



ProAdmin API Users' Guide



Last updated: 8/20/2025

This document was prepared to assist users of Winklevoss Technologies' ProAdmin System; its contents may not be used for any other purpose without written permission. The material contained herein is supplied without representation or warranty of any kind. Winklevoss Technologies, LLC therefore assumes no responsibility and shall have no liability arising from the supply or use of this document or the material contained herein.

Copyright © 2025 Winklevoss Technologies, LLC
Printed in the United States of America. All rights reserved.
Unauthorized reproduction is strictly prohibited.

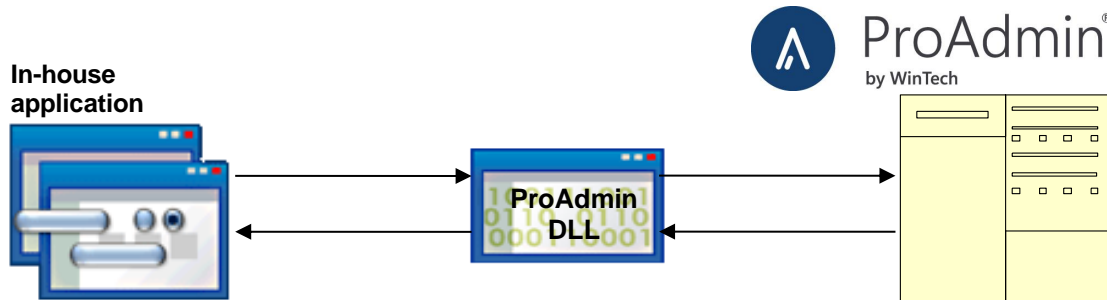
Windows®, Visual Basic® and Access® are registered trademarks of Microsoft Corporation.

Contents

- OVERVIEW..... 4**
 - Licensing Requirements4
- API SETUP (32-BIT) 5**
 - Registration of ProAdmin API5
 - Calling the ProAdmin API7
- API SETUP (64-BIT) 8**
 - Registration of ProAdmin API8
 - Calling the ProAdmin API 10
 - Disposing of the PVAPI64 Instance 11
- API FUNCTIONS12**
 - System Related 12
 - FileClose 12
 - FileOpen 13
 - ListLibraries..... 15
 - QueryLibrary 16
 - Administration Factors 18
 - GetRUNADFACTARGS 18
 - RunADFACT 18
 - Error\Warning Messages 21
 - GetMessage..... 21
 - SetMessage..... 22
 - Appendix A: Errors 24

Overview

ProAdmin API is designed to expose certain functions in ProAdmin to other programs via an apartment threaded ActiveX DLL. The API facilitates seamless integration of ProAdmin within existing in-house applications allowing for greater automation of repetitive functions.



ProAdmin API may be configured to use a local or a network installation of ProAdmin. This guide assumes that ProAdmin is already set up correctly – for instructions on setting up ProAdmin, refer to the ProAdmin installation guide (readme.doc) in the ProAdmin installation folder.

Licensing Requirements

A ProAdmin license server (for details see the ProAdmin License Server Guide in the ProAdmin installation guide) or site license must be available for the API to function.

API Setup (32-bit)

Registration of ProAdmin API

- The ProAdmin API is now ready to be registered. Any registration statements must be executed with "administrator rights" on the computer. This guide uses Visual Basic to provide examples; the procedure using other languages is similar.
- Register PVAPI.DLL using the REGSVR32 command.

Examples:

```
regsvr32 "C:\Program Files\Wintech\ProAdmin\PAAPI.dll"
```

```
regsvr32 "N:\Apps\PAAPI.dll"
```

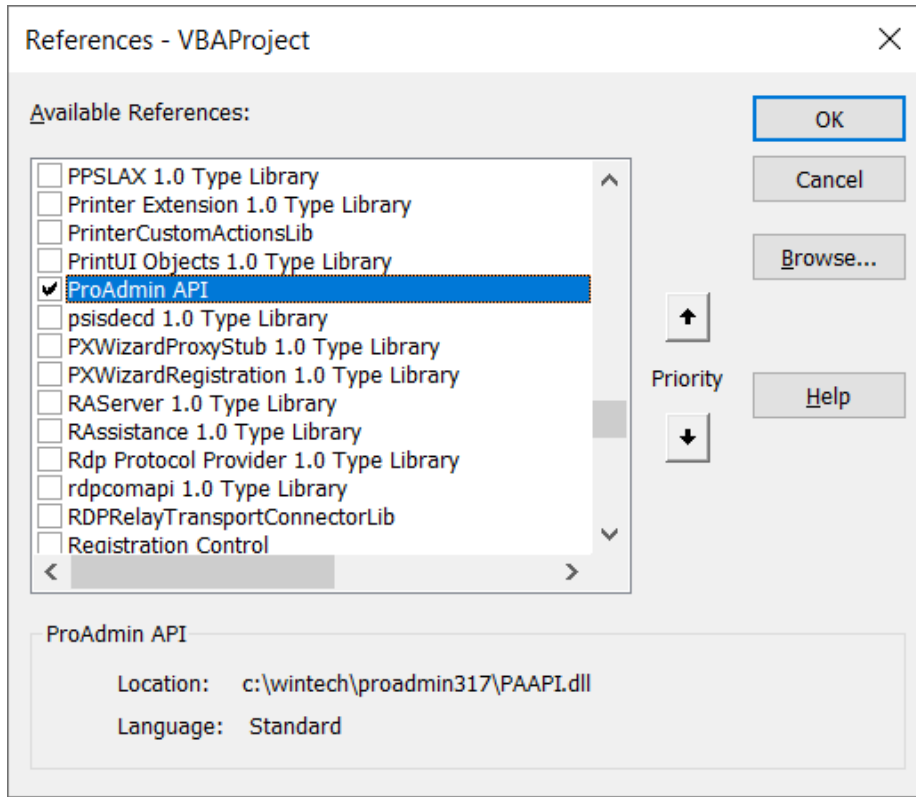
- Register the ProAdmin application for usage by the ProAdmin API. Specify the ProAdmin.exe and ProAdmin.ini to be used by the ProAdmin API. The folder containing the ProAdmin.ini file will be considered the "user" directory and must allow "write" access.

Examples:

```
"C:\Program Files\Wintech\ProAdmin\ProAdmin.exe" "C:\ProAdminUser\ProAdmin.ini" /regserver
```

```
"N:\Apps\ProAdmin.exe" "C:\ProAdminUser\ProAdmin.ini" /regserver
```

- **DEVELOPERS ONLY** - From the list of available reference libraries select "ProAdmin API", as shown below:



- ProAdmin API is now ready to be called from other applications.

Calling the ProAdmin API

The API contains one main class - PAAPI. Every function exposed by the API is called in the following manner:

- ProAdmin functions are exposed using only one PAAPI method, PACall, which accepts two arguments:
 - [0] The ProAdmin function name (case-insensitive String). Examples of such function names would be FILEOPEN, FILECLOSE, etc.
 - [1] Parameter List (Variant). NOTE: If parameters are required, all arguments must be supplied for the functions to operate correctly.
- PACall always returns a Variant Array.

A generic Visual Basic code snippet may resemble the following:

```
Dim eng as New PAAPI
Dim RetVal as Variant
Dim xArgList()
xArgList = Array(param1,param2,1,0)
RetVal = eng.PVCall("ProAdminFuncName", xArgList)
Dim ErrCode as Long, ErrDesc as String
ErrCode = RetVal(0)
ErrDesc = RetVal(1)
```

Of course, it is reasonable to assume that the eng object defined above will typically be parked in a module and available publicly in production applications, so that different instances are not invoked for each function call.

API Setup (64-bit)

Registration of ProAdmin API

- The ProAdmin API is now ready to be registered. Any registration statements must be executed with "administrator rights" on the computer. This guide uses Visual Basic for Applications to provide examples; the procedure using other languages is similar.
- You may need to register the ProAdminCo.dll using the REGSVR32 command:

```
regsvr32 "C:\Program Files\WinTech\ProAdmin\ ProAdminCo.dll"
```

- Register PVAPI.DLL using the REGASM command.
- Navigate to the Framework64 folder by enter the following:

```
CD "C:\Windows\Microsoft.NET\Framework64\v4.0.30319\"
```

Examples:

```
RegAsm.exe /codebase "C:\Program Files\WinTech\ProAdmin\PAAPI64.dll" /tlb
```

```
RegAsm.exe /codebase "N:\Apps\PAAPI64.dll" /tlb
```

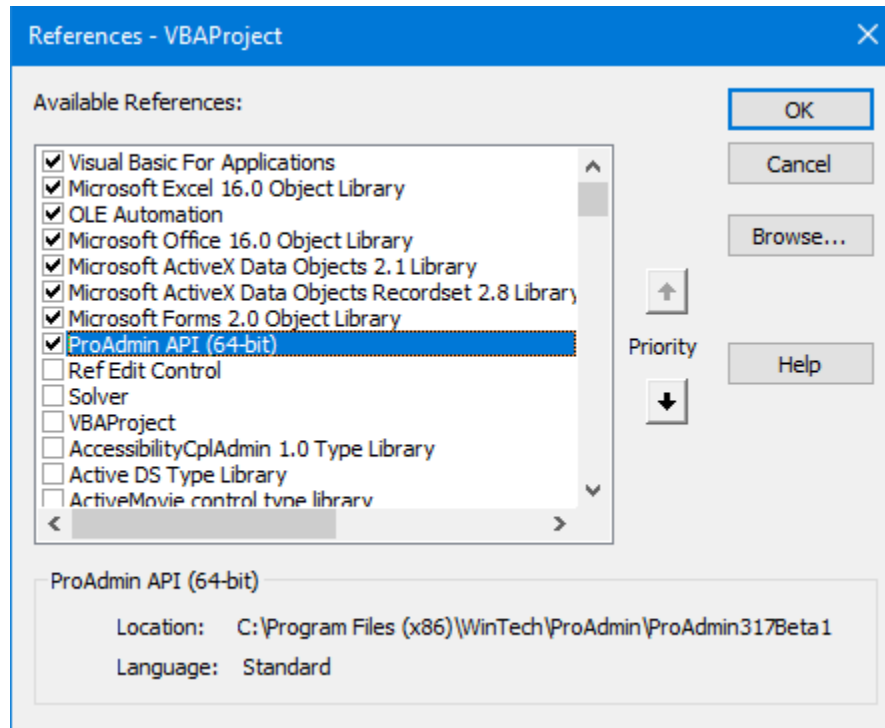
- Register the ProAdmin application for usage by the ProAdmin API. Specify the ProAdmin.exe and ProAdmin.ini to be used by the ProAdmin API. The folder containing the ProAdmin.ini file will be considered the "user" directory and must allow "write" access.

Examples:

```
"C:\Program Files\WinTech\ProAdmin\ProAdmin.exe" "C:\ProAdminUser\ProAdmin.ini" /regserver
```

```
"N:\Apps\ProAdmin.exe" "C:\ProAdminUser\ProAdmin.ini" /regserver
```


- **DEVELOPERS ONLY** - From the list of available reference libraries select "ProAdmin API", as shown below:



- ProAdmin API is now ready to be called from other applications.

Calling the ProAdmin API

The API contains one main class – PAAPI64. Every function exposed by the API is called in the following manner:

- ProAdmin functions are exposed using only one PAAPI64 method, PACall, which accepts two arguments:
 - [0] The ProAdmin function name (case-insensitive String). Examples of such function names would be FILEOPEN, FILECLOSE, etc.
 - [1] Parameter List (Variant). NOTE: If parameters are required, all arguments must be supplied for the functions to operate correctly.
- PACall always returns a Variant Array.

A generic Visual Basic code snippet may resemble the following:

```
Dim eng as New PAAPI64.PAAPI64
```

```
' note the extra .PAAPI64 for the 64-bit version
```

```
Sub FuncName()
```

```
Dim RetVal as Variant
```

```
Dim xArgList()
```

```
xArgList = Array(param1,param2,1,0)
```

```
Set eng = New PVAPI64.PVAPI64
```

```
' note the above line should be before the first use of eng
```

```
RetVal = eng.PVCall("ProAdminFuncName", xArgList)
```

```
Dim ErrCode as Long, ErrDesc as String
```

```
ErrCode = RetVal(0)
```

```
ErrDesc = RetVal(1)
```

```
End Sub
```

Of course, it is reasonable to assume that the eng object defined above will typically be parked in a module and available publicly in production applications, so that different instances are not invoked for each function call.

Disposing of the PAAPI64 Instance

Disposing of the ActiveX object by calling the Dispose method.

A generic Visual Basic code snippet may resemble the following:

```
eng.Dispose
```

```
Set eng = Nothing
```

Note, closing your program will automatically dispose of the object as well.

API Functions

System Related

FileClose

Description:

Closes the current client folder.

Arguments

<None>

Return Value (Variant Array)

[0]	Error code (long)
[1]	Error description (string)

Visual Basic Code Snippet

```
Dim RetVal As Variant
RetVal = eng.PVCall("FILECLOSE")
' Now process return values
Dim ErrCode as Long, ErrDesc as String
ErrCode = RetVal(0)
ErrDesc = RetVal(1)
```

FileOpen

Description:

Opens a ProAdmin client folder. Closes current one if one currently open, regardless of whether the new file was successfully opened.

Argument (Variant Array)	
[0]	UpdateFolderIfNecessary (long: 1 = yes, 0 =No)
[1]	ClientFolder – Path to folder (string)
[2]	ComputationMode (long) Code that indicates ProAdmin's computation mode: <ol style="list-style-type: none">1. = U.S. Qualified Pension,2. = Retiree Medical,3. = Public Pension,4. = Nonqualified,5. = Canadian Pension,6. = German Pension,7. = United Kingdom Pension
[3]	OpenMode (optional; string) – If specified, a string that describes the manner in which to open the client. Valid values are: 'MUSTWRITE' open client normally only; do not open as "read only" if client is in use (default) 'READWRITE' try to open client normally first, but then open as "read only" if client is in use 'READONLY' open client as "read only"

Return Value (Variant Array)	
[0]	Error code (long)
[1]	Error description (string) Notable Errors (also see Appendix A): 31 = Client Folder doesn't exist 32 = Client Folder is an invalid path 33 = Client Folder isn't a valid ProAdmin client folder 34 = Client Folder in use 35 = Client Folder needed updating, but UpdateFolderIfNecessary is False 36 = Unknown error opening client 37 = Client Folder has been updated by a newer version of ProAdmin

Visual Basic Code Snippet

```
Dim eng as New PAAPI64 ' This only has to be done once in a module

Sub FileOpen()
Dim RetVal As Variant
Dim FilArr()
'      Open Client in U.S. Qualified Mode, with READWRITE access,
'      and update if necessary
FilArr = Array(1, "C:\MyClient", 1, "READWRITE")
RetVal = eng.PVCall("FILEOPEN", FilArr)
' Now process return values
Dim ErrCode as Long, ErrDesc as String
ErrCode = RetVal(0)
ErrDesc = RetVal(1)
End Sub
```

ListLibraries

Description:

Lists all available ProAdmin libraries.

Arguments

<None>

Return Value (Variant Array)

[0]	Error code (long)
[1]	Error description (string)
[2]	List of short library names - used as argument in QueryLibrary (string array)
[3]	List of descriptive library names (string array)

Visual Basic Code Snippet

```
Dim RetVal As Variant
RetVal = eng.PVCall("LISTLIBRARIES")
' Now process return values
Dim ErrCode as Long, ErrDesc as String
ErrCode = RetVal(0)
ErrDesc = RetVal(1)
' Check for errors before continuing
If RetVal(0) = 0 Then
    ' Declare arrays to contain long and short library names
    Dim ShName(), DescName() As String
    ShName = RetVal(2)
    DescName = RetVal(3)
    ' View each entry using a messagebox
    For i = LBound(ShName) To UBound(ShName)
        MsgBox ts.WriteLine ShName(i) + " | " + DescName(i)
    Next
End If
```

QueryLibrary

Description:

Returns information about all the entries in a library. Use ListLibraries for a list of short and full library names.

Arguments

Short Library Name (e.g., "RECLAYOUT", as returned by ListLibraries).

Return Value (Variant Array)

[0]	Error code (long)
[1]	Error description (string)
[2]	List of components/fieldnames (string array); message type (1=warning, 2=error) for the Message Definitions Library (long array)
[3]	List of descriptions (string array)
[4]	List of modification timestamps (string array)
[5]	ID# of element

Visual Basic Code Snippet

```
DimRetVal As Variant
RetVal = eng.PVCall("QUERYLIBRARY", "RECLAYOUT")
' Now process return values
Dim ErrCode as Long, ErrDesc as String
ErrCode = RetVal(0)
ErrDesc = RetVal(1)
' Check for errors before harvesting data
If RetVal(0) = 0 Then
    Dim FldArr(), DescArr(), TimeStampArr(), IDNumArr() As Variant
    FldArr = RetVal(2)
    DescArr = RetVal(3)
    TimeStampArr = RetVal(4)
    IDNumArr = RetVal(5)
    ' Parse the entries
    For i = LBound(FldArr) To UBound(FldArr)
        MsgBox ts.WriteLine FldArr(i) + " | " + DescArr(i) + _
            " | " + TimeStampArr(i) + " | " + IDNumArr(i)
    Next
End If
```


Sample Output (separated by |)

```
FAEBEN | 2%*SVC*FAE | 10/24/1994 13:41:57 | 4
INTCASHBAL | Integrated, age graded, cash balance | 10/24/1994
14:50:15 | 3
GROSSBEN | Gross Benefit | 10/24/1994 13:42:10 | 2
PIA | Social Security Offset | 10/24/1994 13:42:51 | 1
PROJBEN1 | Project & Prorate benefit, piece 1 | 10/24/1994
13:43:02 | 5
PROJBEN2 | Project & Prorate Benefit, piece 2 | 10/24/1994
13:43:13 | 6
```

Administration Factors

GetRUNADFACTARGS

Description:

Returns the default values for an RunADFACT call based on the named Administration Factor Tool library entry (from the opened client).

Note: This function is only available via the API; it is not used by the ProAdmin interface.

Arguments

[0]	Name of Administration Factor (string)
------------	---

Return Value (Variant Array).

[0]	Error code (long)
[1]	Error description (string) Notable Errors (also see Appendix A): 84 = Message Definition does not exist
[2]	RunADFACT vector (variant). See RunADFACT arguments in its documentation below.
[3]	Empty string or vector of payment form names
[4]	Empty string or name of the normal form
[5]	Empty string or vector of valid commutative function names

RunADFACT

Description:

Returns annuity factor(s), conversion factor(s), or commutation function for the named Administration Factor (that exists in the client's Administration Factor Tool Library).

Note: This function is only available via the API; it is not used by the ProAdmin interface.

Arguments

[0]	Administration Factor name (string) The name of the Administration Factor (in the Administration Factors Library) used to calculate annuity factors, conversion factors, or commutation functions. {i.e., the assumptions}
[1]	Calculation Type (long): 1=annuity factors; 2=conversion factors; 3=commutation functions.

	Note: If you request a calculation type that the Administration Factor wasn't setup for, then you'll get an appropriate error message.
[2]	Annuity Factor or commutation function name (string). For an annuity factor calculation type, annuity factors will be returned for the named annuity factor (which must be defined in the named Administration Factor). For the conversion factor type, conversion factors will be returned for this named annuity factor using the Normal Form specified in the Administration Factor. For a commutation functions, it must be one of these names: Dx, Dy, Dxy, ,Nx, Ny, Nxy, ,N(m)x, N(m)y, N(m)xy, ,Sx, Sy, Sxy, ,Cx, Cy, Cxy, ,Mx, My, Mxy, ,Rx, Ry, Rxy, ,ex, ey, or exy; where x indicates primary, y indicates beneficiary, and xy indicates joint life.
[3]	Participant age(s) (integer, long, or double scalar, list (i.e., vector), matrix of ages or blank). Blank indicates that you want to use the age settings in the named Administration Factor. Note that the age participant age settings in the library (see [0]) define the valid age range and Include Monthly ages defines whether fractional ages are allowed.
[4]	Participant sex (integer, long, or double scalar, list (i.e., vector), or matrix of sex codes: 0 = male; 1=female). If the Participant age is a scalar, then this must be a scalar. If the Participant age is a list, then this can be a scalar (i.e., use the same sex for all ages) or it must be a list with the same number of values as there are participant ages. If the Participant age is a matrix, then this can be a scalar (use the same sex for all ages) or it must be a matrix with the same number of rows and columns as there are for participant ages.
[5]	Beneficiary age(s) (integer, long, or double scalar, list (i.e., vector), matrix of ages, or blank when this parameter is not used). If the Participant's age is a scalar, then this can be a scalar, list or matrix. If the Participant's age is a list or a matrix, then this can be a scalar (use the same beneficiary age for all participant ages) or if it's a list/matrix, it must have the exact shape and number of values as the participant ages. Note that the beneficiary age settings in the library (see [0]) define the valid age range and Include Monthly ages defines whether fractional ages are allowed.
[6]	(not used , the opposite of participant's sex will be assumed; this will be Beneficiary sex ; currently the only valid value is "", MV, vbNULL, vbEMPTY)
[7]	Interest rate(s) (scalar, list, or blank when you want to set the settings in the named Administration Factor) If this is a scalar, then it's a static rate (double). If it's a list, then the first element is a code (integer): 1=static rate, followed by a single rate (double); 2=variable forward rates, followed by an even number of duration (integer)/rate (double) pairs; 3=PBGC style rates, followed by 4 rates (doubles), the <i>immediate rate</i> , the <i>prior 7 years (K1)</i> , the <i>prior 8 years (K2)</i> , and the <i>all prior years (K3)</i> ; 4=segment-style rate, followed by 3 segment rates (doubles); 5=variable spot rates, followed by an even number of duration (integer)/rate (double) pairs.
[8]	Calculation year (integer or blank when you want to set the settings in the named Administration Factor) The calculation year is required for age by year of birth, dynamic and fully generational mortality tables.

[9]	not used (this will be the mortality assumptions; currently the only valid value is "", MV, vbNULL, vbEMPTY)
------------	---

Return Value (Variant Array)

[0]	Error code (long)
[1]	Error description (string) Notable Errors (also see Appendix A): 84 = Name: <i>xyz entry does not exist</i> 98 = parameter validation errors, e.g., Invalid Administration Factor name Administration Factor "xyx" has not been setup to handle annuity factors Invalid commutative function: <i>ddd</i> No payment forms are defined At least 2 payment forms must be defined to perform a Conversion Factor calculation. Requested payment form <i>pfname</i> is not defined in this Administration Factor Conversion factor requested but normal form has not been specified The primary's age must fall within the range <i>nn</i> to <i>mm</i> Monthly primary ages are NOT allowed by the Administration Factor library: <i>xyx</i>
[2]	Participant age(s) (integer, long, or double scalar, list (i.e., vector), or matrix of ages). Will be empty if participant age wasn't needed in the calculation.
[3]	Participant sex (integer, long, or double scalar, list (i.e., vector), or matrix of sex codes: 0 = male; 1=female). Will be empty if participant sex wasn't needed in the calculation.
[4]	Beneficiary age(s) (integer, long, or double scalar, list (i.e., vector), or matrix of ages). Will be empty if beneficiary age wasn't needed in the calculation.
[5]	Beneficiary sex (integer, long, or double scalar, list (i.e., vector), or matrix of sex codes: 0 = male; 1=female) Will be empty if beneficiary sex wasn't needed in the calculation.
[6]	Administration factors or commutation function results (double scalar, list or matrix)

Error\Warning Messages

GetMessage

Description:

Returns information about the Message Definitions in the Message Definition Library.

Note: This function is only available via the API; it is not used by the ProAdmin interface.

Arguments

[0]	Name of Message Definition (string)
[1]	Name of Message Type (long): 1=warning, 2=error

Return Value (Variant Array).

[0]	Error code (long)
[1]	Error description (string) Notable Errors (also see Appendix A): 84 = Message Definition does not exist
[2]	Code for Output (string)
[3]	Message Type (long) 1 = Warning; 2 = Error (stop processing)
[4]	Message text (string)

Visual Basic Code Snippet

```
Dim RetVal as Variant  
RetVal = eng.PVCall("GETMESSAGE", "MYPLAN", 2)
```

SetMessage

Description:

Save updated Message to the Message Definition Library.

Note: This function is only available via the API; it is not used by the ProAdmin interface.

Argument (Variant Array)	
[0]	Name of original Message Definition (string)
[1]	Message type of original Message Definition (long): 1=warning, 2=error
[2]	New Message Definition name for "Save As" or empty string ("") to "Replace" in Message Definition library (string)
<hr/>	
[3]	Message values. Number of values must match the existing number of values. See GetMessage (vector)
[4]	Log file name (string) Default log file name is PROADMINAPI1.TMP, PROADMINAPI2.TMP, etc.
[5]	Error Code Overrides - Error codes separated by spaces under which the user wants processing to continue instead of aborting (string). Alternatively, an empty string must be used if the intention is to halt processing on all errors. NOTE: Errors cannot be overridden unless specified.
<hr/>	
Return Value (Variant Array)	
[0]	Error code (long)
[1]	Error description (string) Notable Errors (also see Appendix A): 311 = Running this command will erase the results of other objects (can override and writes to message log)

Note: You must always specify the **name** and **type** of an existing Message Definition, even if you are creating a new Message Definition. This is to ensure that the newly created message will have the same structure as the other messages in the Message Library.

Visual Basic Code Snippet

```
Dim MsgInfo, RetVal as Variant

' Get the message info for the message named MyMsg
MsgInfo = eng.PVCall("GETMESSAGE","MyMsg", 1)

If 0 = MsgInfo(0) Then ' no error (i.e., we got the info)
    ' We successfully retrieved the info for MyMsg, so let's
    ' change every " the " to " THE " in the message text

    Dim MsgTxt as string
    MsgTxt = Replace(MsgInfo(4), " the ", " THE ")
    If (0 <> StrComp(MsgTxt, MsgInfo(4))) Then ' message text changed?
        ' update the original message with the new message text
        Dim NewInfo(5) ' 6 elements, 0 thru 5

        NewInfo(0) = "MyMsg" ' name of the original message
        NewInfo(1) = 1 ' message type of the original message
        NewInfo(2) = "" ' empty (we're not renaming the message)
        NewInfo(3) = Array(MsgInfo(2), MsgInfo(3), MsgTxt) ' message info
        NewInfo(4) = "" ' empty or message log file name
        NewInfo(5) = "" ' empty or errors that are OK to keep processing

        RetVal = eng.PVCall("SETMESSAGE", NewInfo) ' update MyMsg
    End If
End If
```

In the above code snippet, if the message we are modifying was used in a saved calculation, the message would NOT have been updated, and you would have gotten an error: **311 Running this command will erase the results of other objects.**

If you want the message to be updated (and don't mind that the associated calculation results are erased), set the 5th element to "311" (e.g., NewInfo(4) = "311"). Now, when you run the code snippet, this error will be ignored, the message will be updated, and the associated calculation results will be erased.

Appendix A: Errors

Errors that apply to the ProAdmin API functions are listed below:

Error Codes and Descriptions	
-1	Invalid function name. Verify that the function name (passed as the first argument) in the Pvcall function is spelt correctly.
-2	ProAdmin API Licensing Requirements not met. Ensure that the API is adequately licensed to perform the function.
-3	Internal API Error. Contact ProAdmin Support for assistance.
-4	ProAdmin update available but not applied. Manually apply ProAdmin update to continue (refer to the ProAdmin Installation Guide readme.doc in the ProAdmin Installation Folder for details).
0	Success.
10	Incorrect number of arguments.
11	No client folder currently open.
13	Invalid boolean argument (plus further message).
14	Invalid file name specification.
17	Invalid code.
18	File already exists, quit without writing to it.
19	File is read-only or in use.
22	Invalid specification of field names.
23	Invalid library name.
24	File already exists. Enter a new name.
31	Client Folder doesn't exist.
32	Client Folder is an invalid path.
33	Client Folder isn't a valid ProAdmin client folder.
34	Client Folder in use.
35	Client Folder needed updating, but UpdateFolderIfNecessary is False.
36	Unknown error opening client.
37	Client Folder has been updated by a newer version of ProAdmin.
38	Rights for File Open must be specified as MUSTWRITE, READWRITE or READONLY.

63	Invalid date.
84	"Library name" entry does not exist
85	Incorrect number of items for library entry
86	New name for "Library name" is already in use
99	Unknown error. Contact ProAdmin Support for assistance.
121	Library Entry error (writes to message log).
303	Ambiguous coded field values that are similar to existing values (can override and writes to message log).
311	Running this command will erase the results of other objects (can override and writes to message log).