a session has completed by observing that no requests from a tagged user have come in for a while. Application timers can be used to determine session ends so that session resources can be freed up on the server side.

For `RpGetHttpCookie`, this prototype is required, where the index is `0` through `9`, and the function returns a pointer to the cookie:

```
char * RpGetHttpCookie(void *theTaskDataPtr,
Unsigned8 theIndex);
```

For `RpSetHttpCookie`, this prototype is used, where the index is again `0` through `9`, and theCookie points to a string representing the cookie:

```
void RpSetHttpCookie(void *theTaskDataPtr,
Unsigned8 theIndex, char * theCookie);
```

The cookie size should be less than 40 bytes.

This table describes the routines:

Routine	Description
`RpGetHttpCookie`	Retrieves the cookie that the browser sends in. Uses `theIndex` as a parameter to choose the string in the array to reference.
`RpSetHttpCookie`	Sets a cookie value that is passed to the browser. Uses `theIndex` as a parameter to choose the string in the array to reference. `theCookie` is a pointer to a cookie value.

Security Control

CHAPTER 9

This chapter describes internal Web server security environment.

Overview

The internal Web server security environment consists of a user database, the individual pages and other resources to be protected, and a set of security realms. A *security realm* is a collection of objects (pages, forms, images, and so on). An individual page can belong to one or more realms, and an individual user can have access to one or more realms. If an object belongs to more than one realm, RomPager uses the security method of the the lowest numbered realm in the challenge.

The first time a browser request (`Get`, `POST`, and so on) is made for an object in a given realm, the user is prompted for a known username and password. When a user logs on, a security session is established for that user. The browser negotiates subsequent requests so users don't need to re-enter the username and password for each request. The request, however, is still authenticated, because the browser carries the user's authentication credentials for each subsequent request.

The HTTP protocol currently supports two authentication methods:

- **Basic method.** Encodes the username-password using the Base64 encoding scheme and sends it over the wire to the server, which decodes it and compares it with the known username-password.

  The basic method is supported by all browsers and provides some protection in that another browser user cannot access the protected realm without knowing the username-password.

  Because a determined attacker can use a packet sniffer to gain the encoded username-password string, and because the Base64 algorithm is a widely-known, reversible encoding scheme, this method provides minimal protection but is better than plain text passwords.
- **Digest method**. Provides greater protection because it uses one-way encoded hash numbers and never sends the username-password over the wire. With this method, the server creates a challenge code and sends it to the browser. The browser uses both the MD5 one-way hash algorithm to create a response based on the challenge, and the user-provided username-password. The server calculates the expected response, using its copy of the username-password and challenge, and compares it with the actual response. If they match, access to the object is granted.

Because the challenge can change with each transaction, and the username-password is never sent over the wire, this method provides fairly strong identification. The challenge is usually made unique by incorporating such things as the device IP address and the time, so each device sends out unique challenges.

The digest method is appropriate for an embedded environment, but at present has not been widely implemented by most browsers. Internet Explorer 5.0 and above supports part of the digest authentication specification.

The AWS supports both basic and digest authentication. You can assign pages and forms to eight realms. The realms can overlap, allowing you to create supersets and sidesets. You can, for example, have one realm for regular users, another for supervisory users, and a third for field engineering users. The supervisory and field engineering users might each have access to all the pages and forms of the regular users, but no access to each other's unique pages and forms.

Implementation

The security database contains a user data base and a realm data base. Each user has a username, password, and access code that indicates the realms that the user has access to and a session timer that indicates how long the user has access to the realm without being challenged again. Each username is unique in the database. The number of users to allow in the security database is 32.

A maximum of eight realms can each have a realm name. The realm name is passed to the browser so it can be displayed in security dialogs to the user. Realms are accessed by an index of 0 to 7 in the security callback access routines.

The individual pages and forms have an access code that indicates the security realms to which the object belongs. The access codes for an object are specified in the HTML source using the RomPager comment tags shown here:

- To set the access code of a page, use the `RpAccess` parameter of the `RpPageHeader` comment tag.
- To set the access code for a form, use the `RpAccess` parameter of the `RpFormHeader` comment tag.

The internal storage of the object access codes is in the `fObjectAccess` field of the `rpObjectDescription` structure.

The default for each object is `kRpPageAccessUnprotected`. An object can belong to multiple realms by setting multiple flags. The User Settings (`PgUsers`) page in the sample pages is an example page that belongs to the first and second realms. Using overlapping realms allows the creation of superuser realms where one realm has complete access and another realm has only partial access.

When an object that belongs to a protected given realm is accessed, the AWS sends a challenge to the browser the first time an object in the realm is accessed. The browser displays a request to the user that shows the realm name and asks for a username and password. Different browsers use different dialog formats for the password request, but all display the username in clear text and the password in obscured form. With HTTP basic authentication, the username-password combination is returned to the server in the Base64 encoding scheme. When the server receives a valid username-password, it establishes a security session for that user. As long as the session is active, users have access to the objects in all the realms for which they are authorized.

As the user makes additional requests, the browser sends along the encoded username-password combination that was last entered. If the user has requested another object in the same realm, the server uses the received username-password combination to authorize the request. If the user requests an object in another realm, the server issues another challenge to the browser, which in turn asks the user for a new username-password combination.

The RomPager engine provides a series of callback routines that can be used to dynamically administer the security database. The `RpHSInitSecurityTable` routine (in the `security.c` file in `netos/src/examples/nahttp_pd`), initializes the security tables using these callback routines. The security callback routines are used for realm database manipulation, user database manipulation, session control and feature access control. These routines can be called at initialization, at various exit points in the Web application, or from other device management functions such as SNMP or Telnet environments.

Realm manipulation routines

```
extern char *               RpGetRealmName(void *theTaskDataPtr, unsigned
char theIndex);
extern void                 RpSetRealmName(void *theTaskDataPtr, unsigned
char theIndex,
              char * theReealNamePtr);
extern rpSecurityLevel RpGetSecurityLevel(void*theTaskDataPtr,
                       unsigned char theIndex);
extern void                       RpSetSecurityLevel(void*theTaskDataPtr,
                     unsigned char theIndex,
                     rpSecurityLevel theSecurityLevel);
extern void                       RpSetRealmLocking (void *theTaskDataPtr,
                     Boolean theLockState);
```

This table describes the routines:

Routine	Description
RpGetRealmName	Is passed an index (0–7) of the realm and returns a pointer to an ASCII string that contains the current realm name.
RpSetRealmName	Is passed an index (0-7) of the realm and a pointer to an ASCII string that contains the new realm name. The realm name is sent to the browser when a protected object is accessed, so that the name can be displayed in security dialogs to the user.
RpGetSecurityLevel	Is passed an index (0–7) of the realm and returns a security level code.
RpSetSecurityLevel	Is passed an index (0–7) of the realm and returns a security level code that indicates the new security level for the realm.

This table describes the available security level codes:

Security level	Description
eRpSecurity_Disabled	Turns off security for a particular realm. Pages and objects that belong to a realm with this level are unprotected.
eRpSecurity_PasswordOnly	Allows a user with knowledge of the username and password access to the realm. The HTTP basic authentication mechanism is used for this level.

Security level	Description
eRpSecurity_PasswordAndIpAddress	Requires users to provide a username and password and run a browser from a specified IP address to gain access to the realm. The HTTP basic authentication mechanism is used for this level.
eRpSecurity_DigestPasswordOnly	Allows a user with knowledge of the username and password access to the realm. The HTTP digest authentication mechanism is used for this level. A new challenge is created once per session. This method is supported by Internet Explorer 5.0 and above.
eRpSecurity_DigestAndIpAddress	Requires the user to provide the username and password and be running a browser from a specified IP address to gain access to the realm. The HTTP digest authentication mechanism is used for this level. A new challenge nonce is created once per session. This method is supported by Internet Explorer 5.0 and above.

Multiple users who have access to the same realm, or multiple users who are using the same username/password identifier, can access the same pages and resources from the Web server. Multiple access is fine for pages that only display information.

For pages that have forms to control parameter values, however, multiple access can cause problems. If two users each access the same control page at the same time, and each makes changes based on the information they see and then submits a form, the first set of changes is lost, because the second user overrides them.

To prevent this possibility and allow control of critical resources, the security system offers a concept called *realm locking*, which is controlled by the `RpSetRealmLocking` callback routine. The `RpSetRealmLocking` routine is called to enable and disable realm locking. If realm locking is enabled, when a user logs on successfully, all the pages in the realms the user has access to become reserved for that user until the user completes the session. Other users who log on (even if they normally have access to a given realm) are blocked from accessing the pages and forms in that realm until the first user logs off or realm locking is disabled. This capability is useful for ensuring that device information is not changed by multiple users at the same time.

User database manipulation routines

```
extern   Boolean RpSetUserAttributes(void *theTaskDataPtr,
char *theUsernamePtr,
                                     char *thePasswordPtr,
rpAccess theAccessCode
                                     Unsigned32 thelpAddress,
                                     Unsigned16 theTimeoutSeconds);
extern void  RpGetUserAttributes(void *theTaskDataPtr,
Unsigned16 theUserlndex,
                                     char **theUsernamePtr,
char **thePasswordPtr,
                                    rpAccess *theAccessCodePtr,
                                    Unsigned32 *theIpAddressptr,
                                    Unsigned16 *theTlmeoutSecondsPtr);
extern void     RpDeleteUser(void*theTaskDataPtr, char *theUsernamePtr);
extern char *   RpGetCurrentUserName(void *theTaskDataptr);
extern void     RpSetPasswordCookie(void *theTaskDtrPtr,
char *theUsernamePtr,
                                     void *theCookie);
extern void*    RpGetPasswordCookie(void *thetaskDataPtr,
                                     char *theUsernamePtr);
```

This table describes the routines:

Routine	Description
`RpSetUserAttributes`	Called to create a user entry for a new user or modify the attributes of an existing user. Each username in the user database must be unique. The first time `RpSetUserAttributes` is called for a specific username, a new entry is set up in the user database. Additional calls with the same username can be used to change the attributes for a specific user. `RpSetUserAttributes` returns False if there is no more room in the user database. If the value of the `IpAddress` parameter is `0`, any IP address is acceptable. If an IP address is specified, objects that belong to a realm with an appropriate security level code (such as `RpSecurity_PasswordAndIpAddress)` require the user to run a browser at the specified address to access the object. If the value of the `theTimeoutSeconds` parameter is `0`, the server's master security session timeout value is used.

Routine	Description
RpGetUserAttributes	Is passed an index (0-based) into the user routine returns the username, password, realm access code, IP address, and session value for the given user entry. If there is no user entry for the requested index, the return value of theUsernamePtr is NULL. This routine is useful for sequentially accessing the RomPager security database to store the information somewhere else. The user database entries are returned sequentially, so after an empty user entry is returned, no more user entries with a higher index.
RpDeleteUser	Is passed a pointer to an ASCII string that contains the name of the user to be deleted; used to remove an active entry from the user database. If the user being deleted has an active security session, the session is terminated, and the user entry is deleted.
RpGetCurrentUserName	Can be called from within a page to determine the name of the user who has authenticated access. It returns a pointer to an ASCII string that contains the name of the current authenticated user. The internal user security database can store an opaque variable that can be used for communicating with external password servers such as Radius servers. The opaque variable (or cookie) is set using the RpSetPasswordCookie routine and retrieved using the RpGetPasswordCookie routine.

Session manipulation routines

```
extern void RpSetCurrentSessionCloseFunction(void *theTaskDataPtr,
      rpSessionCloseFuncPtr theFunctionPtr, void *theUserCookie);
typedef void (*rpSessionCloseFuncPtr) (void *theTaskDataPtr,
             void *theUserCookie);
extern void
RpSetServerPasswordTimeout(void*theTaskDataPtr,
                           Signed16 theTimeoutSeconds);
extern Boolean             RpCheckSession(void *theTaskDataPtr,rpAccess
theAccessCode);
exter void                 RpResetCurrentSession(void*theTaskDataPtr,
                           Signed16Ptr thelndexValuesPtr);
extern void RpResetUserSession(void *theTaskDataPtr, char*theUsernamePtr);
```

This table describes the routines:

Routine	Description
RpSetCurrentSessionCloseFunction	Used to set up a routine that is called when the current user security session completes. Typically, this happens when the user's security session timer expires. The value of the security session timer is set when the user entry is created or modified with RpSetUserAttributes. An opaque variable (theUserCookie) can be passed with RpSetCurrentSessionCloseFunction. This variable is returned by a completion function (of type rpSessionCloseFuncPtr) when the session times out.
RpSetServerPasswordTimeout	Used to change the server's master security session timeout value. This timeout value is set at initialization time to the value defined by kPasswordSessionTimeout in RpCanfig.h. The master security session timeout value is used as the default timeout value for calls to the RpSetUserAttributes routine that have the theTimeoutSeconds parameter set to 0. The RomPager engine maintains timeout counters for each user session. The timeout counter is initialized to the timeout value for the user each time a protected object is accessed. If the counter hits 0, the server forces a challenge to the browser, even if the correct username-password combination was sent in with the request. This technique can be used to protect objects from access by an unauthorized user who uses an unattended browser at an authorized user's desk.
RpCheckSession	Used to find out the current access state for a realm or realms. The call passes in an access code that contains the realms to be checked and receives back a Boolean value that indicates whether access is currently authorized for the realms.

Routine	Description
RpResetCurrentSession	Can be called from within a page to reset the security session for the current authenticated user. The user security session is reset and any locked realms released. Further access by the user results in a re-challenge. This call uses the rpProcessDataFuncPtr format so that it can be issued directly from a page or form to terminate a security session for the current user.
RpResetUserSession	Can be called from any point within the device; forces the reset of specific user security session

Access control routine

The security access codes specifies one or more realms that a given object or function belongs to. The format of the access code uses flags such as kRpPageAccessRealml, kRpPageAccessRealm2, and so on, to identify the realms.

Access codes for pages and forms that are stored in ROM are assigned by using HTML comment tags in the pages that are compiled with PageBuilder. The access codes for objects stored in the file system are returned when the file is opened. The RpSetlppPrinterName callback routine allows the application to specify an IPP printer name and security access code.

Because the PUT command can directly load a file into the device, only users who are authorized for the realms specified by the theAccessCode parameter can upload files to the device.

External security validation

```
extern rpPasswordState  SpwdGetExternalPassword(void*theTaskDataPtr,
                                Unsigned8 theConnectibnld;
                                char *theUsernamePtr,
                                char *thePasswordPtr,
                                Unsigned32 *thelpAddressPtr,
                                rpAccess *theRealmAccessPtr);
```

Normally, the RomPager engine maintains internal security databases for handling HTTP user authentication requests. Another approach to user authentication is to have the RomPager engine handle the HTTP security protocols and pass the actual authentication request to a process elsewhere in the device. To enable this external password facility, define the `RomPagerExternalPassword` flag in `RpConfig.h` as 1.

When the External Password feature is enabled, the `SpwdGetExternalPassword` routine is called to obtain the external information the server needs to validate the information supplied by the browser.

`SpwdGetExternalPassword` provides a pointer to the username that needs to be validated, and pointers for the routine to return validation information. The external validation routine is also provided with a connection number that can be used to aid in processing multiple simultaneous validation requests.

The external validation routine must return three pieces of information so the AWS can finish the validation process:

- The valid password must be copied to the buffer the thePasswordPtr parameter points to.
- If the user should be restricted to making accesses from a particular IP address, this address should be placed in the area pointed to by the `theIpAddressPtr` parameter.
- If no restrictions are placed on the IP address, the value of the address should be set to 0.

Lastly, the validation application should provide the realm access code that specifies the realms the user has access to.

The external validation routine is called the first time the browser provides user credentials after being challenged. For subsequent requests for the same user session, the AWS uses the information provided by the external validation routine to validate these requests.

If the user session is terminated, or timed out for inactivity, or if a different user attempts access, the external password function is called again to provide the validation information.

From the point of view of the host operating system, the RomPager engine is a single task. The engine contains its own scheduler and control blocks for supporting multiple HTTP requests. The call that the engine makes to the external password routine must be asynchronous for any activity that incurs delay to allow the engine to service other requests. The call return state determines whether the external password function has been completed. If the username is to be validated on an external password server, any verification operation that can incur delay needs to create a separate operating system task that can block on call completion.

These are the possible return states from `theSpwdGetExternal` Password call:

- `eRpPasswordPending` — The external authorization process is not yet complete. The RomPager engine calls the `SpwdGetExternalPassword` routine again to retrieve the validation information.
- `eRpPasswordNotAuthorized` — The external process is complete; the user is not authorized because the username was not recognized. The AWS rejects the request.
- `eRpPasswordDone` — The external process is complete. The user is partially authorized. The string pointed to by `thePasswordPtr` has been filled in with the authorized password from the external database. The realm access flag has been set. The IP address (if any) has been set. The AWS completes the rest of the authorization process.

Internal Object Structures

CHAPTER 10

This chapter describes the ROM object list and the object header.

Overview

The RomPage engine uses:

- The ROM object list to locate stored ROM objects
- The object header to analyze HTTP requests and prepare responses

The next sections describe the ROM object list and the object header.

ROM object list

The *ROM object list* consists of a master object list that points to individual object lists, each of which points to a set of ROM objects. The RomPager engine uses the ROM object list to find the stored ROM objects (pages, forms, graphics, applets, and so on). The lists are searched linearly, so you can achieve some small performance gains by putting the frequently accessed pages toward the front.

The home page – or root page – is the first object in the first list (`Index offset 0`). No other order dependencies exist in the lists.

The ROM object list for a single page application, in `RpPage1.c`, looks like this:

```
rpObjectDescPtrPtrgRpMasterObjectList[] = {
                                        gRpObjectList,
                                        0
};
rpObjectDescriptionPtrgRpObjectList[] = {
                                        &PgFirstPage,
                                        0
};
```

The ROM object list for the multi-page demo application, shown in `RpPages.c`, looks like this:

```
extern rpObjectDescriptionPtr gRpValidationObjectList[];
extern rpObjectDescriptionPtr gRpChassisObjectList[];
extern rpObjectDescriptionPtr gRpJavaDemoObjectList[];
extern rpObjectDescriptionPtr gRpRomPopObjectList[];
```

```
rpObjectDescPtrPtrgRpMasterObjectList[] = {
                                        gRpObjectList,
                                        gRpValidationObjectList,
                                        gRpChassisObjectList,
                                        gRpJavaDemoObjectList,
                                        gRpRomPopObjectList,
                                        0
};
                                        rpObjectDescriptionPtr gRpObjectList[]
= {
                                        &PgMain,
                                         ...
                                        0
};
```

Object headers

The RomPager engine uses the object header to analyze the HTTP request and prepare the necessary HTTP headers in the response. The source definitions of the object header structures are in the `RpPages.c` file.

This is the structure of the `rpObjectDescription`:

```
typedef struct rpObjectDescription {
                              char *            fURL;
                              rpItem *          fItemsArrayPtr;
                              rpObjectExtensionPtrfExtensionPtr;
                              Unsigned32        fLength;
                              rpAccess          fObjectAccess;
                              rpDataType        fMimeDataType;
                              rpObjectType      fCacheObjectType;
}          rpObjectDescription, *rpObjectDescriptionPtr;
```

This table describes the fields:

Field name	Description
fURL	Contains the path used to find the object
fItemsArrayPtr	Points to the list of elements used to process the object.
fExtensionPtr	Points to an optional extension structure discussed below.
fLength	Contains the length of the object, if it is known. Images and applets are usually fixed-length items whose length are filled in by PBuilder. The length of HTML pages with dynamic text or form items is calculated at run time.
fObjectAccess	Contains the security realms for the object: ■ An unprotected object has the value kRpPageAccess_Unprotected in this field. ■ An object that belongs to both realms 1 and 4 has the value kRpPageAccess_Realm1+kRpPageAccess_Realm4 in this field.
fMimeDataTypefield	Contains the MIME type of the HTTP object. Typical values are: ■ eRpDataTypeHtml ■ eRpDataTypeImageGif
fCacheObjectType	Contains information that controls browser caching: ■ Static objects can be cached by the browser ■ Dynamic objects are forced to update because the information on the page may have changed. The values stored in this field are eRpObjectTypeStatic and eRpObjectTypeDynamic.
rpObjectExtension structure (optional)	Used by all form objects and other objects that need additional capabilities. The structure looks like this: typedef struct rpObjectExtension { rpProcessDataFuncPtr fProcessDataFuncPtr; rpObjectDescription *fPagePtr; rpObjectDescription *fRefreshPagePtr; Unsigned16 fRefreshSeconds; rpObjectFlags fFlags; char *fJavaScriptPtr; }rpObjectExtension, *rpObjectExtensionPtr;

Field name	Description
fProcessDataFuncPtr	Points to an optional processing routine that can be called for the object: ■ If the object is a page, the routine is called before any element processing occurs. ■ If the object is a form, the routine is called after all the individual form elements are processed.
fPagePtr	Points to the page that will be served after the form is processed. This field can be overridden using the RpSetNextPage or RpSetNextFilePage callback routine.
fRefreshSeconds	Used to specify the length in seconds of the refresh time.
fRefreshPagePtr	Points to the object to be served after the refresh time expires. For a discussion of refreshing pages with client pull, see the chapter "Dynamic pages."
fFlags	Contains optional flags that control special behavior for the object: ■ If the kRpObjFlag_Direct flag is set on a form object, the HTML for the next page object is sent directly. Normally, the HTML for the next page object is sent using HTTP redirect commands to preserve the browser cache and history information. ■ kRpObjFlag_Aggregate. Used to identify a page that will be sent by RomPager with embedded images. ■ kRpObjFlag_Disposition. Used to identify a page that is sent to the browser with an HTTP header of Content-Disposition: Attachment. Some browsers use this header to store a page directly as a file, rather than interpreting it.
fJavaScriptPtr field	Contains a pointer to the Java Script that is inserted into the <FORM> tag if JavaScript is being used.

Appendix: PBuilder HTML Comment Tag Examples

The example pages in this appendix are taken from the demonstration pages shipped with the Advanced Web Server. To see the pages in action, compile the demo pages with your device and use a standard browser to view the pages.

Example 1: Single text validation page

HTML with comment tags

```
<!-- RpPageHeader RpUrl=/SingleTextValidation -->
<HTML>
<HEAD>
<TITLE>Single Text Validation</TITLE>
</HEAD>
<BODY>
<H1 ALIGN=CENTER>RomPager Single Text Validation</H1>
<P>This page has an entry field for text that is not converted.</P>
<!-- RpFormHeader ACTION="/Forms/SingleTextValidation" -->
<FORM METHOD="POST" ACTION="/Forms/SingleTextValidation">
<!-- RpEnd -->
<!-- RpFormInput TYPE="TEXT" NAME="TextInput" SIZE="20" MAXLENGTH="20"
           RpGetType=Direct RpGetPtr=theTextValue
           RpSetType=Direct RpSetPtr=theTextValue RpTextType=ASCII -->
<INPUT TYPE="TEXT" NAME="TextInput" SIZE="20" MAXLENGTH="20"
VALUE="Initial Text">
<!-- RpEnd -->
<BR>
<BR>
<!-- RpFormInput TYPE="SUBMIT" NAME="Submit" VALUE="Submit" -->
<INPUT TYPE="SUBMIT" NAME="Submit" VALUE="Submit">
<!-- RpEnd -->
<!-- RpFormInput TYPE="RESET" VALUE="Reset" -->
<INPUT TYPE="RESET" VALUE="Reset">
<!-- RpEnd -->
<P> </P>
Return to: <A HREF="ValidationSuite">Validation Main Page</A>
</FORM>
</BODY>
</HTML>
```

C source file generated by PBuilder

```
#include “RpExtern.h”

extern rpObjectDescription PgSingleText;

static char PgSingleText_Item_1[] =
          C_oHTML_oHEAD_oTITLE “Single Text Validation”
C_xTITLE_xHEAD_oBODY “<H1”
          C_ALIGN_CENTER “>“ C_S_RomPager “ Single Text Validation” C_xH1
C_oP
          “This page has an entry field for text that is not converted.”
          C_xP;

extern char *theTextValue;
static rpTextFormItem PgSingleText_Item_3 = {
          “Text”,
          &theTextValue,
          &theTextValue,
          eRpVarType_Direct,
          eRpVarType_Direct,
          eRpTextType_ASCII,
          20,
          20
};
static char PgSingleText_Item_4[] =
          C_oBR
          C_oBR;
static rpButtonFormItem PgSingleText_Item_5 = {
          “Submit”
};
static rpButtonFormItem PgSingleText_Item_6 = {
          “Reset”
};
static char PgSingleText_Item_7[] =
          C_oP_NBSP_xP “\n”
          “Return to: “ C_oANCHOR_HREF “ValidationSuite\”>Validation Main
Page”
          C_xANCHOR C_xFORM
          C_xBODY_xHTML;
```

```
static rpItem PgSingleText_FormItems[] = {
          { eRpItemType_FormAsciiText, &PgSingleText_Item_3 },
          { eRpItemType_FormSubmit, &PgSingleText_Item_5 },
          { eRpItemType_FormReset, &PgSingleText_Item_6 },
          { eRpItemType_LastItemInList }
};
static rpObjectExtension PgSingleText_FormObjectExtension = {
          (rpProcessDataFuncPtr) 0,
          &PgSingleText,
          (rpObjectDescriptionPtr) 0,
          0,
          kRpObjFlags_None
};
rpObjectDescription PgSingleText_Form = {
          "/Forms/SingleTextValidation",
          PgSingleText_FormItems,
          &PgSingleText_FormObjectExtension,
          (Unsigned32) 0,
          kRpPageAccess_Unprotected,
          eRpDataTypeForm,
          eRpObjectTypeDynamic
};
static rpItem PgSingleText_Items[] = {
          { eRpItemType_DataZeroTerminated, &PgSingleText_Item_1 },
          { eRpItemType_FormHeader, &PgSingleText_Form },
          { eRpItemType_FormAsciiText, &PgSingleText_Item_3 },
          { eRpItemType_DataZeroTerminated, &PgSingleText_Item_4 },
          { eRpItemType_FormSubmit, &PgSingleText_Item_5 },
          { eRpItemType_FormReset, &PgSingleText_Item_6 },
          { eRpItemType_DataZeroTerminated, &PgSingleText_Item_7 },
          { eRpItemType_LastItemInList }
};
rpObjectDescription PgSingleText = {
          "/SingleTextValidation",
          PgSingleText_Items,
          (rpObjectExtensionPtr) 0,
          (Unsigned32) 0,
```

```
            kRpPageAccess_Unprotected,
            eRpDataTypeHtml,
            eRpObjectTypeDynamic
};
```

C stub routines file generated by PBuilder

```
#include “RpExtern.h”
char *theTextValue = “ “;
```

HTML without comment tags as served by the Advanced Web Server

```
<HTML>
<HEAD>
<TITLE>Single Text Validation</TITLE>
</HEAD>
<BODY>
<H1 ALIGN=center>RomPager Single Text Validation</H1>
<P>
This page has an entry field for text that is not converted.</P>
<FORM METHOD=“POST” ACTION=“/Forms/SingleTextValidation”>
<INPUT TYPE=“TEXT” NAME=“Text” SIZE=“20” MAXLENGTH=“20” VALUE=“Initial
Text”>
<BR><BR>
<INPUT TYPE=“SUBMIT” NAME=“Submit” VALUE=“Submit”>
<INPUT TYPE=“RESET” VALUE=“Reset”>
<P> </P>
Return to: <A HREF=“ValidationSuite”>Validation Main Page</A>
</FORM>
</BODY>
</HTML>
```

Example 2: Configure network info page

HTML with comment tags

```
<HTML>
<HEAD>
<TITLE>RomPager Configure Network Info</TITLE>
</HEAD>
<BODY>
<P>
<H2><CENTER>Configure Network Address</CENTER>
</H2>
<!-- RpFormHeader ACTION="/Forms/NetworkInfo" -->
<FORM METHOD="GET" ACTION="/Forms/NetworkInfo">
<!-- RpEnd -->
<P>
<HR>
Return to: <A HREF="/Menu">The RomPager Menu</A> or 
<A HREF="/Main">The RomPager Main Page</A><HR>
<P>


Variables in a form can be entered and displayed in dotted decimal format
and hex format, as well as other numeric formats.  This page is formatted
using table commands, and will look best with Netscape 1.1 or other
browsers that support tables.
<BLOCKQUOTE>
<BR>
<TABLE><CAPTION><B>TCP/IP Address Information</B></CAPTION><TR><TD>IP
Address:</TD>
<TD>
<!-- RpFormInput TYPE="TEXT" NAME="IpAddress" SIZE="15" MAXLENGTH="15"
           RpGetType=Direct RpGetPtr=theIpAddress
           RpSetType=Direct RpSetPtr=theIpAddress RpTextType="DotForm" -->
<INPUT TYPE="TEXT" NAME="IpAddress" SIZE="15" MAXLENGTH="15"
VALUE="0.0.0.0">
<!-- RpEnd -->
</TD>
<TD>X'
<!-- RpDisplayText RpGetType=Direct RpGetPtr=theIpAddress
```

```
            RpTextType=Hex RpSize=8 -->
00000000
<!-- RpEnd -->
'</TD>
<P>
</TR>
<TR><TD>Subnet Mask:</TD>
<TD>
<!-- RpFormInput TYPE="TEXT" NAME="SubnetMask" SIZE="15" MAXLENGTH="15"
            RpGetType=Function RpGetPtr=GetSubnetMask
            RpSetType=Function RpSetPtr=SetSubnetMask RpTextType="DotForm"
-->
<INPUT TYPE="TEXT" NAME="SubnetMask" SIZE="15" MAXLENGTH="15"
VALUE="0.0.0.0">
<!-- RpEnd -->
</TD>
<TD>X'
<!-- RpDisplayText RpGetType=Function RpGetPtr=GetSubnetMask
            RpTextType=Hex RpSize=8 -->00000000
<!-- RpEnd -->'</TD>
<P>
</TR>
<TR><TD>Default Gateway:</TD>
<TD>

<!-- RpFormInput TYPE="TEXT" NAME="DefaultGateway" SIZE="15"
MAXLENGTH="15"
            RpGetType=Complex RpGetPtr=GetDefaultGateway
            RpSetType=Complex RpSetPtr=SetDefaultGateway
RpTextType="DotForm" -->
<INPUT TYPE="TEXT" NAME="DefaultGateway" SIZE="15" MAXLENGTH="15"
VALUE="0.0.0.0">
<!-- RpEnd -->
</TD>
<TD>
<!-- RpDisplayText RpGetType=Complex RpGetPtr=GetDefaultGateway
RpTextType=HexColonForm
RpSize=11 -->
00:00:00:00
<!-- RpEnd --></TD>
```

```
<P>
</TR>
</TABLE>
<P>
</BLOCKQUOTE>
<BLOCKQUOTE>
<BLOCKQUOTE>
<BLOCKQUOTE>
<!-- RpFormInput TYPE="RESET" VALUE="Reset" -->
<INPUT TYPE="RESET" VALUE="Reset">
<!-- RpEnd -->
    
<!-- RpFormInput TYPE="SUBMIT" NAME="Submit" VALUE="Configure" -->
<INPUT TYPE="SUBMIT" NAME="Submit" VALUE="Configure">
<!-- RpEnd -->
</BLOCKQUOTE>
</BLOCKQUOTE>
</BLOCKQUOTE>
<HR>
<P>
<CENTER><IMG SRC="/Images/Main" WIDTH=160 HEIGHT=80></CENTER>
</FORM>
</BODY>
</HTML>
```

C source file generated by PBuilder

```
#include "RpExtern.h"

extern rpObjectDescription PgConfigureNetwork;

static char PgConfigureNetwork_Item_1[] =
          C_oHTML_oHEAD_oTITLE C_S_RomPager " Configure Network Info"
          C_xTITLE_xHEAD_oBODY C_oP C_oH2 C_oCENTER "Configure Network
Address"
          C_xCENTER
          C_xH2;

static char PgConfigureNetwork_Item_3[] =
```

```
            C_oP C_oHR "Return to:" C_NBSP C_oANCHOR_HREF "/Menu\">The "
            C_S_RomPager " Menu" C_xANCHOR C_NBSP "or" C_NBSP "\n"
            C_oANCHOR_HREF "/Main\">The " C_S_RomPager " Main Page"
C_xANCHOR
            C_oHR C_oP "Variables in a form can be entered and displayed in
dotted "
            "decimal\n"
            "format and hex format, as well as other numeric formats. This
page is\n"
            "formatted using\n"
            "table commands, and will look best with Netscape 1.1 or other
browsers\n"
            "that support tables.\n"
            C_oBLOCKQUOTE C_oBR C_oTABLE "<CAPTION>" C_oB "TCP/IP Address "
            "Information" C_xB "</CAPTION>" C_oTR C_oTD "IP Address:" C_xTD
            C_oTD;

extern char *theIpAddress;

static rpTextFormItem PgConfigureNetwork_Item_4 = {
            "IpAddress",
            &theIpAddress,
            &theIpAddress,
            eRpVarType_Direct,
            eRpVarType_Direct,
            eRpTextType_DotForm,
            15,
            15
};
static char PgConfigureNetwork_Item_5[] =
            C_xTD C_oTD "X\'\n";

extern char *theIpAddress;

static rpTextDisplayItem PgConfigureNetwork_Item_6 = {
            &theIpAddress,
            eRpVarType_Direct,
            eRpTextType_Hex,
            8
};
```

```
static char PgConfigureNetwork_Item_7[] =
          "\'" C_xTD_oP_xTR C_oTR C_oTD "Subnet Mask:" C_xTD
          C_oTD;

extern char *GetSubnetMask(void);
extern void SetSubnetMask(char *theValuePtr);

static rpTextFormItem PgConfigureNetwork_Item_8 = {
          "SubnetMask",
          GetSubnetMask,
          SetSubnetMask,
          eRpVarType_Function,
          eRpVarType_Function,
          eRpTextType_DotForm,
          15,
          15
};
static char PgConfigureNetwork_Item_9[] =
          C_xTD C_oTD "X\'\n";

extern char *GetSubnetMask(void);

static rpTextDisplayItem PgConfigureNetwork_Item_10 = {
          GetSubnetMask,
          eRpVarType_Function,
          eRpTextType_Hex,
          8
};
static char PgConfigureNetwork_Item_11[] =
          "\'" C_xTD_oP_xTR C_oTR C_oTD "Default Gateway:" C_xTD
          C_oTD;

extern char *GetDefaultGateway(void *theServerDataPtr, char *theNamePtr,
                               Signed16Ptr theIndexValuesPtr);
extern void SetDefaultGateway(void *theServerDataPtr, char *theValuePtr,
                               char *theNamePtr, Signed16Ptr
theIndexValuesPtr);

static rpTextFormItem PgConfigureNetwork_Item_12 = {
```

```
            "DefaultGateway",
            GetDefaultGateway,
            SetDefaultGateway,
            eRpVarType_Complex,
            eRpVarType_Complex,
            eRpTextType_DotForm,
            15,
            15
};
static char PgConfigureNetwork_Item_13[] =
            C_xTD
            C_oTD;

extern char *GetDefaultGateway(void *theServerDataPtr, char *theNamePtr,
                               Signed16Ptr theIndexValuesPtr);

static rpTextDisplayItem PgConfigureNetwork_Item_14 = {
            GetDefaultGateway,
            eRpVarType_Complex,
            eRpTextType_HexColonForm,
            11
};
static char PgConfigureNetwork_Item_15[] =
            C_xTD_oP_xTR_xTABLE C_oP C_xBLOCKQUOTE C_oBLOCKQUOTE
C_oBLOCKQUOTE
            C_oBLOCKQUOTE;

static rpButtonFormItem PgConfigureNetwork_Item_16 = {
            "Reset"
};
static char PgConfigureNetwork_Item_17[] =
            C_S_NBSP4 "\n";

static rpButtonFormItem PgConfigureNetwork_Item_18 = {
            "Configure"
};

static char PgConfigureNetwork_Item_19[] =
            C_xBLOCKQUOTE C_xBLOCKQUOTE C_xBLOCKQUOTE C_S_AllegroLogo
C_xFORM
```

```
            C_xBODY_xHTML;

static rpItem PgConfigureNetwork_FormItems[] = {
            { eRpItemType_FormAsciiText, &PgConfigureNetwork_Item_4 },
            { eRpItemType_FormAsciiText, &PgConfigureNetwork_Item_8 },
            { eRpItemType_FormAsciiText, &PgConfigureNetwork_Item_12 },
            { eRpItemType_FormReset, &PgConfigureNetwork_Item_16 },
            { eRpItemType_FormSubmit, &PgConfigureNetwork_Item_18 },
            { eRpItemType_LastItemInList }
};

static rpObjectExtension PgConfigureNetwork_FormObjectExtension = {
            (rpProcessDataFuncPtr) 0,
            &PgConfigureNetwork,
            (rpObjectDescriptionPtr) 0,
            0,
            kRpObjFlags_None
};

rpObjectDescription PgConfigureNetwork_Form = {
            "/Forms/NetworkInfo",
            PgConfigureNetwork_FormItems,
            &PgConfigureNetwork_FormObjectExtension,
            (Unsigned32) 0,
            kRpPageAccess_Unprotected,
            eRpDataTypeForm,
            eRpObjectTypeDynamic
};

static rpItem PgConfigureNetwork_Items[] = {
            { eRpItemType_DataZeroTerminated, &PgConfigureNetwork_Item_1 },
            { eRpItemType_FormHeader, &PgConfigureNetwork_Form },
            { eRpItemType_DataZeroTerminated, &PgConfigureNetwork_Item_3 },
            { eRpItemType_FormAsciiText, &PgConfigureNetwork_Item_4 },
            { eRpItemType_DataZeroTerminated, &PgConfigureNetwork_Item_5 },
            { eRpItemType_DisplayText, &PgConfigureNetwork_Item_6 },
            { eRpItemType_DataZeroTerminated, &PgConfigureNetwork_Item_7 },
            { eRpItemType_FormAsciiText, &PgConfigureNetwork_Item_8 },
            { eRpItemType_DataZeroTerminated, &PgConfigureNetwork_Item_9 },
```

```
            { eRpItemType_DisplayText, &PgConfigureNetwork_Item_10 },
            { eRpItemType_DataZeroTerminated, &PgConfigureNetwork_Item_11
},
            { eRpItemType_FormAsciiText, &PgConfigureNetwork_Item_12 },
            { eRpItemType_DataZeroTerminated, &PgConfigureNetwork_Item_13
},
            { eRpItemType_DisplayText, &PgConfigureNetwork_Item_14 },
            { eRpItemType_DataZeroTerminated, &PgConfigureNetwork_Item_15
},
            { eRpItemType_FormReset, &PgConfigureNetwork_Item_16 },
            { eRpItemType_DataZeroTerminated, &PgConfigureNetwork_Item_17
},
            { eRpItemType_FormSubmit, &PgConfigureNetwork_Item_18 },
            { eRpItemType_DataZeroTerminated, &PgConfigureNetwork_Item_19
},
            { eRpItemType_LastItemInList }
};

rpObjectDescription PgConfigureNetwork = {
            "/NetworkInfo",
            PgConfigureNetwork_Items,
            (rpObjectExtensionPtr) 0,
            (Unsigned32) 0,
            kRpPageAccess_Unprotected,
            eRpDataTypeHtml,
            eRpObjectTypeDynamic
};
```

C stub routines file generated by PBuilder

```
#include "RpExtern.h"

char *theIpAddress = " ";

extern char *GetSubnetMask(void);

extern char *GetSubnetMask(void) {
 char * theResult;

            return theResult;
```

```
}

extern void SetSubnetMask(char *theValuePtr);

extern void SetSubnetMask(char *theValuePtr) {

          return;
}
extern char *GetDefaultGateway(void *theServerDataPtr, char *theNamePtr,
                              Signed16Ptr theIndexValuesPtr);

char *GetDefaultGateway(void *theServerDataPtr, char *theNamePtr,
                              Signed16Ptr theIndexValuesPtr) {
          char * theResult;
          return theResult;
}
extern void SetDefaultGateway(void *theServerDataPtr, char *theValuePtr,
                              char *theNamePtr, Signed16Ptr
theIndexValuesPtr);

void SetDefaultGateway(void *theServerDataPtr, char *theValuePtr,
                              char *theNamePtr, Signed16Ptr
theIndexValuesPtr)
{
          return;
}
```

If the HTML file includes these comment tag, PBuilder uses structured access techniques to build the `Source` and `Stub` Routines files:

```
<!-- RpPageHeader RpUrl=/NetworkInfo RpStructuredAccess RpMaxItems=20 -->
```

C source file generated by PBuilder (structured access)

This file is the same as the source file without structured access except for the object header structures.

```
extern void RpStorePgConfigureNetwork_Form_Data(void *theServerDataPtr,
                                    Signed16Ptr theIndexValuesPtr);

static rpObjectExtension PgConfigureNetwork_FormObjectExtension = {
          RpStorePgConfigureNetwork_Form_Data,
```

```
            &PgConfigureNetwork,
            (rpObjectDescriptionPtr) 0,
            0,
            kRpObjFlags_None
};

rpObjectDescription PgConfigureNetwork_Form = {
            "/Forms/NetworkInfo",
            PgConfigureNetwork_FormItems,
            &PgConfigureNetwork_FormObjectExtension,
            (Unsigned32) 0,
            kRpPageAccess_Unprotected,
            eRpDataTypeForm,
            eRpObjectTypeDynamic
};

extern void RpFetchConfigureNetworkData(void *theServerDataPtr,
                                        Signed16Ptr theIndexValuesPtr);

static rpObjectExtension PgConfigureNetwork_ObjectExtension = {
            RpFetchConfigureNetworkData,
            (rpObjectDescriptionPtr) 0,
            (rpObjectDescriptionPtr) 0,
            0,
            kRpObjFlags_None
};

rpObjectDescription PgConfigureNetwork = {
            "/NetworkInfo",
            PgConfigureNetwork_Items,
            &PgConfigureNetwork_ObjectExtension,
            (Unsigned32) 0,
            kRpPageAccess_Unprotected,
            eRpDataTypeHtml,
            eRpObjectTypeDynamic
};
```

C stub routines file generated by PBuilder (structured access)

```
#include "RpExtern.h"

/*    Built from "ConfigureNetwork.html"    */

typedef struct {
          char *                  fIpAddress;
          char *                  fDisplay_1;
          char *                  fSubnetMask;
          char *                  fDisplay_2;
          char *                  fDefaultGateway;
          char *                  fDisplay_3;
} ConfigureNetworkData, *ConfigureNetworkDataPtr;

/*    Structured Access Form Storage routine (to be fleshed out)    */

void RpStorePgConfigureNetwork_Form_Data(void *theServerDataPtr,
                                         Signed16Ptr theIndexValuesPtr);

void RpStorePgConfigureNetwork_Form_Data(void *theServerDataPtr,
                                         Signed16Ptr theIndexValuesPtr)
{
          ConfigureNetworkDataPtrtheDataPtr;
          char *                  theEmptyCharPtr;

          theDataPtr = (ConfigureNetworkDataPtr)
RpGetRequestCookie(theServerDataPtr);
          theEmptyCharPtr = theDataPtr->fIpAddress;
          theEmptyCharPtr = theDataPtr->fSubnetMask;
          theEmptyCharPtr = theDataPtr->fDefaultGateway;
          return;
}

/*     Structured Access Page Access routine (to be fleshed out)       */

void RpFetchConfigureNetworkData(void *theServerDataPtr,
                                         Signed16Ptr theIndexValuesPtr);

void RpFetchConfigureNetworkData(void *theServerDataPtr,
```

```
                                                Signed16Ptr theIndexValuesPtr)
{
           ConfigureNetworkDataPtrtheDataPtr;

           theDataPtr = (ConfigureNetworkDataPtr)
RpGetRequestCookie(theServerDataPtr);
           theDataPtr->fIpAddress = "";
           theDataPtr->fDisplay_1 = "";
           theDataPtr->fSubnetMask = "";
           theDataPtr->fDisplay_2 = "";
           theDataPtr->fDefaultGateway = "";
           theDataPtr->fDisplay_3 = "";
return;
}

/*    Automatic Structured Access Page and Form routines    */

extern char *RpGet_IpAddress(void *theServerDataPtr, char *theNamePtr,
                             Signed16Ptr theIndexValuesPtr);

char *RpGet_IpAddress(void *theServerDataPtr, char *theNamePtr,
                             Signed16Ptr theIndexValuesPtr) {
           ConfigureNetworkDataPtrtheDataPtr;

           theDataPtr = (ConfigureNetworkDataPtr)
RpGetRequestCookie(theServerDataPtr);
           return theDataPtr->fIpAddress;
}

extern void RpSet_IpAddress(void *theServerDataPtr, char *theValuePtr,
                             char *theNamePtr, Signed16Ptr
theIndexValuesPtr);

void RpSet_IpAddress(void *theServerDataPtr, char *theValuePtr,
                             char *theNamePtr, Signed16Ptr
theIndexValuesPtr) {
           ConfigureNetworkDataPtrtheDataPtr;

           theDataPtr = (ConfigureNetworkDataPtr)
RpGetRequestCookie(theServerDataPtr);
```

```
            theDataPtr->fIpAddress = theValuePtr;
            return;
}

extern char *RpGet_Display_1(void *theServerDataPtr, char *theNamePtr,
                             Signed16Ptr theIndexValuesPtr);

char *RpGet_Display_1(void *theServerDataPtr, char *theNamePtr,
                             Signed16Ptr theIndexValuesPtr) {
            ConfigureNetworkDataPtrtheDataPtr;

            theDataPtr = (ConfigureNetworkDataPtr)
RpGetRequestCookie(theServerDataPtr);
            return theDataPtr->fDisplay_1;
}

extern char *RpGet_SubnetMask(void *theServerDataPtr, char *theNamePtr,
                             Signed16Ptr theIndexValuesPtr);

char *RpGet_SubnetMask(void *theServerDataPtr, char *theNamePtr,
                             Signed16Ptr theIndexValuesPtr) {
            ConfigureNetworkDataPtrtheDataPtr;

            theDataPtr = (ConfigureNetworkDataPtr)
RpGetRequestCookie(theServerDataPtr);
            return theDataPtr->fSubnetMask;
}

extern void RpSet_SubnetMask(void *theServerDataPtr, char *theValuePtr,
                             char *theNamePtr, Signed16Ptr
theIndexValuesPtr);

void RpSet_SubnetMask(void *theServerDataPtr, char *theValuePtr,
                             char *theNamePtr, Signed16Ptr
theIndexValuesPtr) {
            ConfigureNetworkDataPtrtheDataPtr;

            theDataPtr = (ConfigureNetworkDataPtr)
RpGetRequestCookie(theServerDataPtr);
            theDataPtr->fSubnetMask = theValuePtr;
```

```
            return;
}
extern char *RpGet_Display_2(void *theServerDataPtr, char *theNamePtr,
                              Signed16Ptr theIndexValuesPtr);
char *RpGet_Display_2(void *theServerDataPtr, char *theNamePtr,
                              Signed16Ptr theIndexValuesPtr) {
            ConfigureNetworkDataPtrtheDataPtr;

            theDataPtr = (ConfigureNetworkDataPtr)
RpGetRequestCookie(theServerDataPtr);
            return theDataPtr->fDisplay_2;
}

extern char *RpGet_DefaultGateway(void *theServerDataPtr, char
*theNamePtr,
                              Signed16Ptr theIndexValuesPtr);
char *RpGet_DefaultGateway(void *theServerDataPtr, char *theNamePtr,
                              Signed16Ptr theIndexValuesPtr) {
            ConfigureNetworkDataPtrtheDataPtr;

            theDataPtr = (ConfigureNetworkDataPtr)
RpGetRequestCookie(theServerDataPtr);
            return theDataPtr->fDefaultGateway;
}

extern void RpSet_DefaultGateway(void *theServerDataPtr, char
*theValuePtr,
                              char *theNamePtr, Signed16Ptr
theIndexValuesPtr);
void RpSet_DefaultGateway(void *theServerDataPtr, char *theValuePtr,
                              char *theNamePtr, Signed16Ptr
theIndexValuesPtr) {
            ConfigureNetworkDataPtrtheDataPtr;

            theDataPtr = (ConfigureNetworkDataPtr)
RpGetRequestCookie(theServerDataPtr);
            theDataPtr->fDefaultGateway = theValuePtr;
            return;
}

extern char *RpGet_Display_3(void *theServerDataPtr, char *theNamePtr,
```

```
                                Signed16Ptr theIndexValuesPtr);
char *RpGet_Display_3(void *theServerDataPtr, char *theNamePtr,
                                Signed16Ptr theIndexValuesPtr) {
          ConfigureNetworkDataPtrtheDataPtr;

          theDataPtr = (ConfigureNetworkDataPtr)
RpGetRequestCookie(theServerDataPtr);
          return theDataPtr->fDisplay_3;
}
```

HTML without comment tags as served by the Advanced Web Server

```
<HTML>
<HEAD>
<TITLE>RomPager Configure Network Info</TITLE>
</HEAD>
<BODY>
<P>
<H2><CENTER>Configure Network Address</CENTER>
</H2>
<FORM METHOD="POST" ACTION="/Forms/NetworkInfo">
<P>
<HR>
Return to: <A HREF="/Menu">The RomPager Menu</A>
 or <A HREF="/Main">The RomPager Main Page</A><HR>
<P>
Variables in a form can be entered and displayed in dotted decimal format
and hex format, as well as other numeric formats. This page is formatted
using table commands, and will look best with Netscape 1.1 or other
browsers that support tables.
<BLOCKQUOTE>
<BR>
<TABLE><CAPTION><B>TCP/IP Address Information</B>
</CAPTION><TR><TD>IP Address:</TD>
<TD><INPUT TYPE="TEXT" NAME="gIpAddress" SIZE="15"
MAXLENGTH="15" VALUE="0.0.0.0">
</TD>
<TD>X'00000000'</TD>
<P>
</TR>
```

```
<TR><TD>Subnet Mask:</TD>
<TD><INPUT TYPE="TEXT" NAME="gSubnetMask" SIZE="15"
MAXLENGTH="15" VALUE="0.0.0.0">
</TD>
<TD>X'00000000'</TD>
<P>
</TR>
<TR><TD>Default Gateway:</TD>
<TD><INPUT TYPE="TEXT" NAME="gDefaultGateway" SIZE="15"
MAXLENGTH="15" VALUE="0.0.0.0">
</TD>
<TD>00:00:00:00</TD>
<P>
</TR>
</TABLE>
<P>
</BLOCKQUOTE>
<BLOCKQUOTE>
<BLOCKQUOTE>
<BLOCKQUOTE>
<INPUT TYPE="RESET" VALUE="Reset">
    
<INPUT TYPE="SUBMIT" NAME="Submit" VALUE="Configure">
</BLOCKQUOTE>
</BLOCKQUOTE>
</BLOCKQUOTE>
<HR>
<P>
<CENTER><IMG SRC="/Images/Main" WIDTH=160 HEIGHT=80></CENTER>
</FORM>
</BODY>
</HTML>
```

Example 3: Printer status page

HTML with comment tags

```
<HTML>
<HEAD>
<TITLE>RomPager Printer Status</TITLE>
</HEAD>
<BODY>
<P>
<H2><CENTER>RomPager Printer Status</CENTER>
</H2>
<P>
<HR>
Return to: <A HREF="/Menu">The RomPager Menu</A> or <A
HREF="/Main">The RomPager Main Page</A><HR>
<P>
This page is a simulation of a print job queue with a variable number of
entries. The entries are displayed using table commands and can best be
seen with Netscape or other browsers that support tables. The job list will
change each time the page is displayed. In Netscape, use the Reload button
to redisplay the page.<CENTER>
<!-- RpDynamicDisplay RpGetType=function RpGetPtr=gTestPrintJobs
RpItemCount=2 -->
          <!-- RpHiddenDataZeroTerminated RpText="<P> </P>There are
no jobs in the print queue<P> </P>" -->
          <!-- RpItemGroup -->
                           <!-- RpDataZeroTerminated -->
                           <TABLE CELLPADDING=5>
                               <TR>
                                   <TH ALIGN=left>Job Name</TH>
                                   <TH ALIGN=left>Status</TH>
                                   <TH ALIGN=left>Owner</TH>
                                   <TH># Pages</TH>
                                   <P>
                               </TR>
                           <!-- RpEnd -->
                           <!-- RpRepeatGroupDynamic
RpFunctionPtr=gGetPrinterStatusLimits -->
                               <!-- RpDZT --><TR><TD><!-- RpEnd -->
```

```
                                    <!-- RpDisplayText RpGetType=complex
RpGetPtr=gGetPrintJobName -->
                                        Resume
                                    <!-- RpEnd -->
                                    <!-- RpDataZeroTerminated
RpIdentifier=gRpEndCellStartCell -->
                                        </TD><TD><!-- RpEnd -->
                                    <!-- RpDisplayText RpGetType=complex
RpGetPtr=gGetPrintJobStatus -->
                                        Printing
                                    <!-- RpEnd -->
                                    <!-- RpUseIdentifier
RpIdentifier=gRpEndCellStartCell
                                            RpItemType=DataZeroTerminated -
->
                                        </TD><TD>
                                    <!-- RpEnd -->
                                    <!-- RpDisplayText RpGetType=complex
RpGetPtr=gGetPrintJobOwner -->
                                        CEO
                                    <!-- RpEnd -->
                                    <!-- RpDZT --></TD><TD ALIGN=center><!--
RpEnd -->
                                    <!-- RpDisplayText RpGetType=complex
RpGetPtr=gGetPrintJobPages
                                            RpTextType=Unsigned8 -->
                                        2
                                    <!-- RpEnd -->
                                    <!-- RpDZT --></TD><P></TR><!-- RpEnd -->
                                <!-- RpLastItemInGroup -->
                                <!-- RpSample -->
                                    <TR><TD>Monthly Status Report</TD>
                                    <TD>Waiting</TD>
                                    <TD>COO</TD>
                                    <TD ALIGN=center>5</TD>
                                    <P>
                                    </TR>
                                    <TR><TD>Party Announcement</TD>
                                    <TD>Held</TD>
                                    <TD>CEO</TD>
                                    <TD ALIGN=center>1</TD>
```

```
                                        <P>
                                        </TR>
                                    <!-- RpEnd -->
                                    <!-- RpDZT --></TABLE><!-- RpEnd -->
                    <!-- RpLastItemInGroup -->
<!-- RpLastItemInGroup -->
</CENTER>
<HR>
<P>
<CENTER><IMG SRC="/Images/Main" WIDTH=160 HEIGHT=80></CENTER>
</BODY>
</HTML>
```

C source file generated by PBuilder

```
#include "RpExtern.h"
extern rpObjectDescription PgPrinter_Status;
static char PgPrinter_Status_Item_1[] =
          C_oHTML_oHEAD_oTITLE C_S_RomPager " Printer Status"
          C_xTITLE_xHEAD_oBODY C_oP C_oH2 C_oCENTER C_S_RomPager " Printer
Status"
          C_xCENTER C_xH2 C_oP C_oHR "Return to:" C_NBSP C_oANCHOR_HREF "/
Menu\">"
          "The " C_S_RomPager " Menu" C_xANCHOR C_NBSP "or" C_NBSP
          C_oANCHOR_HREF "/Main\">The " C_S_RomPager " Main Page"
C_xANCHOR
          C_oHR C_oP "This page is a simulation of a print job queue with
a "
          "variable number of\n"
          "entries. The entries are displayed using table commands and can
best be\n"
          "seen with Netscape or other browsers that support tables. The
job list\n"
          "will change each time the page is displayed. In Netscape, use
the\n"
          "Reload button to redisplay the page." C_oCENTER "\n";

static char PgPrinter_Status_Item_3[] =
          C_oP_NBSP_xP "There are no jobs in the print queue"
          C_oP_NBSP_xP;
```

```
static char PgPrinter_Status_Item_5[] =
          C_oTABLE_CELLPADDING "5>\n"
          C_oTR "\n"
          C_oTH_ALIGN_LEFT ">Job Name" C_xTH C_oTH_ALIGN_LEFT ">Status"
C_xTH
          C_oTH_ALIGN_LEFT ">Owner" C_xTH C_oTH "# Pages" C_xTH C_oP
C_xTR;

static char PgPrinter_Status_Item_7[] =
          C_oTR
          C_oTD;

extern char *gGetPrintJobName(void *theServerDataPtr, char *theNamePtr,
                              Signed16Ptr theIndexValuesPtr);

static rpTextDisplayItem PgPrinter_Status_Item_8[] = {
          gGetPrintJobName,
          eRpVarType_Complex,
          eRpTextType_ASCII,
          20
};

extern char gRpEndCellStartCell[] =
          C_xTD
          C_oTD;

extern char *gGetPrintJobStatus(void *theServerDataPtr, char *theNamePtr,
                              Signed16Ptr theIndexValuesPtr);
static rpTextDisplayItem PgPrinter_Status_Item_10[] = {
          gGetPrintJobStatus,
          eRpVarType_Complex,
          eRpTextType_ASCII,
          20
};
extern char gRpEndCellStartCell[];
extern char *gGetPrintJobOwner(void *theServerDataPtr, char *theNamePtr,
                              Signed16Ptr theIndexValuesPtr);

static rpTextDisplayItem PgPrinter_Status_Item_12[] = {
          gGetPrintJobOwner,
```

```
            eRpVarType_Complex,
            eRpTextType_ASCII,
            20
};

static char PgPrinter_Status_Item_13[] =
            C_xTD C_oTD_ALIGN_CENTER ">";

extern Unsigned8 gGetPrintJobPages(void *theServerDataPtr, char
*theNamePtr,
                              Signed16Ptr theIndexValuesPtr);

static rpTextDisplayItem PgPrinter_Status_Item_14[] = {
            gGetPrintJobPages,
            eRpVarType_Complex,
            eRpTextType_Unsigned8,
            20
};

static char PgPrinter_Status_Item_15[] =
            C_xTD_oP_xTR;

rpItem PgPrinter_Status_Item_6_Group[] = {
            { eRpItemType_DataZeroTerminated, PgPrinter_Status_Item_7 },
            { eRpItemType_DisplayText, PgPrinter_Status_Item_8 },
            { eRpItemType_DataZeroTerminated, gRpEndCellStartCell },
            { eRpItemType_DisplayText, PgPrinter_Status_Item_10 },
            { eRpItemType_DataZeroTerminated, gRpEndCellStartCell },
            { eRpItemType_DisplayText, PgPrinter_Status_Item_12 },
            { eRpItemType_DataZeroTerminated, PgPrinter_Status_Item_13 },
            { eRpItemType_DisplayText, PgPrinter_Status_Item_14 },
            { eRpItemType_DataZeroTerminated, PgPrinter_Status_Item_15 },
            { eRpItemType_LastItemInList }
};
extern void gGetPrinterStatusLimits(void *theServerDataPtr, Signed16Ptr
theStart,
                              Signed16Ptr theLimit, Signed16Ptr
theIncrement);

extern rpRepeatGroupDynItem PgPrinter_Status_Item_6[] = {
```

```
            gGetPrinterStatusLimits,
            PgPrinter_Status_Item_6_Group
};

static char PgPrinter_Status_Item_16[] =
            C_xTABLE;

rpItem PgPrinter_Status_Item_4[] = {
            { eRpItemType_DataZeroTerminated, PgPrinter_Status_Item_5 },
            { eRpItemType_RepeatGroupDynamic, PgPrinter_Status_Item_6 },
            { eRpItemType_DataZeroTerminated, PgPrinter_Status_Item_16 },
            { eRpItemType_LastItemInList }
};

rpItem PgPrinter_Status_Item_2_Group[] = {
            { eRpItemType_DataZeroTerminated, PgPrinter_Status_Item_3 },
            { eRpItemType_ItemGroup, PgPrinter_Status_Item_4 },
            { eRpItemType_LastItemInList }
};

extern Unsigned8 gTestPrintJobs(void);

extern rpDynamicDisplayItem PgPrinter_Status_Item_2[] = {
            gTestPrintJobs,
            eRpVarType_Function,
            2,
            PgPrinter_Status_Item_2_Group
};

static char PgPrinter_Status_Item_17[] =
            C_xCENTER C_S_AllegroLogo
            C_xBODY_xHTML;

static rpItem PgPrinter_Status_Items[] = {
            { eRpItemType_DataZeroTerminated, PgPrinter_Status_Item_1 },
            { eRpItemType_DynamicDisplay, PgPrinter_Status_Item_2 },
            { eRpItemType_DataZeroTerminated, PgPrinter_Status_Item_17 },
            { eRpItemType_LastItemInList }
};
```

```
rpObjectDescription PgPrinter_Status = {
          "/Printer_Status",
          PgPrinter_Status_Items,
          (rpObjectExtensionPtr) 0,
          (Unsigned32) 0,
          kRpPageAccess_Unprotected,
          eRpDataTypeHtml,
          eRpObjectTypeDynamic
};
```

C stub routines file generated by PBuilder

```
#include "RpExtern.h"

extern char *gGetPrintJobName(void *theServerDataPtr, char *theNamePtr,
                              Signed16Ptr theIndexValuesPtr);

char *gGetPrintJobName(void *theServerDataPtr, char *theNamePtr,
                              Signed16Ptr theIndexValuesPtr) {
          char * theResult;
          return theResult;
}

extern char *gGetPrintJobStatus(void *theServerDataPtr, char *theNamePtr,
                              Signed16Ptr theIndexValuesPtr);
char *gGetPrintJobStatus(void *theServerDataPtr, char *theNamePtr,
                              Signed16Ptr theIndexValuesPtr) {
          char * theResult;
          return theResult;
}
extern char *gGetPrintJobOwner(void *theServerDataPtr, char *theNamePtr,
                              Signed16Ptr theIndexValuesPtr);

char *gGetPrintJobOwner(void *theServerDataPtr, char *theNamePtr,
                              Signed16Ptr theIndexValuesPtr) {
          char * theResult;

          return theResult;
```

```
}

extern Unsigned8 gGetPrintJobPages(void *theServerDataPtr, char
*theNamePtr,
                              Signed16Ptr theIndexValuesPtr);

Unsigned8 gGetPrintJobPages(void *theServerDataPtr, char *theNamePtr,
                              Signed16Ptr theIndexValuesPtr) {
           Unsigned8  theResult;

           return theResult;
}

extern void gGetPrinterStatusLimits(void *theServerDataPtr, Signed16Ptr
theStart,
                              Signed16Ptr theLimit, Signed16Ptr
theIncrement);

void gGetPrinterStatusLimits(void *theServerDataPtr, Signed16Ptr theStart,
                              Signed16Ptr theLimit, Signed16Ptr
theIncrement) {

           *theStart = 1;
           *theLimit = 10;
           *theIncrement = 1;
           return;
}
extern Unsigned8 gTestPrintJobs(void);

extern Unsigned8 gTestPrintJobs(void) {
           Unsigned8  theResult;
           return theResult;
}
```

HTML without comment tags as served by the Advanced Web Server

```
<HTML>
<HEAD>
<TITLE>RomPager Printer Status</TITLE>
</HEAD>
<BODY>
<P>
<H2><CENTER>RomPager Printer Status</CENTER>
</H2>
<P>
<HR>
Return to: <A HREF="/Menu">The RomPager Menu</A> or 
<A HREF="/Main">The RomPager Main Page</A><HR>
<P>
This page is a simulation of a print job queue with a variable number of
entries. The entries are displayed using table commands and can best be
seen with Netscape or other browsers that support tables. The job list will
change each time the page is displayed. In Netscape, use the Reload button
to redisplay the page.<CENTER>
<TABLE CELLPADDING=5>
<TR>
<TH ALIGN=left>Job Name</TH>
<TH ALIGN=left>Status</TH>
<TH ALIGN=left>Owner</TH>
<TH># Pages</TH>
<P>
</TR>
<TR><TD>Resume</TD>
<TD>Printing</TD>
<TD>CEO</TD>
<TD ALIGN=center>2</TD>
<P>
</TR>
<TR><TD>Monthly Status Report</TD>
<TD>Waiting</TD>
<TD>COO</TD>
<TD ALIGN=center>5</TD>
<P>
</TR>
<TR><TD>Party Announcement</TD>
```

```
<TD>Held</TD>
<TD>CEO</TD>
<TD ALIGN=center>1</TD>
<P>
</TR>
</TABLE>
</CENTER>
<HR>
<P>
<CENTER><IMG SRC="/Images/Main" WIDTH=160 HEIGHT=80></CENTER>
</BODY>
</HTML>
```

Index

S

T

U

V


Digi®