cancelSubscription

Method Cancels a subscription.

Syntax **cancelSubscription** (***objectKey*** String) Logical

Description Cancels the subscription specified by *objectKey*. The next time you poll, OBEX removes your name from the subscriber list in the publisher's object store. Call **poll** to do this under program control. To get valid values for *objectKey*, call **enumSubscriptions** and take values from the *Object_Key* field. This method does not delete the object from the object store; to do that, you must call **deleteObject**.

If *objectKey* does not exist in the user's object store, this method fails. This method returns True if it succeeds; otherwise, it returns False.

Example This example opens another form and uses it as a dialog box to get input. The code calls **enumSubscriptions** to create a table of the subscriptions, then uses the TableName property to bind that table to a table frame in the dialog form. Choose a subscription by moving the cursor to a record in the table frame, then click OK in the dialog form. A call to **cancelSubscription** cancels that subscription, a call to **deleteObject** removes the object from the object store, and a call to **poll** removes the subscriber's name from the list in the publisher's object store.

```
method pushButton(var eventInfo Event)
{
cancelSubscription
Assumptions:
- wgLib is a Library var declared in the form's Var window.
- wgLib is opened in the form's built-in open method.
- All workgroup methods are declared in the form's Uses window.
- The wgDlg form contains an unbound table frame named dlgTF,
  an OK button that calls formReturn(True),
  and a Cancel button that calls formReturn(False).
}
  var
    wgDlg                 Form
    msgCaption, msgText,
    dlgName, dlgTitle,
    subsTbName, objectKey,
    acctName, acctTbName      String
    dlgRetVal, inclSecnd      Logical
    tb                    Table
  endVar

  ; Initialize variables.
  dlgName = "wgDialog"
  dlgTitle = "Choose a subscription to cancel."
  msgText = "Could not open " + dlgName + "."
  subsTbName = ":PRIV:usrSubs"
  acctTbName = ":PRIV:usrAcct"
  inclSecnd = True ; Include secondary accounts.

  ; Open the dialog form using WinStyleHidden to make it invisible.
  if not wgDlg.open(dlgName, WinStyleHidden + WinStyleDefault) then
    errorShow(msgText)
    return
  endIf
```

```
  wgDlg.Title = dlgTitle  ; Set up the form.
  wgLib.enumSubscriptions(subsTbName) ; Create a table of the user's subscriptions.
  wgDlg.dlgTF.TableName = subsTbName ; Bind table frame to table of subscriptions.
  wgDlg.show(); Make the form visible.

  dlgRetVal = wgDlg.wait() ; Wait for user to make a choice.

; If user clicks the OK button, get the value of the Object_Key field in the
; record selected by the user (that is, the record containing the active field,
; the field that has focus).
  if dlgRetVal = True then
    objectKey = wgDlg.dlgTF.Object_Key.Value
    wgDlg.hide()                    ; Hide the dialog box (it's used later).
    wgLib.cancelSubscription(objectKey) ; Cancel the subscription.
    wgLib.deleteObject(objectKey)       ; Delete object from object store.
  else
    wgDlg.close() ; Close the dialog box.
    return        ; Exit the method. The user canceled the operation.
  endIf

  ; Set up the dialog box to list accounts.
  dlgTitle = "Choose an account to poll."
  wgDlg.Title = dlgTitle
  wgLib.enumAccounts(acctTbName, inclSecnd) ; Create a table of the user's accounts.
  wgDlg.dlgTF.TableName = acctTbName        ; Bind table frame to table of accounts.
  wgDlg.show()                      ; Make the form visible.

  dlgRetVal = wgDlg.wait()                ; Wait for user to make a choice.

; If user clicks the OK button, get the value of the AccountName field in the
; record selected by the user (that is, the record containing the active field,
; the field that has focus).
  if dlgRetVal = True then
    acctName = wgDlg.dlgTF.AccountName.Value
    wgLib.poll(acctName)
  endIf

  wgDlg.close()  ; Close the dialog form.

  if tb.attach(subsTbName) then
    tb.delete() ; No further need for this data.
  endIf
  if tb.attach(acctTbName) then
    tb.delete() ; No further need for this data.
  endIf
endMethod
```

See Also deleteObject
enumSubscriptions

deleteObject

Method Deletes an object from the user's object store.

Syntax **deleteObject** (***objectKey*** String) Logical

Description Deletes an object from the user's object store without asking for confirmation. Objects can be publications, subscriptions, or alerts. *objectKey* specifies which object to delete. To get valid values for *objectKey*, call **enumAlerts**, **enumPublications**, or **enumSubscriptions** and take values from the *Object_Key* field.

When you use **deleteObject** to delete a subscription, you will continue to receive new versions of the object unless you call **cancelSubscription** first.

If *objectKey* does not exist in the user's object store, this method fails. This method returns True if it succeeds; otherwise, it returns False.

Example This example opens another form and uses it as a dialog box to get input. The code creates a table listing the objects (alerts, publications, or subscriptions) you specify by choosing from a drop-down edit list. Then it uses the TableName property to bind that table to a table frame in the dialog form. You choose an object by moving the cursor to a record in the table frame, then click the OK button in the dialog form to delete the object.

```
method pushButton(var eventInfo Event)
{
deleteObject
Assumptions:
- wgLib is a Library var declared in the form's Var window.
- wgLib is opened in the form's built-in open method.
- All workgroup methods are declared in the form's Uses window.
- This form contains a drop-down edit list named objTypeList
  and a button that executes the following code.
- The form wgDlg contains an unbound table frame named dlgTF,
  an OK button that calls formReturn(True) and a
  Cancel button that calls formReturn(False).
}
  var
    wgDlg                                Form
    objType, objectKey, msgCaption,
    msgText, dlgName, dlgTitle,
    tbName, alertsTbName, pubsTbName, subsTbName  String
    dlgRetVal, isSubscription                Logical
  endVar

  ; Initialize variables.
  dlgName       = "wgDialog"
  dlgTitle      = "Choose an object to delete."
  msgCaption    = "Object type required:"
  msgText       = "Choose an object type from the Object Types list."
  alertsTbName  = ":PRIV:usrAlert"
  pubsTbName    = ":PRIV:usrPubs"
  subsTbName    = ":PRIV:usrSubs"
  objType       = objTypeList.Value
  isSubscription = No

  ; Find out which type of object to delete, and call the appropriate
  ; method to create a table listing objects in the user's object store.
  switch
```

```
  case objType = "Alert" : tbName = alertsTbName
      wgLib.enumAlerts(tbName)

  case objType = "Publication" : tbName = pubsTbName
      wgLib.enumPublications(tbName)

  case objType = "Subscription" : tbName = subsTbName
      wgLib.enumSubscriptions(tbName)
      isSubscription = Yes

  otherwise : msgInfo(msgCaption, msgText)
          return
endSwitch

; Open the dialog form, using WinStyleHidden to make it invisible.
if not wgDlg.open(dlgName, WinStyleHidden + WinStyleDefault) then
  errorShow(msgText)
  return
endIf

wgDlg.Title = dlgTitle; Set up the form.
wgDlg.dlgTF.TableName = tbName ; Bind table frame to table enum'ed above.
wgDlg.show() ; Make the form visible.
dlgRetVal = wgDlg.wait(); Wait for user to make a choice.

; If user clicks the OK button, get the value of the Object_Key field
; in the record selected by the user.
if dlgRetVal = True then
  objectKey = wgDlg.dlgTF.Object_Key.Value
  if isSubscription then
    wgLib.cancelSubscription(objectKey)
  endIf
  if wgLib.deleteObject(objectKey) then ; Delete the object
    message("Object deleted from the object store.")
  else
    wgDlg.close()
    errorShow()
    return
  endIf
endIf

wgDlg.close() ; Close the dialog form.
endMethod
```

See Also cancelSubscription

enumAccounts

Method Creates a table listing Object Exchange accounts.

Syntax **enumAccounts** (***tableName*** String, ***includeSecondary*** Logical) Logical

Description Creates a table of account information, where *tableName* is the file name of the table to create. The following table shows the structure.

Field name	Type	Description	Sample values
AccountName	A20	The name of the account	"MCI-2", "myLANacct"
PrimaryOrSecondary	A10	Is this account primary or secondary?	"Primary", "Secondary"
TransportName	A30	Which transport does this account use?	"MCI", "LAN"
IsActive	A3	Is this account active?	"Yes", "No"
IsSelected	A3	Is this account selected? Used internally by the Workgroup desktop.	"Yes", "No"

If *includeSecondary* is False, the table lists only primary accounts; if *includeSecondary* is True, the table lists primary and secondary accounts.

This method requires a full lock on *tableName*. If the table contains data, this method empties it without asking for confirmation. This method returns True if it succeeds; otherwise, it returns False.

Example This example calls **enumAccounts** to create a table named :PRIV:UsrAccts listing your accounts. Then it scans the table and assigns values to two drop-down edit lists: one lists primary accounts, the other lists secondary accounts. Each list contains a list object named *listObj*. A user could choose an item from one these lists to specify an account to poll. You could check the value of the IsActive field of the *UsrAccts* table to find out if the account is active before you try to poll it.

```
method pushButton(var eventInfo Event)
{
enumAccounts
Assumptions:
- wgLib is a Library var declared in the form's Var window.
- wgLib is opened in the form's built-in open method.
- All workgroup methods are declared in the form's Uses window.
- The form contains drop-down edit lists named primAcctList and secAcctList.
- Each list contains a list object named listObj.
}
  var
    acctTbName    String
    acctTb        Table
    acctTC        TCursor
    inclSecnd     Logical
  endVar

  ; Initialize variables.
  acctTbName  = ":PRIV:usrAcct"
  inclSecnd   = True ; Include secondary accounts.

  ; Initialize list objects.
```

```
  primAcctList.listObj.List.Count = 0
  secAcctList.listObj.List.Count  = 0

  wgLib.enumAccounts(acctTbName, inclSecnd) ; Create a table listing accounts.

  ; Fill the lists.
  acctTC.open(acctTbName)
  scan acctTC:
    switch
      case acctTC.PrimaryOrSecondary = "Primary" :
        primAcctList.listObj.List.Selection =
          primAcctList.listObj.List.Selection + 1
        primAcctList.listObj.List.Value = acctTC.AccountName

      case acctTC.PrimaryOrSecondary = "Secondary" :
        secAcctList.listObj.List.Selection =
          secAcctList.listObj.List.Selection + 1
        secAcctList.listObj.List.Value = acctTC.AccountName
    endSwitch
  endScan
  acctTC.close()  ; Close the cursor before deleting the table.
  acctTb.attach(acctTbName)
  acctTb.delete() ; No further need for this data.
endMethod
```

See Also

enumAlerts
enumPublications
enumSubscriptions

enumAlerts

Method Creates a table listing Object Exchange alerts.

Syntax **enumAlerts** (***tableName*** String) Logical

Description Creates a table of data about alerts, where *tableName* is the file name of the table to create. The following table shows the structure.

Field name	Type	Description	Sample values
Object_DateCreated	D	The date the alert was issued. Format depends on system settings.	7/23/97
Object_TimeCreated	A15	The time the alert was issued. Format depends on system settings.	"09:56:46.00"
Object_SeqNo	S	This alert's position in the list.	1
Object_DateLastDistrib	D	Same as Object_DateCreated	7/23/97
Object_TimeLastDistrib	A15	Same as Object_TimeCreated	"09:56:46.00"
Object_Key	A60	String that identifies the alert.	"&1917341207271993"
Object_DistributedBy	A60	Name of user.	"rhall"
Object_Description	A80	Message describing the alert condition.	"Post Office doesn't exist", "You have a problem with your address(es)"
Object_Tag	A80	More information about the alert or object.	"No object associated with alert", "Q4 sales"
Object_NumberOfVersions	N	Version number of the alert.	1.00

This method requires a full lock on *tableName*. If the table contains data, this method replaces it without asking for confirmation. This method returns True if it succeeds; otherwise, it returns False.

Example This example calls **enumAlerts** to create a table listing the alerts in the user's object store, then scans the table for alerts more than two days old and calls **deleteObject** to delete them from the object store. (The example for **deleteObject** shows a technique for deleting one object at a time).

```
method pushButton(var eventInfo Event)
{
enumAlerts
Assumptions:
- wgLib is a Library var declared in the form's Var window.
- wgLib is opened in the form's built-in open method.
- All workgroup methods are declared in the form's Uses window.
}
  var
    ageDate      Date
```

```
    alertsTbName,
    msgText       String
    alertsTb      Table
    alertsTC      TCursor
  endVar

  ageDate      = Date(today() - 2) ; 2 days old
  alertsTbName = ":PRIV:usrAlert"
  msgText      = "Couldn't open " + alertsTbName

  wgLib.enumAlerts(alertsTbName) ; Create a table listing alerts.
  if alertsTC.open(alertsTbName) then
    alertsTC.edit()

    ; Scan the table for alerts more than 2 days old and
    ; delete them from the object store (not from the table).
    scan alertsTC:
      if alertsTC.Object_DateLastDistrib < ageDate then
        wgLib.deleteObject(alertsTC.Object_Key)
      endIf
    endScan

    alertsTC.endEdit()
    alertsTC.close()  ; Close the cursor before deleting the table.

    alertsTb.attach(alertsTbName)
    alertsTb.delete() ; No further need for this data.
  else
    errorShow(msgText)
    return
  endIf

endMethod
```

See Also

deleteObject
enumAccounts
enumPublications
enumSubscriptions
getAlertDetails

enumPublications

Method Creates a table listing the Paradox publications in the user's object store.

Syntax **enumPublications** (***tableName*** String) Logical

Description Creates a table of data about the publications in the user's object store, where *tableName* is the file name of the table to create. The following table shows the structure.

Field name	Type	Description	Sample values
Object_DateCreated	D	The date the publication was issued. Format depends on system settings.	7/23/97
Object_TimeCreated	A15	The time the publication was issued. Format depends on system settings.	"09:56:46.00"
Object_SeqNo	S	This publication's position in the list.	1
Object_DateLastDistrib	D	The date the publication was last distributed. Format depends on system settings.	7/23/97
Object_TimeLastDistrib	A15	The time the publication was last distributed. Format depends on system settings.	"09:56:46.00"
Object_Key	A60	String that identifies the publication.	"ORDERS.20491529"
Object_DistributedBy	A60	Publications are always distributed by OBEX.	"OBEX"
Object_Description	A80	Message describing the publication (may be user-defined).	"Q3 Sales", "Save this as NewCust.DB"
Object_Tag	A80	More information about the publication (supplied by OBEX).	"Publish,Pages,Text, Table Data:Q4,1"
Object_NumberOfVersions	N	Version number of the publication .	1.00

This method requires a full lock on *tableName*. If the table contains data, this method replaces it without asking for confirmation.

This method returns True if it succeeds; otherwise, it returns False.

Example See deleteObject for an example showing how to call **enumPublications**; see issueNewVersion for an example showing how to use the resulting table.

See Also enumAccounts
enumAlerts
enumSubscriptions

enumSubscriptions

Method Creates a table listing the subscriptions in the user's object store.

Syntax **enumSubscriptions** (***tableName*** String) Logical

Description Creates a table of subscription data, where *tableName* is the file name of the table to create. The following table shows the structure.

Field name	Type	Description	Sample values
Object_DateCreated	D	The date the subscription was created. Format depends on system settings.	7/23/97
Object_TimeCreated	A15	The time the subscription was created. Format depends on system settings.	"09:56:46.00"
Object_SeqNo	S	This subscription's position in the list	1
Object_DateLastDistrib	D	The date the subscription was last distributed. Format depends on system settings.	7/23/97
Object_TimeLastDistrib	A15	The time the subscription was last distributed. Format depends on system settings.	"09:56:46.00"
Object_Key	A60	String that identifies the subscription	"ISSUES.11755103", "Q2orders"
Object_DistributedBy	A60	Who distributed this subscription?	"RGRETTER (Lan)"
Object_Description	A80	Message describing the subscription (may be user-defined)	"Q3 Sales", "Orders by state"
Object_Tag	A80	More information about the subscription (supplied by The Object Exchange)	"Publish,Pages,Text, File set:Orders,1"
Object_NumberOfVersions	N	Version number of the subscription	1.00

This method requires a full lock on *tableName*. If the table contains data, this method replaces it without asking for confirmation.

This method returns True if it succeeds; otherwise, it returns False.

Example See cancelSubscription, deleteObject, and modifySubscribers.

See Also enumAccounts
enumAlerts
enumPublications

getAlertDetails

Method Gets information about a specified Object Exchange alert.

Syntax **getAlertDetails** (***objectKey*** String, var ***alertDetails*** String) Logical

Description Returns in *alertDetails* information about an Object Exchange alert, where *objectKey* is a string identifying the alert. To get valid values for *objectKey*, call **enumAlerts** and take values from the *Object_Key* field.

This method returns True if it succeeds; otherwise, it returns False.

Example This example shows how to display the details for each alert in a table. First, use **enumAlerts** to create a table of alerts, then create a form bound to that table. Attach the following code to the page's built-in **action** method to get alert details each time the form displays another record.

```
method action(var eventInfo ActionEvent)
; This code is attached to the page's built-in action method.
{
getAlertDetails
Assumptions:
- wgLib is a Library var declared in the form's Var window.
- wgLib is opened in the form's built-in open method.
- All workgroup methods are declared in the form's Uses window.
- USRALERT is a table created by calling enumAlerts.
- This form contains the following field objects:

 DateFld       is bound to the Object_DateCreated field of USRALERT.
 TimeFld       is bound to the Object_TimeCreated field of USRALERT.
 DescFld       is bound to the Object_Description field of USRALERT.
 DetailsFld    is an unbound field that displays the data returned by getAlertDetails.
}

  var
    alertDetails,
    dmTbName,
    dmFldName,
    objectKey       String
  endVar

  ; Initialize variables.
  dmTbName  = ":PRIV:usrAlert"
  dmFldName = "Object_Key"

  ; DataArriveRecord is triggered each time the form displays another record.
  if eventInfo.id() = DataArriveRecord then
    doDefault ; Execute the default code first.

    ; Get the value of the key field for this record.
    dmGet(dmTbName, dmFldName, objectKey)

    ; Use the value to call getAlertDetails.
    if objectKey <> "" then
      wgLib.getAlertDetails(objectKey, alertDetails)
      DetailsFld.Value = alertDetails
    else
```

```
        return
      endIf
    endIf

endMethod
```

Attach the following code to the built-in **newValue** method of the *DescFld* field object. It forces the form to display details for the first alert when the form opens.

```
method newValue(var eventInfo Event)
  if eventInfo.reason() = StartUpValue then
    doDefault
    self.action(DataArriveRecord)
  endIf
endMethod
```

See Also

enumAlerts
getNote

getFileList

Method Gets a list of the files in an object.

Syntax **getFileList** (***objectKey*** String, ***backVersion*** SmallInt, var ***fileList*** ArrayOfStrings) Logical

Description Fills *fileList* with a list of the files in the object specified by *objectKey*. (Typically, *objectKey* is used to specify a file set, but this method also works with Table Data and Query Results.) To get valid values for *objectKey*, call **enumPublications** or **enumSubscriptions** and take values from the Object_Key field.

If the object is a publication, *fileList* lists the full file name (including the path) of each file. If the object is a subscription, *fileList* omits the path.

The argument *backVersion* specifies the version of the object, where a value of 0 specifies the newest (last-issued) version, and larger values specify older versions. To find out how many versions of an object are available, call **enumPublications** or **enumSubscriptions**, then search for *objectKey* in the Object_Key field of the resulting table and read the value of the Object_NumberOfVersions field for that record.

If *objectKey* or *backVersion* does not exist, this method fails. This method returns True if it succeeds; otherwise, it returns False.

Example This example shows how to display the file list for a file set. When you click a radio button, code attached to the field's built-in **newValue** method executes. It fills the drop-down edit list with descriptions of each object of the specified type. Then, when you choose an item from the list, code attached to the list's built-in **newValue** method executes and displays a view dialog box listing the files in that file set.

The following code is attached to the built-in **newValue** method of *objTypeFld*, an unbound field object displayed as radio buttons.

```
method newValue(var eventInfo Event)
{
getFileList
Assumptions:
- wgLib is a Library var declared in the form's Var window.
- wgLib is opened in the form's built-in open method.
- All workgroup methods are declared in the form's Uses window.
- Custom data types are declared in the form's Type window.
- This form contains an unbound field object named objTypeFld
  displayed as radio buttons, and an unbound drop-down edit list
  named objDescList that contains a list object named listObj.
- A custom method named addFileSetToList has been declared elsewhere.
- The following code executes when you choose one of the
  radio buttons in objTypeFld.
}
  var
    pubsTbName, subsTbName,
    objType, objectKey       String
    objTC                    TCursor
  endVar

  ; Initialize variables.
  pubsTbName = ":PRIV:usrPubs"
  subsTbName = ":PRIV:usrSubs"

  if eventInfo.reason() = EditValue then
    doDefault
```

```
    objType = self.value

    switch
      case objType = "Publication" :
        wgLib.enumPublications(pubsTbName)
        objTC.open(pubsTbName)

      case objType = "Subscription" :
        wgLib.enumSubscriptions(subsTbName)
        objTC.open(subsTbName)
    endSwitch

    if addFileSetToList(objTC) then
      ; Display the first item in the list.
      objDescList.listObj.List.Selection = 1
      objDescList.Value = objDescList.listObj.List.Value
    else
      return
    endIf
  endIf
endMethod
```

Following is the code for the custom method **addFileSetToList**. It takes as an argument a TCursor opened onto a table (either of publications or subscriptions) and calls **getObjectSummary** for each object in the table. If the object is a file set (that is, if the value of the SourceType index of the **getObjectSummary** DynArray is "File set"), this code adds a description of the object to the drop-down edit list.

```
method addFileSetToList(const objTC TCursor) Logical
; Custom method called by objTypeFld::newValue
  var
    objSumDA                DynArrayOfAny
    keyFld, daIndex, fileSetID,
    msgCaption, msgText     String
  endvar

  ; Initialize variables.
  keyFld     = "Object_Key"
  daIndex    = "SourceType"
  fileSetID  = "File set"
  msgCaption = "No file sets"
  msgText    = "There are no file sets of this type."

  objDescList.listObj.List.Count = 0 ; Empty the list.
  setMouseShape(MouseWait)
  message("Filling list ...")

; Scan the table of objects. If an object is a file set,
; add its description to the drop-down edit list.

  scan objTC:
    wgLib.getObjectSummary(objTC.(keyFld), objSumDA)
    if objSumDA[daIndex] = fileSetID then
      objDescList.listObj.List.Selection =
      objDescList.listObj.List.Selection + 1
```

```
      objDescList.listObj.List.Value = objSumDA["Description"]
    endIf
  endScan

  setMouseShape(MouseArrow)
  message("")

  if objDescList.listObj.List.Count = 0 then
    msgInfo(msgCaption, msgText)
    return False
  else
    return True
  endIf
endMethod
```

The following code is attached to the built-in **newValue** method of *objDescList*, the drop-down edit list. When you choose a file set description from the list, this code calls **getFileList** to generate a list of the files in that file set. Then it displays the list in a view dialog box.

```
method newValue(var eventInfo Event)

  var
    pubsTbName, subsTbName,
    descFldName, keyFldName,
    viewCaption, objectKey    String
    backVer                   SmallInt
    fileListAR                ArrayOfStrings
    objTC                     TCursor
  endvar

  ; Initialize variables.
  pubsTbName  = ":PRIV:usrPubs"
  subsTbName  = ":PRIV:usrSubs"
  descFldName = "Object_Description"
  keyFldName  = "Object_Key"
  viewCaption = "File list: "
  backVer     = 0 ; This example always uses the latest version.

  if eventInfo.reason() = EditValue then
    switch
      case objTypeFld.Value = "Publication" :
        objTC.open(pubsTbName)

      case objTypeFld.Value = "Subscription" :
        objTC.open(subsTbName)

      otherwise : msgInfo("Specify an Object Type.",
                    "Choose Publication or Subscription.")
                  return
    endSwitch

    if objTC.locate(descFldName, self.Value) then
      objectKey = objTC.(keyFldName)
      if wgLib.getFileList(objectKey, backVer, fileListAR) then
```

```
        fileListAR.view(viewCaption + self.Value)
      else
        errorShow("Could not get file list.")
        return
      endIf
    endIf
  endIf
endMethod
```

The following code is attached to the drop-down edit list's built-in **close** method. It empties the list.

```
method close(var eventInfo Event)
  ; Empty the list.
  self.listObj.List.Count = 0
endMethod
```

See Also getObjectSummary

getNote

Method Gets the note that was sent with a publication or subscription.

Syntax **getNote** (***objectKey*** String, ***backVersion*** SmallInt, var ***note*** String) Logical

Description Returns in *note* the note that was sent with the publication or subscription specified by *objectKey*. To get valid values for *objectKey*, call **enumPublications** or **enumSubscriptions** and take values from the *Object_Key* field. The argument *backVersion* specifies the version of the object, where a value of 0 specifies the newest (last-issued) version, and larger values specify older versions. To find out how many versions of an object are available, call **enumPublications** or **enumSubscriptions**, then search for *objectKey* in the Object_Key field of the resulting table and read the value of the Object_NumberOfVersions field for that record

If *objectKey* or *backVersion* does not exist, this method fails. This method returns True if it succeeds; otherwise, it returns False.

Example This example shows how to display the note attached to a specified subscription. First, use **enumSubscriptions** to create a table of subscriptions named :PRIV:usrSubs, then create a form bound to that table. Attach the following code to a button on the form. Run the form, choose a subscription in the *DescriptionTF* table frame and click the button to display the note for that subscription. By default, this code displays the note for the last-issued (newest) version of the object; you can display notes for older versions by entering a value in the *BackVersionFld* field object.

```
method pushButton(var eventInfo Event)
{
getNote
Assumptions:
- wgLib is a Library var declared in the form's Var window.
- wgLib is opened in the form's built-in open method.
- All workgroup methods are declared in the form's Uses window.
- USRSUBS is a table created by calling enumSubscriptions.
- This form contains a table frame, field objects, and a button:

 DescriptionTF  is a table frame bound to USRSUBS.
              It displays only the Object_Description field.
              The user highlights a field to select an object.
 DateFld       is bound to the Object_DateLastDistrib field of USRSUBS.
 TimeFld       is bound to the Object_TimeLastDistrib field of USRSUBS.
 FromFld       is bound to the Object_DistributedBy field of USRSUBS.
 VersionsFld   is bound to the Object_NumberOfVersions field of USRSUBS.
 BackVersionFld is an unbound field object.
 Note          is a button. The user highlights a field to select an object,
              then clicks the button to execute the following code which
              displays the note in a dialog box.
}

  var
    dmTbName, dmFldName,
    objectKey,  theNote,
    msgCaption, msgText,
    badVersionMsg              String
    backVer                 SmallInt
  endVar
```

```
  ; Initialize variables.
  dmTbName   = ":PRIV:usrSubs"
  dmFldName  = "Object_Key"
  msgCaption = "Note"
  msgText    = "Could not get note."
  badVersionMsg = "Enter a number from 0 to " +
                  String((SmallInt(VersionsFld.Value) - 1))

  ; Get the value of the Object_Key field for the current record.
  dmGet(dmTbName, dmFldName, objectKey)

  ; Decide which version to use by reading the value of the BackVersion field
  ;object. If the field is blank, use 0.
  if BackVersion.Value = "" then
    backVer = 0
  else
    try
      backVer = SmallInt(BackVersion.Value)
    onFail
    msgStop("Invalid version number", badVersionMsg)
    errorClear()
    return
    endTry
  endIf
  if backVer > (Versions.Value - 1) then
    msgStop("Invalid version number", badVersionMsg)
    return
  endIf

  ; Get the note for the selected object.
  if wgLib.getNote(objectKey, backVer, theNote) then
    theNote.view(msgCaption)
  else
    errorShow(msgText)
    return
  endIf
endMethod
```

See Also getAlertDetails

getObexStatus

Method Fills a DynArray with Object Exchange status information.

Syntax **getObexStatus** (var ***obexStatus*** DynArrayOfAny) Logical

Description Fills the DynArray *obexStatus* with general Object Exchange status information. The following table lists the DynArray indexes.

Index	Description	Sample values
ActiveAccount	Name of active account	"MCI", "None", "R&D LAN"
NumberOfAccounts	Total number of user's OBEX accounts	"3"
NumberOfAlerts	Number of alerts in the Object Exchange	"5"
NumberOfOutgoing	Number of publications to be issued	"2"

If the Object Exchange does not respond, this method fails. This method returns True if it succeeds; otherwise, it returns False.

Example This example calls **getObexStatus**, then calls a custom method (assumed to be defined elsewhere) to process alerts. The examples for **enumAlerts** and **getAlertDetails** show code that processes alerts.

```
method pushButton(var eventInfo Event)
{
getObexStatus
Assumptions:
- wgLib is a Library var declared in the form's Var window.
- wgLib is opened in the form's built-in open method.
- All workgroup methods are declared in the form's Uses window.
- Custom data types are declared in the form's Type window.
- A custom method named processAlerts has been defined elsewhere.
}
  var
    statusDA DynArrayOfAny
    daIndex  String
  endVar

  ; Initialize variables.
  daIndex = "NumberOfAlerts"

  wgLib.getObexStatus(statusDA)         ; Get OBEX status.

  ; If there are alerts, call a custom method to process them.
  ; You could also call statusDA.view("OBEX status")
  if SmallInt(statusDA[daIndex]) > 0 then
    processAlerts() ; Call a custom method (defined elsewhere).
  endIf
endMethod
```

See Also getObjectSummary

getObjectSummary

Method Fills a DynArray with summary information about an object.

Syntax **getObjectSummary** (***objectKey*** String, var ***objectSummary*** DynArrayOfAny) Logical

Description Fills the DynArray *objectSummary* with data about the object specified by *objectKey*. Objects can be alerts, publications, or subscriptions. To get valid values for *objectKey*, call **enumAlerts**, **enumPublications**, or **enumSubscriptions** and take values from the *Object_Key* field. If *objectSummary* already exists, this method overwrites it without asking for confirmation. The following table lists the indexes of the DynArray.

Index	Description	Sample values
DateCreated	Date object was created. Format depends on system settings.	"09/11/98"
DateLastDistributed	Date object was last distributed. Format depends on system settings.	"12/12/98"
Description	Description of object (may be user-defined).	"Sales by zip code"
DistributedBy	Who distributed this object?	RHALL (MHS)
MultiVersion	Is this a multiversion object?	"Yes", "No"
NumberOfVersions	How many versions are stored?	"1"
ObjectKey	String that identifies the object.	"ORDERS.1755103"
ObjectName	Name of this object.	"ORDERS"
SourceType	What kind of object is it?	"Table Data", "Query Result", "File set"
Status	Status of this object.	"Available", "Pending"
Time Created	Time object was created. Format depends on system settings.	"19:07:44.00"
TimeLastDistributed	Time object was last distributed. Format depends on system settings.	"19:07:44.00"
VersionDepth	How many versions can be stored?	"2"

If *objectKey* does not exist in the user's object store, this method fails. This method returns True if it succeeds; otherwise, it returns False.

Example This example shows how to display summary data for a specified subscription. First, use **enumSubscriptions** to create a table named :PRIV:usrSubs, then create a form bound to that table. Attach the following code to a button on the form. When you run the form, choose a subscription in the *DescriptionTF* table frame and click the button to display summary data for that subscription.

```
method pushButton(var eventInfo Event)
{
getObjectSummary
Assumptions:
- wgLib is a Library var declared in the form's Var window.
- wgLib is opened in the form's built-in open method.
- All workgroup methods are declared in the form's Uses window.
- Custom data types are declared in the form's Type window.
- USRSUBS is a table created by calling enumSubscriptions.
- This form contains a table frame, field objects, and a button:

  DescriptionTF  is a table frame bound to USRSUBS.
                 It displays only the Object_Description field.
  DateFld        is bound to the Object_DateLastDistrib field of USRSUBS.
  TimeFld        is bound to the Object_TimeLastDistrib field of USRSUBS.
  FromFld        is bound to the Object_DistributedBy field of USRSUBS.

  Summary        is a button. The user highlights a field to select an object,
                 then clicks the button to execute the following code which
                 displays summary data in a dialog box.
}
  var
    objectKey,
    dmTbName, dmFldName,
    msgCaption, msgText  String
    summaryDA            DynArrayOfAny
  endVar

  ; Initialize variables.
  dmTbName   = ":PRIV:usrSubs"
  dmFldName  = "Object_Key"
  msgCaption = "Object Summary"
  msgText    = "Could not get summary data."

  ; Get the value of the Object_Key field for the current record.
  dmGet(dmTbName, dmFldName, objectKey)

  ; Get summary data for the selected object.
  if wgLib.getObjectSummary(objectKey, summaryDA) then
    summaryDA.view(msgCaption)
  else
    errorShow(msgText)
    return
  endIf
endMethod
```

See Also getObexStatus

getSubscribers

Method	Fills an array with addresses of subscribers to a specified publication.
Syntax	**getSubscribers** (***objectKey*** String, var ***subscribers*** ArrayOfStrings) Logical
Description	Fills the array specified by *subscribers* with the addresses of subscribers to the publication specified by *objectKey*. To get valid values for *objectKey*, call **enumPublications** and take values from the *Object_Key* field. This method returns True if it succeeds; otherwise, it returns False.
Example	See modifySubscribers.
See Also	enumSubscriptions verifyAddresses

issueNewVersion

Method Issues a new version of an existing Paradox publication.

Syntax **issueNewVersion** (***objectKey*** String, ***note*** String) Logical

Description Issues a new version of a publication specified by *objectKey*. To get valid values for *objectKey*, call **enumPublications** and take values from the *Object_Key* field. The argument *note* specifies the note to send with the new version. *note* can be either a quoted string (maximum length: 255 characters) or an ObjectPAL String variable (maximum length: 32,767 characters).

Unlike **transfer** and **publish**, this method assumes that the object key for the table, query result, or file set still exists in the object store of the sender.

If the publication no longer exists in the user's object store, this method fails. This method returns True if it succeeds; otherwise, it returns False.

Note: Data is not sent to subscribers until you poll the appropriate account, either interactively or by calling **poll**.

Example This example shows how to use a form to issue new versions of existing publications. First, use **enumPublications** to create a table named :PRIV:usrPubs, then create a form bound to that table. Run the form, choose a publication in the *DescriptionTF* table frame and click the button to issue a new version of that publication. This code assumes you have declared a custom method named **pollAcct** that polls a selected account (see the example for **poll**). This code also displays a dialog box that prompts you to enter a note to send with the publication.

```
method pushButton(var eventInfo Event)
{
issueNewVersion
Assumptions:
- wgLib is a Library var declared in the form's Var window.
- wgLib is opened in the form's built-in open method.
- All workgroup methods are declared in the form's Uses window.
- USRPUBS is a table created by calling enumPublications.
- A custom method named pollAcct is declared elsewhere.
- This form contains a table frame, field objects, and a button:

 DescriptionTF  is a table frame bound to USRPUBS.
              It displays only the Object_Description field.
              The user highlights a field to select an object.
 DateFld       is bound to the Object_DateLastDistrib field of USRPUBS.
 TimeFld       is bound to the Object_TimeLastDistrib field of USRPUBS.

 New_Version    is a button. The user highlights a field to select a
               publication, then clicks the button to execute the following
              code, which issues a new version of the publication.
}

  var
    objectKey,
    dmTbName, dmFldName,
    theNote, notePrompt,
    msgCaption, msgText     String
  endVar

  ; Initialize variables.
```

```
  dmTbName   = ":PRIV:usrPubs"
  dmFldName  = "Object_Key"
  msgCaption = "Note"
  msgText    = "Could not issue new version."
  notePrompt = "Enter a note or click Cancel."
  theNote    = notePrompt

  ; Get the value of the Object_Key field for the current record.
  dmGet(dmTbName, dmFldName, objectKey)

  ; Open a dialog box where the user can enter a note.
  theNote.view(msgCaption)

  ; If the user clicks Cancel, the note won't change;
  ; it will be the same as the prompt.
  if theNote = notePrompt then
    theNote = ""
  endIf

  ; Issue a new version of the selected publication.
  if wgLib.issueNewVersion(objectKey, theNote) then
    ; Call a custom method to poll a selected account.
    ;(See the example for poll.)
    pollAcct()
  else
    errorShow(msgText)
    return
  endIf
endMethod
```

See Also

publish
transfer
poll

modifySubscribers

Method Modifies the subscriber list for a publication in the user's object store.

Syntax **modifySubscribers** (***objectKey*** String, var ***subscribers*** ArrayOfStrings,
distributeLastVersion Logical) Logical

Description Modifies the subscriber list for a publication identified by *objectKey*. To get valid values for *objectKey*, call **enumPublications** and take values from the Object_Key field.

subscribers is an array representing the new subscriber list. It stores the addresses of workgroup members who will receive the publication. It completely replaces any existing subscriber list for that publication. (Use **getSubscribers** to get the addresses of current subscribers; use **verifyAddresses** to make sure subscriber addresses are valid.)

distributeLastVersion specifies whether to distribute the last-issued version of the publication to new subscribers in the list. If you want new subscribers to receive the current version of the publication in the object store, already received by other subscribers, specify True. In other words, specify True to give new subscribers the data that other subscribers already have. If you want new subscribers to start receiving versions only when the next version is issued to all subscribers, specify False.

If *objectKey* does not exist in the user's object store, this method fails. This method returns True if it succeeds; otherwise, it returns False.

Example This example shows how to create a form you can use to modify the subscriber list for a specified publication. Start by using **enumPublications** to create a table of publications named :PRIV:USRPUBS.DB. Then use Paradox interactively to create :PRIV:TEMPADDR.DB, a table with one field: Subscribers, A100. Create the form and bind it to the *TempAddr* table, select a tabular layout, then place an unbound drop-down edit field and a button.

Declare the variable *objectKey* in the form's Var window to make it available to both the list and the table frame.

```
Var
  objectKey String
endVar
```

The following code is attached to the drop-down list box's built-in **open** method. It assumes the list box contains a list object named *listObj*. It uses the DataSource property and fills the list with values from the Object_Description field of the *UsrPubs* table.

```
method open(var eventInfo Event)
  doDefault              ; Execute the built-in code first.
  self.listObj.DataSource = "[:PRIV:UsrPubs.Object_Description]"
endMethod
```

The following code is attached to the list box's built-in **newValue** method. This code executes when you choose an item from the list.

```
method newValue(var eventInfo Event)
; This code assumes custom data types are declared in the form's Type window.

    var
            keyFldName, descFldName,
            addrFldName, pubsTbName      String
```

```
        i                   SmallInt
        subAddrAR           ArrayOfStrings
        subAddrTC, pubsTC                   TCursor
    endVar

    if eventInfo.reason() = EditValue then
        doDefault

        ; Initialize variables.
        keyFldName  = "Object_Key"
        descFldName = "Object_Description"
        addrFldName = "Subscribers"
        pubsTbName  = ":PRIV:usrPubs"

        ; Get the object key for this publication.
        pubsTC.open(pubsTbName)
        pubsTC.locate(descFldName, self.Value)
        objectKey = pubsTC.(keyFldName)

        ; Use the object key to get subscriber addresses.
        wgLib.getSubscribers(objectKey, subAddrAR)

        TEMPADDR.edit() ; Put the table frame into Edit mode.
        subAddrTC.attach(TEMPADDR) ; Attach a TCursor (also in Edit mode).
        subAddrTC.empty()

        ; Copy addresses from the array to the TCursor.
        for i from 1 to subAddrAR.size()
            subAddrTC.insertAfterRecord()
            subAddrTC.(addrFldName) = subAddrAR[i]
        endFor
        subAddrTC.postRecord()

        TEMPADDR.reSync(subAddrTC) ; Update the table frame with data from the
TCursor.
    endIf
endMethod
```

Now the user can edit address records in the table frame. When edits are complete, the user can click OK to commit changes and modify the subscriber list.

The following code is attached to the OK button. It verifies the addresses in the new list, then calls **modifySubscribers** to update the subscriber list for the publication.

```
method pushButton(var eventInfo Event)
; This code assumes custom data types are declared in the form's Type window.
    var
        updateNewSubs  Logical
        addrFldName    String
        subAddrAR      ArrayOfStrings
        subAddrTC      TCursor
    endVar

    ; Initialize variables.
    addrFldName = "Subscribers"
    updateNewSubs = Yes ; Send last-issued version to new subscribers.
```

```
        TEMPADDR.endEdit() ; No more edits to the table frame.
        subAddrTC.attach(TEMPADDR)

        scan subAddrTC:
                subAddrAR.addLast(subAddrTC.(addrFldName))
        endScan

        ; Make sure the addresses are valid.
    if not wgLib.verifyAddresses(subAddrAR) then
                errorShow("Invalid address")
                return
        endIf

    ; Modify the subscriber list and send the latest version
    ; of the publication to the new subscribers.

        if not objectKey.isAssigned() then
                ; objectKey is declared in the form's Var window,
                ; and is assigned when you choose a value from the list.
                msgInfo(" ", "Choose a publication from the list.")
                return
        endIf

    if wgLib.modifySubscribers(objectKey, subAddrAR, updateNewSubs) then
      pollAcct() ; Call a custom method (see the example for poll).
    else
      errorShow("Could not modify subscriber list.")
    endIf
endMethod
```

Attach the following code to the table frame's built-in **close** method.

```
method close(var eventInfo Event)
      self.empty()
endMethod
```

See Also getSubscribers
enumSubscriptions

poll

Method Makes the Object Exchange poll an account.

Syntax **poll** (***accountName*** String) Logical

Description Makes the Object Exchange poll *accountName*. To get valid values for *accountName*, call **enumAccounts** and take values from the *AccountName* field.

If *accountName* does not exist, this method fails. This method returns True if it succeeds; otherwise, it returns False.

Example This example shows how to poll a specified account. First, use **enumAccounts** to create a table named :PRIV:UsrAccts. Place a drop-down edit list in the form, and set the DataSource property of its list object to point to the AccountName field of the *UsrAccts* table. Then place a button and attach the following code. Run the form, choose an account name from the list, then click the button to poll that account.

```
method pushButton(var eventInfo Event)
{
poll
Assumptions:
- wgLib is a Library var declared in the form's Var window.
- wgLib is opened in the form's built-in open method.
- All workgroup methods are declared in the form's Uses window.
- This form contains a button and a drop-down edit list field named acctList
  that lists OBEX accounts (see enumAccounts for an example).
The list field contains a list object named listObj.
  The user chooses an account name from the list, then clicks the
  button to execute the following code.
}
  var
    dateFldName, descFldName,
    msgCaption, msgText,
    subsTbName, acctTbName   String
    pollDate              Date
    subsTC                TCursor
    newSubsAR               Array[] String
  endVar

  ; Initialize variables.
  pollDate    = today()
  acctName    = acctList.Value
  dateFldName = "Object_DateLastDistrib"
  descFldName = "Object_Description"
  subsTbName  = ":PRIV:usrSubs"
  msgCaption  = "Account name required"
  msgText     = "Choose an account name from the list."

  ; Make sure user chooses an account.
  if acctName = "" then
    msgInfo(msgCaption, msgText)
    return
  endIf

  ; Poll the account.
  if not wgLib.poll(acctName) then
```

```
    errorShow("Poll failed")
    return
  endIf
endMethod
```

The following code is attached to *acctList*, a field object displayed as a drop-down edit list. This code assumes that *acctList* contains a list object named *listObj*. This code fills the list with values from the AccountName field of the *UsrAccts* table.

```
method open(var eventInfo Event)
      doDefault
      self.listObj.DataSource = "[:PRIV:UsrAccts.AccountName]"
endMethod
```

See Also enumAccounts

publish

Method Creates a new multiversion publication.

Syntax **publish** (***objectKey*** String**,**
objectType String**,**
description String**,**
var ***fileList*** ArrayOfStrings**,**
var ***subscribers*** ArrayOfStrings**,**
versionDepth SmallInt**,**
note String) Logical

Description Creates a new multiversion publication. (Use **transfer** to send a single-version publication, use **issueNewVersion** to send an updated version of an existing publication.)

Note: Data is not sent to subscribers until you poll the appropriate account, either interactively or by calling **poll**.

For *objectKey*, you can specify a unique value yourself (maximum length: 40 characters) or specify an empty string (""). If you specify an empty string, OBEX generates a unique value when it sends the publication. If *objectKey* already exists, this method fails.

objectType can be "Table Data," "Query Result," or "File Set."

description is a string of up to 80 characters describing the object. You must supply a value for *description*.

fileList is an array of one or more file names (including path or alias). If you are publishing table data or a query result, this array should contain only one item: the file name of the table or query (if it contains more than one, the other items are ignored). If you are publishing a file set, this array contains the names of each file in the set.

subscribers is the subscriber list, an array containing the addresses of workgroup members who will receive the publication. If any addresses in the list are syntactically invalid, OBEX will create an alert when you poll, which you can access using the methods **enumAlerts** and **getAlertDetails**. You can use **verifyAddresses** to make sure addresses are valid. You can use **modifySubscribers** to add, change, or delete addresses in the subscriber list.

versionDepth specifies how many versions to make available to subscribers (maximum value: 99). The version depth cannot be changed after the initial publication.

note is the note to send with the first version of this publication. It can be changed with every new version (see **issueNewVersion** for more information). *note* can be either a quoted string (maximum length: 255 characters) or an ObjectPAL String variable (maximum length: 32,767 characters).

This method returns True if it succeeds; otherwise, it returns False.

Example This example shows how to create a form for publishing objects. First, create a form bound to :ADDRBOOK:Address, a table created by OBEX to store subscriber addresses, then place objects in the form. Run the form and click a radio button to specify an object type. Next, enter a description, a name, and a note in the appropriate fields. Then choose one address at a time from the table frame and click Add to add the address to the list. When the list is complete, click Publish to send the publication to the subscribers in the list.

The following code is attached to the Publish button. It displays the built-in Paradox Browser so you can choose an object to publish, then calls **publish** to publish it, and then calls a custom method named **pollAcct** to poll a selected account.

```
method pushButton(var eventInfo Event)
{
publish
```

```
Assumptions:
- wgLib is a Library var declared in the form's Var window.
- wgLib is opened in the form's built-in open method.
- All workgroup methods are declared in the form's Uses window.
- Custom data types are declared in the form's Type window.
- A custom method named pollAcct is declared elsewhere.
- The alias :ADDRBOOK: has been defined to point to the directory
  containing the ADDRESS table. (By default :ADDRBOOK: points to
C:\PDOXWIN\ADDRBOOK.)
- This form contains a table frame, field objects, and 2 buttons:

  pubTypeFld     is an unbound field object displayed as 3 radio buttons.
                 The user chooses a radio button to specify the type.

  pubDescFld     is an unbound field object.
                 The user types a description of the publication.

  pubNameFld     is an unbound field object.
                 The user types the name of the publication.

  pubNoteFld     is an unbound field object.
                 The user types a note about the publication.

  addrTF         is a table frame bound to :ADDRBOOK:ADDRESS.
                 The user highlights a field to select an address,
                 then clicks the Add button add the address to
                 Address List.

  addrListFld    is an unbound field object displayed as a list.
                 It contains a list object named addrListObj.

  Add            is a button that copies an address from the
                 Address Table to the Address List.

  Publish        is a button that executes the following code to
                 publish the publication.
}
  var
    FBI              FileBrowserInfo {see System::fileBrowser for info}
    srcTb, srcQbe, srcFileSet,
    objectKey, objectName,
    msgText, pubType,
    pubNote, pubDesc     String
    addrAR, fileListAR   ArrayOfStrings
    addrCount, i, v      SmallInt
  endVar

  ; Initialize variables.
      srcTb = "Table Data"
      srcQbe = "Query result"
      srcFileSet = "File Set"
   v = 3 ; Allow three versions of the publication.
  pubDesc   = pubDescFld.Value
  if pubDesc = "" then
    msgStop("Description required.", "Enter a description.")
    return
```

```
endIf

addrCount = addrListFld.addrListObj.List.Count
if addrCount = 0 then
  msgStop("No addresses", "Specify at least one address.")
  ; For simplicity, this example requires at least one address.
  ; You could allow 0 addresses and call modifySubscribers before polling.
  return
else
  addrAR.grow(addrCount) ; Set the array size to match the list size.
endIf

addrListFld.addrListObj.List.Selection = 1
for i from 1 to addrCount
  addrAR[i] = addrListFld.addrListObj.List.Value
  addrListFld.addrListObj.List.Selection =
  addrListFld.addrListObj.List.Selection + 1
endFor

objectKey = "" ; Let OBEX create a key value.
pubType   = pubTypeFld.Value
pubNote   = pubNoteFld.Value
msgText   = "Unable to publish the document."

switch
  case pubType = srcTb :
    FBI.SelectedType   = fbTable
    FBI.AllowableTypes = fbTable

    ; Use the Browser to get one file name.
    if fileBrowser(objectName, FBI) then
      fileListAR.setSize(1)
      fileListAR[1] = objectName
    else
      errorShow()
      return
    endIf

  case pubType         = srcQbe :
    FBI.SelectedType   = fbQuery
    FBI.AllowableTypes = fbQuery

    ; Use the Browser to get one file name.
    if fileBrowser(objectName, FBI) then
      fileListAR.setSize(1)
      fileListAR[1] = objectName
    else
      errorShow()
      return
    endIf

  case pubType = srcFileSet :
    FBI.Path = workingDir()
    FBI.AllowableTypes = fbFiles

    ; Use the Browser to get a list (array) of file names
```

```
        if not fileBrowser(fileListAR, FBI) then
          errorShow()
          return
        endIf

      otherwise : msgInfo("Object type required.",
                          "Choose Table Data, Query Result, or File Set.")
      return
    endSwitch

    pubNameFld.Value = fileListAR[1]

   ; Publish the document.
   if wgLib.publish(objectKey, pubType, pubDesc, fileListAR, addrAR, v, pubNote) then
pollAcct() ; Call a custom method (see the example for poll).
    else
      errorShow(msgText)
      return
    endIf
endMethod
```

The following code is attached to the Add button. It reads the current value from the table frame and adds it to the list.

```
method pushButton(var eventInfo Event)
  var
    msgCaption, msgText  String
  endVar

  ; Initialize variables.
  msgCaption = "No address"
  msgText    = "Choose an address from the table."

  if addrTF.Address.Value = "" then
    ; If user didn't choose an address, display a message and exit.
    msgInfo(msgCaption, msgText)
    return
  else
    ; Add the selected address to the list.
    addrListFld.addrListObj.List.Selection = addrListFld.addrListObj.List.Count + 1
    addrListFld.addrListObj.List.Value = addrTF.Address.Value
  endIf
endMethod
```

See Also

issueNewVersion
transfer

saveAs

Method Saves a specified version of an object that the user subscribes to into a table or directory.

Syntax **saveAs** (***objectKey*** String, ***backVersion*** SmallInt,
targetObjectName String, ***fillTableNow*** Logical) Logical

Description Saves a specified version of *objectKey* into a table (if the object is table data or a query result) or directory (if the object is a file set) specified in *targetObjectName*. To get valid values for *objectKey*, call **enumSubscriptions** and take values from the *Object_Key* field.

The argument *backVersion* specifies the version of the object, where a value of 0 specifies the newest (last-issued) version, and larger values specify older versions. To find out how many versions of an object are available, call **enumSubscriptions**, then search for *objectKey* in the Object_Key field of the resulting table and read the value of the Object_NumberOfVersions field for that record.

If the table or directory does not exist, Paradox creates it automatically. If *targetObjectName* is a table and already exists, it is emptied and filled with the data in *objectKey*. This method requires a full lock on the table. If *targetObjectName* is a file set and already exists, it retains all of its current data, and any file data that does not already exist in the file set is added to it.

Important: Files in a file set are not accessible until you call **saveAs** because they are stored in a compressed format in the user's object store. Calling **saveAs** uncompresses the files into a directory.

fillTableNow is relevant only to tables sent as Table Data or Query Result, not to tables sent in a file set. Specify True to create the table along with its data. Specify False to create an empty table; you can call **tableRefresh** to fill the table later. When you're working with a file set, you still must supply a value for *fillTableNow* (otherwise your code won't compile); however, the value is ignored when the code executes.

If *objectKey* does not exist in the user's object store, this method fails. This method returns True if it succeeds; otherwise, it returns False.

Example This example shows how to choose an object from a list of subscriptions and save it. First, use **enumSubscriptions** to create a table named :PRIV:usrSubs, then create a form and bind it to that table. Place a table frame, field object, and button. Run the form, choose an object from the list of descriptions, then click Save. A dialog box prompts you to name the object, then a call to **saveAs** saves the object, using that name.

```
method pushButton(var eventInfo Event)
{
saveAs
Assumptions:
- wgLib is a Library var declared in the form's Var window.
- wgLib is opened in the form's built-in open method.
- All workgroup methods are declared in the form's Uses window.
- USRSUBS is a table created by calling enumSubscriptions.
- This form contains a table frame, field objects, and a button:

  DescriptionTF   is a table frame bound to USRSUBS.
                  It displays only the Object_Description field.
                  The user highlights a field to select an object.
  FromFld         is bound to the Object_DistributedBy field of USRSUBS.

  Save            is a button. The user highlights a field to select an object,
                  then clicks the button to execute the following code.
}
```

```
var
  fillNow                Logical
  dmTbName,
  objectKey, objFldName,
  newObjName, promptText,
  saveOkText, saveErrText   String
  backVersion            SmallInt
endVar

; Initialize variables.
fillNow       = True ; This value is ignored if the object is a file set.
backVersion = 0 ; This example saves the latest version of the object.
dmTbName      = ":PRIV:usrSubs"
objFldName    = "Object_Key"
promptText    = "Enter a name for the object."
newObjName    = promptText
saveOkText    = "Object saved."
saveErrText   = "Object not saved."

newObjName.view("New object name")
if newObjName = promptText then
  message(saveErrText)
  return
endIf

; Get the value of the Object_Key field
; for the current record of the table specified in dmTbName.
dmGet(dmTbName, objFldName,  objectKey)

if wgLib.saveAs(objectKey, backVersion, newObjName, fillNow) then
  message(saveOkText)
else
  errorShow(saveErrText)
endIf
endMethod
```

See Also enumSubscriptions

tableRefresh

Method Refreshes a table with a specified version from the object store.

Syntax

tableRefresh (***objectKey*** String,
backVersion SmallInt,
tableName String,
var ***dateLastTaken*** Date,
var ***timeLastTaken*** String) Logical

Description

Refreshes *tableName* with the data contained in *objectKey*. To get valid values for *objectKey*, call **enumSubscriptions** and take values from the *Object_Key* field.

The argument *backVersion* specifies the version of the object, where a value of 0 specifies the newest (last-issued) version, and larger values specify older versions. To find out how many versions of an object are available, call **enumSubscriptions**, then search for *objectKey* in the Object_Key field of the resulting table and read the value of the Object_NumberOfVersions field for that record.

This method requires a full lock on *tableName*. If the table specified by *tableName* does not exist, Paradox creates it automatically.

This method returns in *dateLastDistributed* and *timeLastDistributed* the date and time the last version of this object was distributed, irrespective of the value of *backVersion*. You can compare these values to the date and time returned by **getObjectSummary** to make sure you're always working with the lastest data.

If *objectKey* does not exist in the user's object store, this method fails. This method returns True if it succeeds; otherwise, it returns False.

Example

This example shows how to create a form that updates a table with the latest data from your object store when you click a button. First, use **enumSubscriptions** to create a table named :PRIV:usrSubs, then create a form and bind it to the table. Place a table frame and a button. Create a second form containing an unbound table frame and a button. Save the second form and name it :PRIV:tempForm. This form is a template. Run the first form and choose an object by selecting a record in the table frame. When you click the button, a call to **saveAs** writes the data from the selected object to a table with that name. Then the template form opens and displays that table. This form contains a button you can click to call **getObjectSummary** and **tableRefresh** to refresh the table (if necessary) with the latest data from the object store. (You could also modify this code slightly and attach it to the form's built-in **timer** method to make the form refresh the data automatically.)

Following is the code from the first form's Var window.

```
Var
    wgLib                   Library
    lastDate                Date
    lastTime,
    objectKey,
    tbName                  String
    backVer                 SmallInt
endVar
```

The following code is attached to the Save With Form button's built-in **pushButton** method. It saves an object to a table, then calls the custom method **saveWithForm** to display that table in a form.

```
method pushButton(var eventInfo Event)
{
```

```
tableRefresh
Assumptions:
- wgLib is a Library var declared in the form's Var window.
- wgLib is opened in the form's built-in open method.
- All workgroup methods are declared in the form's Uses window.
- Custom data types are declared in the form's Type window.
- USRSUBS is a table created by calling enumSubscriptions.
- This form contains a custom method named saveWithForm.
- This form contains a table frame and a button:

  DescriptionTF   is a table frame bound to USRSUBS.
                  It displays only the Object_Description field.

  btnSaveWithForm is a button. The user highlights a field to select an object,
                  then clicks the button to execute the following code.
}
  const
    fillNow       = Yes
    dmTbName      = ":PRIV:usrSubs"
    dmObjFldName  = "Object_Key"
    okText        = "Table updated successfully."
    promptText    = "Enter a file name."
  endConst

  var
    saveIt        String
    objSumDA      DynArrayOfAny
  endVar

  ; Initialize variables.
  backVer = 0 ; Declared in form::Var. This example uses the latest version.
  tbName  = promptText ; Declared in form::Var.

  ; Get the value of the Object_Key field for the current record of the table.
  dmGet(dmTbName, dmObjFldName,  objectKey)

  wgLib.getobjectSummary(objectKey, objSumDA)
  if objSumDA["SourceType"] = "File Set" then
    return ; This example doesn't handle file sets.
  endIf

  ; Name the object before saving it.
  tbName = ":PRIV:" + objSumDA["ObjectName"]
  saveIt = msgQuestion("Save As", "Save table as: " + tbName + " ?")
  if saveIt <> "Yes" then
    message("Table not saved.")
    return
  endIf

  if wgLib.saveAs(objectKey, backVer, tbName, fillNow) then
    if saveWithForm(objectKey, tbName) then ; Call a custom method.
      message(okText)
    else
      errorShow("saveWithForm")
      return
    endIf
```

```
  else
    errorShow("saveAs")
    return
  endIf
endMethod
```

Following is the code for **saveWithForm**, a custom method declared at the form level. This method loads the template form and binds it to the new table created by the call to **saveAs** in the **pushButton** method.

```
method saveWithForm(const objectKey String, const tbName String) Logical
{
  Custom method called by btnSaveToForm::pushButton.
  Assumptions:
  - The template form :PRIV:tempForm has been created and saved.
    It contains an unbound table frame named tempTF and
    a text box named objKeyTxt.
}

  var
    tempForm       Form
    tempFormName   String
  endVar

  ; Initialize variables.
  tempFormName = ":PRIV:tempForm.fsl"

  if tempForm.load(tempFormName) then
    tempForm.tempTF.TableName  = tbName
    tempForm.objKeyTxt.Text    = objectKey
    ; Write the value of objectKey to a text box so code attached to the template
    ; form can use it to call getObjectSummary and tableRefresh.
  else
    return False
  endIf

  if tempForm.save(tbName) then
    tempForm.run()
    tempForm.bringToTop()
    return True
  else
    errorShow()
    tempForm.close()
    return False
  endIf
endMethod
```

Following is code from the template form's Var window.

```
Var
  wgLib        Library
  lastDate     Date
  lastTime,
  objectKey,
  tbName       String
```

```
  backVer       SmallInt
endVar
```

When the template form opens, the following code calls **tableRefresh** to update it with the latest data from the object store.

```
method open(var eventInfo Event)
if eventInfo.isPreFilter()
  then
    ; This code executes for each object on the form.

  else
    ; This code executes only for the form.
    doDefault
    wgLib.open("C:\\PDOXWIN\\WORKGRP\\WGPAL")

    objectKey = objKeyTxt.Text
    tbName = self.TableName
    backVer = 0

    ; Refresh the table with the latest data from the object store,
    ; and assign values to global variables lastDate and lastTime.
    if not wgLib.tableRefresh(objectKey, backVer, tbName, lastDate, lastTime) then
      errorShow("open -> tableRefresh")
      return
    endIf
endIf
endMethod
```

The following code is attached to the built-in **pushButton** of the Refresh Now button in the template form. It calls **getObjectSummary** and **tableRefresh** as needed to update the form with the latest data from the object store.

```
method pushButton(var eventInfo Event)
  var
    objSumDA DynArrayOfAny
  endVar
  if wgLib.getObjectSummary(objectKey, objSumDA) then
    objStoreDate = objSumDA["DateLastDistributed"]
    objStoreTime = objSumDA["TimeLastDistributed"]
  else
    errorShow()
    return
  endIf

  ; Global variables lastDate, lastTime, objectKey, backVer, and tbName
  ; are assigned in form::open.
  if (objStoreDate > lastDate) OR
    (objStoreDate = lastDate AND objStoreTime > lastTime) then
    wgLib.tableRefresh(objectKey, backVer, tbName, lastDate, lastTime)
  else
    message("This is the latest version.")
    return
  endIf
endMethod
```

See Also enumSubscriptions

transfer

Method Sends an object to subscribers.

Syntax

transfer (***objectKey*** String,
objectType String,
description String,
var ***fileList*** ArrayOfStrings,
var ***subscribers*** ArrayOfStrings
note String) Logical

Description Sends an object to subscribers. For *objectKey*, you can supply a unique value yourself (maximum length: 40 characters) or you can specify an empty string(""). If you specify an empty string, OBEX generates a unique value for *objectKey* when it transfers the object.

Note: Data is not sent to subscribers until you poll the appropriate account, either interactively or by calling **poll**.

The interactive Workgroup Desktop has no feature that does what **transfer** does. When you use **transfer** to send an object, that object is not added to the publications in the object store of the sender. It is added to the list of subscriptions in the receiver's object store, but it is never updated. In other words, **transfer** is a one-time, single-version operation. Use **publish** (equivalent to the Publish button on the Workgroup Desktop SpeedBar) to send a multiversion publication.

If *objectKey* already exists, this method fails. This method returns True if it succeeds; otherwise, it returns False.

objectType can be "Table Data," "Query Result," or "File Set."

description is a string of up to 80 characters describing the object. You must supply a value for *description*.

fileList is an array of one or more file names (including path or alias). If you are sending table data or a query result, this array should contain only one item: the file name of the table or query (if it contains more than one, the other items are ignored). If you are sending a file set, this array contains the names of each file in the set.

subscribers is an array containing the addresses of subscribers who will receive the object. If any addresses in the list are invalid, the Object Exchange will create an alert when you poll, which you can access using the methods **enumAlerts** and **getAlertDetails**. Use **verifyAddresses** to make sure subscriber addresses are valid.

note is the note to send with the object. *note* can be either a quoted string (maximum length: 255 characters) or an ObjectPAL String variable (maximum length: 32,767 characters).

Example This example shows how to create a form for transferring objects. First, create a form bound to *:ADDRBOOK:Address*, a table created by OBEX to store subscriber addresses, then place objects. Run the form and click a radio button to specify an object type. Next, enter a description, a name, and a note in the appropriate fields. Then choose one address at a time from the table frame and click Add to add the address to the list. When the list is complete, click Transfer to send the publication to the subscribers in the list.

The following code is attached to the Transfer button. It displays the built-in Paradox Browser so you can choose an object to send, then calls **transfer** and a custom method named **pollAcct** (see the example for **poll**) to send it.

```
method pushButton(var eventInfo Event)
{
transfer
Assumptions:
- wgLib is a Library var declared in the form's Var window.
```

- wgLib is opened in the form's built-in open method.
- All workgroup methods are declared in the form's Uses window.
- Custom data types are declared in the form's Type window.
- A custom method named pollAcct has been declared elsewhere.
- The alias :ADDRBOOK: has been defined to point to the directory
  containing the Address table.
- This form contains a table frame, field objects, and 2 buttons:

  pubTypeFld   is an unbound field object displayed as 3 radio buttons.
                 The user chooses a radio button to specify the type.

  pubDescFld   is an unbound field object.
                 The user types a description of the publication.

  pubNameFld   is an unbound field object.
                 The user types the name of the publication.

  pubNoteFld   is an unbound field object.
                 The user types a note about the publication.

  addrTF   is a table frame bound to :ADDRBOOK:ADDRESS.
                 The user highlights a field to select an address,
                 then clicks Add to add the address to the Address List.

  addrListFld   is an unbound field object displayed as a list.
                 It contains a list object named addrListObj.

  Add   is a button that copies an address from the
                 Address table to the Address list.

  Transfer   is a button that executes the following code to
                 transfer the publication.
}
```
  var
    FBI              FileBrowserInfo {see System::fileBrowser for info}
    srcTb, srcQbe,
    srcFileSet, objectKey,
    msgText, pubType,
    pubNote, pubDesc,
    objectName           String
    addrAR, fileListAR   ArrayOfStrings
    addrCount, i         SmallInt
  endVar

  ; Initialize variables.
  pubDesc    = pubDescFld.Value
  if pubDesc = "" then
    msgStop("Description required.", "Enter a description.")
    return
  endIf

  addrCount = addrListFld.addrListObj.List.Count
  if addrCount = 0 then
    msgStop("No addresses", "Specify at least one address.")
    return
  else
```

```
  addrAR.grow(addrCount) ; Set the array size to match the list size.
endIf

addrListFld.addrListObj.List.Selection = 1
for i from 1 to addrCount
  addrAR[i] = addrListFld.addrListObj.List.Value
  addrListFld.addrListObj.List.Selection =
  addrListFld.addrListObj.List.Selection + 1
endFor

srcTb      = "Table Data"
srcQbe     = "Query Result"
srcFileSet = "File Set"
objectKey  = "" ; Let OBEX create a key value.
pubType    = pubTypeFld.Value
pubNote    = pubNoteFld.Value
msgText    = "Unable to transfer the document."

switch
  case pubType         = srcTb :
    FBI.SelectedType   = fbTable
    FBI.AllowableTypes = fbTable

    ; Use the Browser to get one file name.
    if fileBrowser(objectName, FBI) then
      fileListAR.setSize(1)
      fileListAR[1] = objectName
    else
      errorShow()
      return
    endIf

  case pubType         = srcQbe :
    FBI.SelectedType   = fbQuery
    FBI.AllowableTypes = fbQuery

    ; Use the Browser to get one file name.
    if fileBrowser(objectName, FBI) then
      fileListAR.setSize(1)
      fileListAR[1] = objectName
    else
      errorShow()
      return
    endIf

  case pubType     = srcFileSet :
    FBI.Path       = workingDir()
    FBI.AllowableTypes = fbFiles

    ; Use the Browser to get a list (array) of file names.
    if not fileBrowser(fileListAR, FBI) then
      errorShow()
      return
    endIf
  otherwise : msgInfo("Object type required.",
                      "Choose Table Data, Query Result, or File Set.")
```

```
            return

  endSwitch

  pubNameFld.Value = fileListAR[1]

  ; Transfer the document.
  if wgLib.transfer(objectKey, pubType, pubDesc, fileListAR, addrAR, pubNote) then
    pollAcct() ; Call a custom method (see the example for poll).
  else
    errorShow(msgText)
  endIf

endMethod
```

The following code is attached to the Add button. It reads the current value from the table frame and adds it to the list.

```
method pushButton(var eventInfo Event)
  var
    msgCaption, msgText  String
  endVar

  ; Initialize variables.
  msgCaption = "No address"
  msgText    = "Choose an address from the table."

  if addrTF.Address.Value = "" then
; If user didn't choose an address, display a message and exit.
    msgInfo(msgCaption, msgText)
    return
  else
; Add the selected address to the list.
    addrListFld.addrListObj.List.Selection = addrListFld.addrListObj.List.Count + 1
    addrListFld.addrListObj.List.Value = addrTF.Address.Value
  endIf
endMethod
```

See Also

issueNewVersion
publish
poll

verifyAddresses

Method Verifies one or more addresses.

Syntax **verifyAddresses** (var ***addresses*** ArrayOfStrings) Logical

Description Verifies addresses specified in the array *addresses*. This method returns True if it succeeds and all addresses in the array are valid; otherwise, it returns False. An address is valid if the syntax is correct and the specified transport has an active primary account configured. This method does not verify that the address exists; it verifies only the syntax.

Example This example shows how to verify addresses stored in a table. First, create a table of addresses (see **getSubscribers** and the example for **modifySubscribers** for more information), then create a form bound to that table. Place a table frame and a button in the form, then attach the following code to the button's built-in **pushButton** method. This code reads each record of the table into an array, then calls **verifyAddresses** to make sure that each address is valid.

```
method pushButton(var eventInfo Event)
{
verifyAddresses
Assumptions:
- wgLib is a Library var declared in the form's Var window.
- wgLib is opened in the form's built-in open method.
- All workgroup methods are declared in the form's Uses window.
- Custom data types are declared in the form's Type window.
- SUBADDR is a table (created elsewhere) that lists subscriber addresses.
- This form contains a table frame and a button:

 subAddrTF    is a table frame named subAddrTF bound to SUBADDR.
              It displays only the Address field.

 OK           is a button. The user clicks this button to execute
              the following code to verify the list of addresses
              in the table frame.
}
  var
    subAddrAR             ArrayOfStrings
    addrFldName,
    errCaption, errText,
    okCaption, okText      String
    subAddrTC             TCursor
  endVar

  ; Initialize variables.
  addrFldName = "Address"
  errCaption  = "Address list is empty."
  errText     = "Enter at least one subscriber address or click Cancel."
  okCaption   = "Addresses are OK"
  okText      = "All addresses are valid."

  ; The table frame must contain at least one address.
  if subAddrTF.nRecords() = 0 then
    msgInfo(errCaption, errText)
    return
  endIf
```

```
    ; Attach a TCursor to the table frame, then read from the TCursor
    ; to fill the array (without UI overhead).
    subAddrTC.attach(subAddrTF)
    scan subAddrTC:
      subAddrAR.addLast(subAddrTC.(addrFldName))
    endScan

    ; Check subAddrAR for invalid addresses.
    if wgLib.verifyAddresses(subAddrAR) then
      msgInfo(okCaption, okText)
    else
      errorShow("Invalid subscriber address")
      return
    endIf
  endMethod
```

See Also getSubscribers

Using Workgroup Methods

To use a workgroup method in your own code, call it from the WGPAL library. Calling a method from the WGPAL library is like calling a method in any other ObjectPAL library. The basic steps are:

1. Declaring Workgroup Methods
2. Declaring Custom Data Types
3. Declaring Workgroup Error Constants
4. Opening the WGPAL Library
5. Calling a Workgroup Method

A template form named WGTMPLT.FSL is installed by default in C:\PDOXWIN\EXAMPLES when you install OBEX. This form has all the workgroup methods, custom data types, and workgroup error constants declared in the form's Uses, Type, and Const windows. Code attached to this form's built-in **open** method opens the WGPAL library. Use this form as a foundation for building your own applications.

See Also

Alphabetic List of ObjectPAL Workgroup Methods

Declaring Workgroup Methods

Declare the methods in a Uses window (typically at the form level). For example, suppose you want to use the custom methods **cancelSubscription** and **enumAccounts**. Put the syntax for each method in the Uses window:

```
Uses ObjectPAL
   cancelSubscription(objectKey String) Logical
   enumAccounts(tableName String, includeSecondary Logical) Logical
EndUses
```

You may find it convenient to declare all the custom methods when you start to develop your application. Workgroup Method Declarations lists these declarations for the custom methods so you can copy them to the Clipboard and paste them directly into your code. Put the code at the form level to make it available to all objects the form contains.

You can also use WGTMPLT.FSL, a template form installed by default in C:\PDOXWIN\EXAMPLES when you install OBEX. This form has all the workgroup methods, custom data types, and workgroup error constants declared in the form's Uses, Type, and Const windows. Code attached to this form's built-in **open** method opens the WGPAL library.

Declaring Custom Data Types

Certain custom methods in the WGPAL library use two custom data types:

- DynArrayOfAny, a DynArray of AnyType values.
- ArrayOfStrings, an Array of String values.

To call a method that uses a custom type, you must declare that type, either in a Type window or in a Type block before the variable declaration section in the method itself.

The following Type block declares the custom data types used by the workgroup methods. You can copy and paste this code into a form's Type window to make the custom types available to all objects in the form.

```
Type
   DynArrayOfAny = DynArray[] AnyType
   ArrayOfStrings = Array[] String
EndType
```

You can also use WGTMPLT.FSL, a template form installed by default in C:\PDOXWIN\EXAMPLES when you install OBEX. This form has all the workgroup methods, custom data types, and workgroup error constants declared in the form's Uses, Type, and Const windows. Code attached to this form's built-in **open** method opens the WGPAL library.

Declaring Workgroup Error Constants

The WGPAL library uses constants to represent errors. If you want to test for these errors in your own code (see Using Workgroup Error Constants), you must declare the constants in your forms. Paste the following code into your form's Const window:

```
Const
   ; Error-code constants
   peWGPdoxWorkgroup        = 1010   ; Workgroup Desktop Error.
   peWGCannotRunForm        = 1020   ; Unable to run Workgroup form.
   peWGCannotConnectToObex  = 1030   ; Unable to connect to OBEX. No DDE
conversation established.
   peWGObexCommandFail      = 1040   ; Unable to execute OBEX command.
   peWGObjectNonExistent    = 1050   ; Object key does not exist in object
store.
   peWGVersionNonExistent   = 1060   ; Object version does not exist in
object store.
   peWGAccountNonExistent   = 1070   ; Account name does not exist.
   peWGObjectKeyExist       = 1080   ; Object key already exists.
   peWGBadObjectKey         = 1090   ; Object key has not been supplied or
is invalid.

   peWGBadVersion           = 1100   ; The version number is outside the
allowable range.
   peWGBadObjectType        = 1110   ; Invalid object type.
   peWGBadObjectDescr       = 1120   ; Object description must be from 1 to
80 characters.
   peWGBadFileSpec          = 1130   ; Invalid file specification.
   peWGCannotCreateFile     = 1140   ; Could not create file.
   peWGCannotOpenFile       = 1150   ; Could not open file.
   peWGCannotAccessFile     = 1160   ; Could not access file.
   peWGCannotExecQuery      = 1170   ; Could not execute stored query.
   peWGNoAnswerTable        = 1180   ; Query does not produce an answer
table.
   peWGCannotCreateTable    = 1190   ; Could not create table.

   peWGCannotOpenTable      = 1200   ; Could not open table.
   peWGCannotAccessTable    = 1210   ; Could not access table.
   peWGCannotFillTable      = 1220   ; Could not fill table.
   peWGCannotCopyTable      = 1230   ; Could not copy table.
   peWGTableStructChanged   = 1240   ; The structure of a table bound to a
Workgroup form has changed.
   peWGAddressProblem       = 1250   ; Addressing problem.
   peWGOutOfContext         = 1260   ; Request not appropriate in this
context.
endConst
```

You can also use WGTMPLT.FSL, a template form installed by default in C:\PDOXWIN\EXAMPLES when you install OBEX. This form has all the workgroup methods, custom data types, and workgroup error constants declared in the form's Uses, Type, and Const windows. Code attached to this form's built-in **open** method opens the WGPAL library.

Using Workgroup Error Constants

To use workgroup error constants in your own code, you must first declare them. See Declaring Workgroup Error Constants. The workgroup error constants are custom ObjectPAL constants; when you use them to test for errors, add the predefined constant UserError to the workgroup error constant. For example,

```
if not wglib.enumSubscriptions(":PRIV:usrSubs") then
   if errorCode() = peWGCannotCreateTable + UserError then
      errorShow("Could not create the table.")
   endIf
endIf
```

For more information about declaring and using custom constants, see the Paradox ObjectPAL documentation.

Opening the WGPAL Library

Before you can call methods from the WGPAL library, you must open it. To open the library, do the following:

1. Declare a Library variable.
2. Call the open method defined for the Library type.

For example, you could put the following code in the form's Var window to declare a Library variable and make it available to all objects in the form.

```
Var
   wgLib Library
EndVar
```

Next, attach the following code (you can copy and paste it if you want to) to the form's built-in **open** method. This code executes and opens the library when the form opens.

```
method open(var eventInfo Event)
   var
      userWgDir, userWinDir, libName,
      winIniSection, winIniVar String
   endVar

   if eventInfo.isPreFilter() then
      ; This code executes for each object on the form.

   else
      ; This code executes only for the form.
      doDefault ; Important: execute the default code first!

      ; Initialize variables.
      libName = "\\WGPAL"
      winIniSection = "PDOXWIN"
      winIniVar = "wgDir"
      userWinDir = windowsDir() ; Get the path to user's WINDOWS directory.

; The path to the user's workgroup directory is stored in a variable in the
PDOXWIN
; section of WIN.INI. The variable's name is wgDir. The following statement
calls
; System::readProfileString to read this variable's value from WIN.INI

      userWgDir= readProfileString(userWinDir + "\\win.ini", winIniSection,
winIniVar)

      if not wgLib.open(userWgDir + libName) then
         errorShow("Couldn't open the WGPAL library.")
      endIf
    endIf
endMethod
```

You can also use WGTMPLT.FSL, a template form installed by default in C:\PDOXWIN\EXAMPLES when you install OBEX. This form has all the workgroup methods, custom data types, and workgroup error constants declared in the form's Uses, Type, and Const windows. Code attached to this form's built-in **open** method opens the WGPAL library.

Calling a Workgroup Method

After you declare the methods, types, and a Library variable and open the WGPAL library, you can call workgroup methods in your own code. For example, the following code is attached to a button's built-in **pushButton** method. It calls the workgroup method **enumPublications** to create a table of the publications stored in your Object Exchange, then opens a table window to display the data.

```
method pushButton(var eventInfo Event)
   var
      tableName String
      myPubsTV  TableView
   endVar

   ; Initialize variables.
   tableName = "mypubs"

   ; Call the workgroup method.
   wgLib.enumPublications(tableName)
   myPubsTV.open(tableName) ; Open the table for viewing.
endMethod
```

Alphabetic List of ObjectPAL Workgroup Methods

Click a method for complete reference information:

cancelSubscription	Cancels a subscription.
deleteObject	Deletes an object from the user's object store.
enumAccounts	Creates a table listing Object Exchange accounts.
enumAlerts	Creates a table listing Object Exchange alerts.
enumPublications	Creates a table listing the Paradox publications in the user's object store.
enumSubscriptions	Creates a table listing the subscriptions in the user's object store.
getAlertDetails	Gets information about a specified Object Exchange alert.
getFileList	Gets a list of the files in a file set.
getNote	Gets the note that was sent with a publication or subscription.
getObexStatus	Fills a DynArray with Object Exchange status information.
getObjectSummary	Fills a DynArray with summary information about an object in the user's object store.
getSubscribers	Fills an array with addresses of subscribers to a specified publication.
issueNewVersion	Issues a new version of an existing Paradox publication.
modifySubscribers	Modifies the subscriber list for a publication in the user's object store.
poll	Makes the Object Exchange poll a specified account.
publish	Creates a new multiversion publication.
saveAs	Saves the last-issued version of an object into a table or directory.
tableRefresh	Refreshes a table with a specified version from the object store.
transfer	Sends an object to subscribers.
verifyAddresses	Verifies one or more subscriber addresses.

Workgroup Method Declarations

The following **Uses . . . endUses** block declares all the custom workgroup methods in the WGPAL library. You can copy and paste this code into a form's Uses window to make the workgroup methods available to all objects in the form. To call one of these methods, you must first open the WGPAL library (see Opening the WGPAL Library).

```
Uses ObjectPAL
   cancelSubscription( objectKey String ) Logical
   deleteObject( objectKey String ) Logical
   enumAccounts( tableName String, includeSecondary Logical ) Logical
   enumAlerts( tableName String ) Logical
   enumPublications( tableName String ) Logical
   enumSubscriptions( tableName String ) Logical
   getAlertDetails( objectKey String, var alertDetails String ) Logical
   getFileList( objectKey String, var fileList ArrayOfStrings ) Logical
   getNote ( objectKey String, backVersion SmallInt, var note String )
Logical
   getObexStatus( var obexStatus DynArrayOfAny ) Logical
   getObjectSummary( objectKey String, var objectSummary DynArrayOfAny )
Logical
   getSubscribers( objectKey String, var subscribers ArrayOfStrings )
Logical
   issueNewVersion( objectKey String, note String ) Logical
   modifySubscribers( objectKey              String,
                      var subscribers        ArrayOfStrings,
                      distributeLastVersion  Logical ) Logical

   poll( accountName String ) Logical

   publish( objectKey        String,
            objectType       String,
            description      String,
            var fileList     ArrayOfStrings,
            var subscribers  ArrayOfStrings,
            versionDepth     SmallInt,
            note             String )   Logical

   saveAs(  objectKey         String,
            backVersion       SmallInt,
            targetObjectName  String,
            fillTableNow      Logical)   Logical

   tableRefresh(  objectKey          String,
                  backVersion        SmallInt,
                  fileName           String,
                  var dateLastTaken  Date,
                  var timeLastTaken  String ) Logical

   transfer(   objectKey        String,
               objectType       String,
               description      String,
               var fileList     ArrayOfStrings,
               var subscribers  ArrayOfStrings,
               note             String) Logical
```

```
   verifyAddresses( var addresses ArrayOfStrings ) Logical
endUses
```