Investigating Exported Code

F

Very few novice (or even intermediate) programmers delve into the mysteries of export object definitions. Occasions will arise, however, when it will be necessary to be able to export an object, make a modification, and reimport the object.

Exporting Code

You export the code from within the Library painter by using either the pop-up menu on an object or the PainterBar button. This prompts you for the destination filename, to which PowerBuilder assigns a three-character file extension based on the object type:

Object Type	*File Extension*
Window	.srw
DataWindow	.srd
Menu	.srm
Structure	.srs
Application	.sra
Function	.srf
Project	.srj
Pipeline	.srp
Query	.srq
Proxy	.srx
User Object	.sru

The first eight characters of the filename are the first eight characters of the object's name, so be very careful when you are exporting objects that have similar names.

Modifications to Code

Two of the most common modifications that are made to an object when it has been exported are replacing the global string and changing the ancestor object. First, you must understand the syntax of the exported file and what it means for inherited objects.

Areas of the File

I'll illustrate the areas of the exported object using the exported code of a window with nine controls. The exported window file (see Listing F.1) provides a good basis to start with; a number of the other object types share a similar structure.

NOTE

Some of the listings in this appendix contain line numbers to provide reference points to the text and are not created or generated as part of the export process.

Listing F.1. An export of a window with nine controls.

```
$PBExportHeader$w_error.srw
forward
global type w_error from Window
end type
type sle_error_number from singlelineedit within w_error
end type
type cb_print_report from u_cb within w_error
end type
type cb_quit from u_cb within w_error
end type
type sle_object_event from singlelineedit within w_error
end type
type sle_object_name from singlelineedit within w_error
end type
type st_12 from statictext within w_error
end type
type st_11 from statictext within w_error
end type
type mle_error from multilineedit within w_error
end type
end forward

global type w_error from Window
int X=595
int Y=485
int Width=2460
int Height=1425
boolean TitleBar=true
string Title="Application Error"
long BackColor=12632256
WindowType WindowType=response!
ToolBarAlignment ToolBarAlignment=AlignAtLeft!
event playsound pbm_custom01
event type integer operation ( integer a_nparm1,  string a_szparm1 )
sle_error_number sle_error_number
cb_print_report cb_print_report
cb_quit cb_quit
sle_object_event sle_object_event
sle_object_name sle_object_name
st_12 st_12
st_11 st_11
mle_error mle_error
end type
global w_error w_error

type ws_error from structure
    String szError
    Integer nError
end type

shared variables
Integer sh_nCount
end variables

type variables
Integer i_nCount
end variables
```

continues

Listing F.1. continued

```

type prototypes
Function uInt FindWindow( string szClass, string SZName) Library
 "user.exe"
end prototypes

forward prototypes
public subroutine wf_notify ()
end prototypes

public subroutine wf_notify ();MailSession  PBmailSession
// Code removed for brevity
end subroutine

event open;g_App.uf_CentreWindow( this)
// Code removed for brevity
end event

on playsound;If FileExists("error.wav") Then
      g_App.Externals.uf_PlaySound("error.wav", 0)
End If
end on

on w_error.create
this.sle_error_number=create sle_error_number
this.cb_print_report=create cb_print_report
this.cb_quit=create cb_quit
this.sle_object_event=create sle_object_event
this.sle_object_name=create sle_object_name
this.st_12=create st_12
this.st_11=create st_11
this.mle_error=create mle_error
this.Control[]={ this.sle_error_number,&
this.cb_print_report,&
this.cb_quit,&
this.sle_object_event,&
this.sle_object_name,&
this.st_12,&
this.st_11,&
this.mle_error}
end on

on w_error.destroy
destroy(this.sle_error_number)
destroy(this.cb_print_report)
destroy(this.cb_quit)
destroy(this.sle_object_event)
destroy(this.sle_object_name)
destroy(this.st_12)
destroy(this.st_11)
destroy(this.mle_error)
end on

type sle_error_number from singlelineedit within w_error
int X=449
int Y=493
```

```
113:    int Width=311
114:    int Height=89
115:    boolean Border=false
116:    boolean AutoHScroll=false
117:    boolean HideSelection=false
118:    boolean DisplayOnly=true
119:    string Text="<Unknown>"
120:    long TextColor=255
121:    long BackColor=12632256
122:    int TextSize=-8
123:    int Weight=400
124:    string FaceName="Arial"
125:    FontFamily FontFamily=Swiss!
126:    FontPitch FontPitch=Variable!
127:    end type
128:
129:    type cb_quit from u_cb within w_error
130:    int X=1797
131:    int Y=225
132:    int Width=572
133:    int TabOrder=10
134:    string Text="&Quit the Application"
135:    end type
136:
137:    on clicked;call u_cb::clicked;If FileExists("rusure.wav") Then
138:       g_App.Externals.uf_PlaySound("rusure.wav", 0)
139:    End If
140:
141:    If MessageBox("Caution", "Are you sure you wish to quit?", &
        StopSign!, YesNo!) = 1 Then
142:       Halt Close
143:    End If
144:    end on
```

Lines 1 through 21 contain the export header declaration. There is no need to ever touch line 1. The remaining lines are declarations of all the controls used in the window. Line 3 is the type declaration for the actual window; in this listing, it is of type `Window`. Note that the window is a *global* type declaration. This is used at runtime to direct PowerBuilder to declare a global variable to point at this window. The remaining controls are declared from either PowerBuilder standard control types (`singlelineedit`, `statictext`, and `multilineedit`) or from user objects (in this case, just `u_cb`).

Lines 23 through 44 are the declarations of the window's attributes; only those attributes that have been assigned values will be listed. Notice that user events and window controls are listed as attributes of the window. Also new to PowerBuilder 5.0 are lines 32 and 34. Line 32 defines the new toolbar attribute available for windows. Line 34 shows the declaration of a user-defined parameterized event that takes two parameters.

Lines 46 through 57 show window structures and shared and instance variables that have been defined for the window.

Lines 59 through 69 are the prototypes for window-level functions, and local external functions are declared next. The actual code for the functions appears after these prototypes.

Lines 71 through 78 detail window events that have script associated with them (at this level and not at the ancestor level). They can take one of two forms. User-defined events that use the `pbm_custom` message identifier take the following form:

```
on EventName;
Event PowerScript
end on
```

Normal object events, parameterized events, and events that you specify without parameters or a message ID follow this format:

```
Event EventName;
Event PowerScript
End Event
```

Lines 80 through 108 show that in the window's event section are two very important events: `on create` and `on destroy`. The first event is where each object is instantiated when the window is created at runtime. Notice that the objects are instantiated into the variables defined in lines 23 through 44 and are added into the control array attribute. During the window destruction, all the controls are destroyed as well.

Lines 110 through 144 of an export file contain the definitions, attributes, and event code for each control. This has been shortened because the other objects repeat the same kind of information.

The layout of the export file varies between the different types of objects, as you will soon see. User objects are the exception and are identical to windows, except in the global-type definition, where they are naturally inherited from `UserObject` rather than `Window`. Of course, this doesn't apply if they are inherited, as you will see later in the section "Object Inheritance," which uses a window as an example.

Application Objects

Application objects (see Listing F.2) naturally contain the definitions of global variables and the global variable data types (such as `Message` and `Error`).

Listing F.2. An export of an application object.

```
$PBExportHeader$oe_010.sra
forward
global u_n_transaction sqlca
global dynamicdescriptionarea sqlda
global dynamicstagingarea sqlsa
global error error
global message message
end forward
```

```
global variables
u_n_application g_App
Boolean g_bOrderWriter, g_bOrderWriterSupervisor, g_bOfficeService
Boolean g_bServiceRep, g_bDeveloper
Boolean g_bLabeling, g_bSaveStatus

// CONSTANTS
Constant HOLD_STATUS = "H"
Constant OPEN_STATUS = "O"
Constant CLOSED_STATUS = "C"
end variables

shared variables
end variables

global type oe_010 from application
end type
global oe_010 oe_010

type prototypes
end prototypes

on open;//Code removed for brevity
end on

on systemerror;open( w_error)
end on

on oe_010.create
appname = "oe_010"
sqlca = create u_n_transaction
sqlda = create dynamicdescriptionarea
sqlsa = create dynamicstagingarea
error = create error
message = create message
end on

on oe_010.destroy
destroy( sqlca )
destroy( sqlda )
destroy( sqlsa )
destroy( error )
destroy( message )
end on
```

Note that PowerBuilder creates and destroys the default global variable data types for you.

Functions

Functions are very easy to understand, but you will rarely export one because you have access to everything about the object in the Function painter. As you can see, the example in Listing F.3 follows very much the same structure as a window, except (of course) it does not contain any controls or the code to declare, create, and destroy them.

Listing F.3. An export of a function.

```
$PBExportHeader$f_boolean_to_number.srf
global type f_boolean_to_number from function_object
end type

forward prototypes
global function integer f_boolean_to_number (boolean bvalue)
end prototypes

global function integer f_boolean_to_number (boolean bvalue);Integer nReturn
// Code removed for brevity
end function
```

Structures

Structures (see Listing F.4) are even simpler than functions. Again, you will only modify the structure object from within the Structure painter.

Listing F.4. An export of a structure.

```
$PBExportHeader$s_outline.srs
global type s_outline from structure
string szentrytext
int nlevel
int nchildren
int nparentindex
string szexpanded
end type
```

Menus

Menus are a little more involved, but they are still similar to windows. As you can see from Listing F.5, when a menu is created at runtime, each individual menu item is also created; this causes the performance degradation hinted at in areas of this book. This should be obvious from the number of `Create` and `Destroy` events in the export in Listing F.5.

Listing F.5. An export of a menu.

```
$PBExportHeader$m_frame.srm
forward
global type m_frame from menu
end type
type m_file from menucascade within m_frame
end type
type m_newmfgorder from menu within m_file
end type
type m_2 from menu within m_file
```

```
end type
type m_exit from menu within m_file
end type
type m_file from menucascade within m_frame
m_newmfgorder m_newmfgorder
m_2 m_2
m_exit m_exit
end type
type m_help from menucascade within m_frame
end type
type m_about from menu within m_help
end type
type m_help from menucascade within m_frame
m_about m_about
end type
end forward

global type m_frame from menu
m_file m_file
m_help m_help
end type
global m_frame m_frame

type variables
Integer       i_nWindowListPosition = 6
String        i_szMRU1, i_szMRU2, i_szMRU3
end variables

forward prototypes
public subroutine mf_setmaintenancepermissions ()
public subroutine mf_openmru (string a_szorderno, integer a_nposition)
end prototypes

public subroutine mf_setmaintenancepermissions ();
//Code removed for brevity
end subroutine

public subroutine mf_openmru (string a_szorderno, integer a_nposition);
//Code removed for brevity
end subroutine

on m_frame.create
m_frame=this
this.m_file=create m_file
this.m_help=create m_help
this.Item[]={this.m_file, &
this.m_help}
end on

on m_frame.destroy
destroy(this.m_file)
destroy(this.m_help)
end on

type m_file from menucascade within m_frame
m_newmfgorder m_newmfgorder
m_2 m_2
m_exit m_exit
end type
```

continues

Listing F.5. continued

```
on clicked;//Code removed for brevity
end on

on m_file.create
this.Text="&File"
this.m_newmfgorder=create m_newmfgorder
this.m_2=create m_2
this.m_exit=create m_exit
this.Item[]={this.m_newmfgorder, &
this.m_2, &
this.m_exit}
end on

on m_file.destroy
destroy(this.m_newmfgorder)
destroy(this.m_2)
destroy(this.m_exit)
end on

type m_newmfgorder from menu within m_file
end type

on clicked;//Code removed for brevity
end on

on m_newmfgorder.create
this.Text="&New Mfg. Order~tCtrl+N"
this.ToolBarItemName="q:\projects\oe\oe_010\mfgsheet.bmp"
this.ToolBarItemText="New Mfg. Sheet"
this.Enabled=false
this.Shortcut=334
end on

type m_2 from menu within m_file
end type

on m_2.create
this.Text="-"
end on

type m_exit from menu within m_file
end type

on clicked;//Code removed for brevity
end on

on m_exit.create
this.Text="E&xit~tCtrl+X"
this.Microhelp="Leave application"
this.Shortcut=344
end on

type m_help from menucascade within m_frame
m_about m_about
end type
```

```
on m_help.create
this.Text="&Help"
this.m_about=create m_about
this.Item[]={this.m_about}
end on

on m_help.destroy
destroy(this.m_about)
end on

type m_about from menu within m_help
end type

on clicked;open(w_about)
end on

on m_about.create
this.Text="&About..."
this.Microhelp="About the application"
end on
```

The export has changed in a very subtle way for menus in PowerBuilder 5.0. Menu titles are now declared of type `MenuCascade` instead of type `Menu` as they were before and as all menu items still are.

DataWindows

DataWindows (see Listing F.6) are very different in their exported format from any of the objects discussed to this point. You might expect this from the range of special commands used with them within your PowerScript.

Listing F.6. An export of a DataWindow.

```
$PBExportHeader$d_order_types.srd
release 5;
datawindow(units=0 timer_interval=0 color=12632256 processing=0
print.documentname="" print.orientation = 0 print.margin.left = 110
print.margin.right = 110 print.margin.top = 97
print.margin.bottom = 97 print.paper.source = 0 print.paper.size = 0
print.prompt=no )
header(height=93 color="536870912" )
summary(height=1 color="536870912" )
 footer(height=1 color="536870912" )
 detail(height=105 color="536870912" )
 table(column=(type=char(2) update=yes key=yes name=type
 dbname="order_type.type" )
 column=(type=char(3) update=yes key=yes name=plant_no
 dbname="order_type.plant_no" )
 column=(type=char(35) update=yes name=description
 dbname="order_type.description" )
 column=(type=number update=yes name=range_begin
```

continues

Listing F.6. continued

```
19:     dbname="order_type.range_begin" )
20:     column=(type=number update=yes name=range_end
21:     dbname="order_type.range_end" )
22:     column=(type=number update=yes name=sequence
23:     dbname="order_type.sequence" )
24:     column=(type=timestamp name=timestamp dbname="order_type.timestamp" )
25:     retrieve="PBSELECT( VERSION(400) TABLE(NAME=~"order_type~" )
26:     COLUMN(NAME=~"order_type.type~") COLUMN(NAME=~"order_type.plant_no~")
27:     COLUMN(NAME=~"order_type.description~")
28:     COLUMN(NAME=~"order_type.range_begin~")
         COLUMN(NAME=~"order_type.range_end~")
29:     COLUMN(NAME=~"order_type.sequence~")
         COLUMN(NAME=~"order_type.timestamp~")) "
30:     update="order_type" updatewhere=1 updatekeyinplace=yes )
31:     column(band=detail id=3 alignment="0" tabsequence=30 border="5" color="0"
32:     x="439" y="16" height="69"
33:     width="805"  name=description  font.face="Arial" font.height="-8"
34:     font.weight="400"  font.family="2"
35:     font.pitch="2" font.charset="0" background.mode="2"
36:     background.color="12632256" )
37:     column(band=detail id=4 alignment="0" tabsequence=40 border="5" color="0"
38:     x="1317" y="16" height="69"
39:     width="316"  name=range_begin  font.face="Arial" font.height="-8"
40:     font.weight="400"  font.family="2"
41:     font.pitch="2" font.charset="0" background.mode="2"
42:     background.color="12632256" )
43:     //
44:     // Repeated for each column
45:     //
46:     text(band=header alignment="0" text="Type"border="0" color="33554432"
47:     x="69" y="16" height="57"
48:     width="110"  name=stock_no_t  font.face="Arial" font.height="-8"
49:     font.weight="400"  font.family="2"
50:     font.pitch="2" font.charset="0" background.mode="1"
51:     background.color="536870912" )
52:     text(band=header alignment="0" text="Plant No"border="0" color="33554432"
53:     x="225" y="16" height="57"
54:     width="179"  name=additional_description_t  font.face="Arial"
55:     font.height="-8" font.weight="400"
56:     font.family="2" font.pitch="2" font.charset="0" background.mode="1"
57:     background.color="536870912" )
58:     //
59:     // Repeated for each text object, and other drawing objects within the
         DataWindow
60:     //
```

Lines 1 through 7 define the specifications for printing the DataWindow, from the margins to the paper orientation.

Lines 8 through 11 state the height of the different bands and the colors for each band.

Each database column and database computed value is listed in lines 12 through 24 with its data type, update information, DataWindow name, and database name. Any database computed values will show up as `compute_000` and higher if you did not name them. That is why it is advisable to name all your database computed fields.

Lines 25 through 29 show the actual retrieval statement for the DataWindow (if it has one). As you can see, it is stored in PowerBuilder's own internal format, which makes it transportable between different databases.

Additional update information is stated in line 30. The columns to include in the update are specified in lines 12 through 24.

Lines 31 through 45 list each column placed on the DataWindow along with the settings of all of its attributes (which are accessible at runtime using the `Describe()` and `Modify()` functions).

Each static text object, along with other drawing objects, is listed with each of its settings in lines 46 through 60.

If you export a query object, you will get a file that contains lines similar to lines 25 to 29.

Projects

Exporting a project yields the same information that is available through the Object | Browsing menu item in the Project painter, because all the objects for the libraries of a project are listed. Listing F.7 shows a shortened list of objects.

Listing F.7. An export of a project.

```
$PBExportHeader$order_entry.srj
EXE:q:\projects\oe\oe_010\oe_010.exe,q:\projects\oe\oe_010\oe_010.pbr,,0,1
CMP:0,0,0,2,1,0
PBD:q:\projects\shared\sh_uobj.pbl,,1
PBD:q:\projects\shared\sh_func.pbl,q:\shared\shared.pbr,1
PBD:q:\projects\shared\sh_wind.pbl,,1
PBD:q:\projects\oe\oe_010\oe_010.pbl,,0
PBD:q:\projects\oe\oe_010\oe_main.pbl,,1
PBD:q:\projects\oe\oe_010\oe_mfg.pbl,,1
PBD:q:\projects\oe\oe_010\oe_mnt.pbl,,1
PBD:q:\projects\oe\oe_010\oe_rept.pbl,,1
PBD:q:\projects\oe\oe_010\oe_stock.pbl,,1
OBJ:q:\projects\oe\oe_010\oe_main.pbl,f_transfer_line_to_line,f
OBJ:q:\projects\shared\sh_func.pbl,f_close_all_mdi_children,f
OBJ:q:\projects\oe\oe_010\oe_mfg.pbl,d_converting_spec_entry,d
OBJ:q:\projects\oe\oe_010\oe_main.pbl,f_convert_decimal_to_fraction,f
OBJ:q:\projects\shared\sh_func.pbl,f_print_multi_lines,f
OBJ:q:\projects\shared\sh_wind.pbl,w_change_password,w
OBJ:q:\projects\oe\oe_010\oe_main.pbl,d_line_items_entry,d
OBJ:q:\projects\shared\sh_func.pbl,f_boolean_to_string,f
OBJ:q:\projects\oe\oe_010\oe_mnt.pbl,w_maintenance,w
OBJ:q:\projects\shared\sh_uobj.pbl,u_n_externals_win32,u
```

The EXE line states the filename and path of the executable that will be created, the resource file (if any), whether the Project painter should prompt for overwrite (`0 = FALSE`, `1 = TRUE`), and whether the libraries in the search path should be regenerated (`0 = FALSE`, `1 = TRUE`).

The CMP line states the compilation options chosen, and has the following format:

```
CMP: MachineCode, LineInfo, TraceInfo, ExeFormat, Optimization, OpenServer
```

`MachineCode` is `0` for PowerBuilder native code, and `1` for machine code. *`LineInfo`*, *`TraceInfo`*, and *`OpenServer`* are `0` for no and `1` for yes. *`ExeFormat`* is `0` for 16-bit, and `2` for 32-bit. *`Optimization`* is `0` for Speed, `1` for Space, and `2` for no optimization.

The PBD lines state each of the libraries in the search path, the resource file (if any), and whether the library should be made into a PBD file (`0 = FALSE`, `1 = TRUE`).

The OBJ lines list the objects that are used by the application, along with the library in which they reside. The last character is the object type, which is used in the Object Browser accessible in the Project painter.

Pipelines

The export of a pipeline object (shown in Listing F.8) shows the settings for the pipeline, a definition of the source tables and columns, the `RETRIEVE` statement to get the data from those tables and columns, and a definition of the destination table and columns.

Listing F.8. An export of a pipeline.

```
$PBExportHeader$p_emp_master_create.srp
$PBExportComments$Creates a copy of the employee table to emp_pipe_master
PIPELINE(source_connect=DemoDB -Revised C:,destination_connect=DemoDB -Revised
C:,type=replace,commit=100,errors=10,keyname="emp_pipe_master_x")
SOURCE(name="employee",COLUMN(type=long,name="emp_id",dbtype="integer",
key=yes,nulls_allowed=no)
COLUMN(type=char,name="emp_fname",dbtype="char(20)",nulls_allowed=no)
COLUMN(type=char,name="emp_lname",dbtype="char(20)",nulls_allowed=no)
COLUMN(type=long,name="dept_id",dbtype="integer",nulls_allowed=no)
COLUMN(type=char,name="bene_health_ins",dbtype="char(1)",nulls_allowed=yes)
COLUMN(type=char,name="bene_life_ins",dbtype="char(1)",nulls_allowed=yes)
COLUMN(type=char,name="bene_day_care",dbtype="char(1)",nulls_allowed=yes))
RETRIEVE(statement="PBSELECT(TABLE(NAME=~"employee~" )
COLUMN(NAME=~"employee.emp_id~") COLUMN(NAME=~"employee.emp_fname~")
COLUMN(NAME=~"employee.emp_lname~")
COLUMN(NAME=~"employee.dept_id~") COLUMN(NAME=~"employee.bene_health_ins~")
COLUMN(NAME=~"employee.bene_life_ins~")
COLUMN(NAME=~"employee.bene_day_care~"))")
DESTINATION(name="emp_pipe_master",COLUMN(type=long,name="emp_id",
dbtype="integer",key=yes,nulls_
allowed=no,initial_value="0")
COLUMN(type=char,name="emp_fname",dbtype="char(20)",nulls_allowed=no,
initial_value="spaces")
COLUMN(type=char,name="emp_lname",dbtype="char(20)",nulls_allowed=no,
initial_value="spaces")
COLUMN(type=long,name="dept_id",dbtype="integer",nulls_allowed=no,
initial_value="0")
COLUMN(type=char,name="bene_health_ins",dbtype="char(1)",nulls_allowed=yes)
COLUMN(type=char,name="bene_life_ins",dbtype="char(1)",nulls_allowed=yes)
COLUMN(type=char,name="bene_day_care",dbtype="char(1)",nulls_allowed=yes))
```

Proxies

Proxy objects are used in distributed PowerBuilder and are created from within the User Object painter. The exported code (see Listing F.9) resembles that of the user object that the proxy objects are based on, with the exception of the actual code for the methods. The methods of the proxy object are simply placeholders (also called *stubs*) for the remote objects methods.

Listing F.9. An export of a proxy object.

```
$PBExportHeader$p_simpleserver.srx
$PBExportComments$Simple proxy object created by save of u_simpleserver
forward
global type p_simpleserver from remoteobject
end type
end forward

global type p_simpleserver from remoteobject
end type
global p_simpleserver p_simpleserver

type variables

end variables

forward prototypes
public function string classname ()
public function boolean postevent (string e)
public function boolean postevent (string e,long w,long l)
public function boolean postevent (string e,long w,string l)
public function int triggerevent (string e)
public function int triggerevent (string e,long w,long l)
public function int triggerevent (string e,long w,string l)
public function int uf_times2 (int arg1)
public subroutine  remote_beep ()
end prototypes

public function string classname ();
any __aapbpm__[]
return invoke_method("classname@rsp0",0,__aapbpm__)
end function

public function boolean postevent (string e);
any __aapbpm__[1]
__aapbpm__[1]=e
return invoke_method("postevent@rtp1s",1,__aapbpm__)
end function
```

continues

Listing F.9. continued

```
public function boolean postevent (string e,long w,long l);
any __aapbpm__[3]
__aapbpm__[1]=e
__aapbpm__[2]=w
__aapbpm__[3]=l
return invoke_method("postevent@rtp3sll",3,__aapbpm__)
end function

public function boolean postevent (string e,long w,string l);
any __aapbpm__[3]
__aapbpm__[1]=e
__aapbpm__[2]=w
__aapbpm__[3]=l
return invoke_method("postevent@rtp3sls",3,__aapbpm__)
end function

public function int triggerevent (string e);
any __aapbpm__[1]
__aapbpm__[1]=e
return invoke_method("triggerevent@rip1s",1,__aapbpm__)
end function

public function int triggerevent (string e,long w,long l);
any __aapbpm__[3]
__aapbpm__[1]=e
__aapbpm__[2]=w
__aapbpm__[3]=l
return invoke_method("triggerevent@rip3sll",3,__aapbpm__)
end function

public function int triggerevent (string e,long w,string l);
any __aapbpm__[3]
__aapbpm__[1]=e
__aapbpm__[2]=w
__aapbpm__[3]=l
return invoke_method("triggerevent@rip3sls",3,__aapbpm__)
end function

public function int uf_times2 (int arg1);
any __aapbpm__[1]
__aapbpm__[1]=arg1
return invoke_method("uf_times2@rip1i",1,__aapbpm__)
end function

public subroutine  remote_beep ();
any __aapbpm__[]
invoke_method("remote_beep@rvp0",0,__aapbpm__)
end subroutine

on p_simpleserver.create
remoteobject::create_object("u_simpleserver")
end on

on p_simpleserver.destroy
remoteobject::destroy_object()
end on
```

Search and Replace

You can use the Search and Replace feature of any text editor to make global name—or even code—changes. A word of warning: Be very careful replacing small words because you might clobber other statements in a way you did not intend. Although the object might import correctly, you will either get strange runtime errors or a general protection fault when you try to open it in a painter.

There should be little or no need to change any of the PowerBuilder object-specific code unless you are making inheritance changes (as you will see in the next section).

Object Inheritance

The other most common reason for exporting an object is to make a change to the inheritance chain. By careful manipulation of the export code, you can reattach an object at any level of the inheritance chain.

For a demonstration of this technique, look at a simple window called `w_oe_error` (see Listing F.10), which is initially inherited from `w_error`. You will change the ancestor object to be `w_dialog_for_errors`.

Listing F.10. The initial export of `w_oe_error`.

```
$PBExportHeader$w_oe_error.srw
forward
global type w_oe_error from w_error
end type
end forward

global type w_oe_error from w_error
end type
global w_oe_error w_oe_error

on timer;call w_error::timer;Timer( 0)

cb_recover.TriggerEvent( Clicked!)
end on

on open;call w_error::open;Timer( 30, this)
end on

on w_oe_error.create
call w_error::create
end on

on w_oe_error.destroy
call w_error::destroy
end on
```

This is actually a simple case of text replacement from `w_error` to `w_dialog_for_errors`, which will cover all the simple code changes. However, you must be aware that some of the controls that were inherited from the original ancestor might not be available in the new ancestor, and you should remove all references to these. Otherwise, you will be unable to import the modified window into PowerBuilder.

Importing Code

You import the code back into PowerBuilder using the Library painter, either with the menu option or the PainterBar button. This prompts you for the source filename and then displays a list of the libraries in the current search path. When you select a library, the file is imported. If any errors occur, they are displayed in a dialog box, and the object will not be created in the destination library.