Apple® IIGS

Undo Manager Reference Manual

Dictionary

better safe than sorry

better safe than sorry

phrase of better

1. *proverb*
   it's wiser to be cautious and careful than to be hasty or rash and so do something you may later regret.

Dedicated to the memories of Joe Kohn 1947-2010
and Ryan Suenega 1967-2011

My thanks go to Eric Shepherd for some excellent suggestions, and especially
to Chris Vavruska, for his extensive help and many suggestions,
as well as helping me with the C and Pascal documentation, and also for
providing me with C, Modula-2, GSoft, and Pascal examples of the
Undo.Tester.

Contents

Introduction to the Undo Manager

Undo Manager

When I first started writing Spectrum™ back in 1986, I needed to have a text editor, and with no TextEdit Tool available, I had to start writing my own editor. I had not got very far, before I was delighted to see in an System update, that a nice new TextEdit Tool had been added that instantly solved my problem. Over the years, I have used the TextEdit tool extensively, and without it, many of the Spectrum™ scripting functions would not be possible.

Powerful as TextEdit is, if you check the many flags and settings in the documentation, it is clear that TextEdit was never fully implemented as was originally intended. Most of these omissions are of minor inconvenience, but anyone who is used to editing text on other computer platforms, will find some glaring omissions from the usual editing trio of Cut, Copy and Paste. There are no active Undo, Clear, or Select All functions, despite Undo and Clear appearing on the standard Edit menus.

Many programmer's like myself, will probably have activated 'Select All' from within their own code, but 'Undo' and 'Clear', are usually left hanging in mid air doing nothing.

If like me, you have typed or edited large sections of text in a TextEdit or LineEdit Control, then accidentally deleted some of the text, you will have been frustrated when selecting 'Undo' to restore the mistake, nothing happened.

The Undo Manager resolves this dilemma for you, and by installing the Undo Manager, and adding a few simple commands to your application code, along with adding Redo, Select All, and Clear to the standard five items of the Edit menu, the Undo, Redo, Clear, and Select All menu items will all now magically become active.

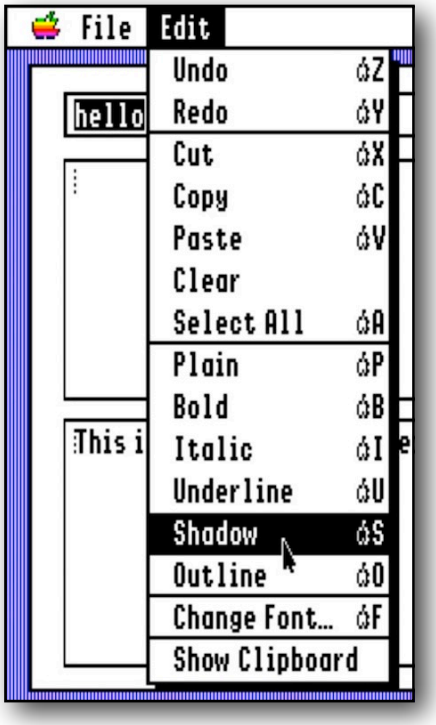
Using the Undo Manager

Requirements

The Undo Manager requires some Tools to be active, though in most applications, these will already have been started. If not, make sure they are started from your application before calling the Undo Manager:

Resource Manager, File Manager, Menu Manager, Control Manager, Event Manager, Memory Manager, Integer Math Tool Set, Miscellaneous Tool Set, Scrap Manager, LineEdit Tool Set, TextEdit Tool Set, WindowMgr, and Undo Manager

The Undo Manager itself should be started after the application has started the above Tools, and any other Tool Sets that the application requires.



To use the Undo Manager, an application only needs to have a standard Edit Menu with the Undo, Redo, Cut, Copy, Paste, Clear, and Select All, menu items present. After Login to the Undo Manager, when a window is then opened, and uses the doModalWindow command to handle its events, by using two simple Tool calls, the Undo Manager will handle those seven Edit Menu items for any LineEdit, TextEdit or Picture controls within that window.

Up to eight individual Logins, from multiple Applications or Windows, can use the Undo Manager at any one time. Up to thirty-two LineEdit, TextEdit, Picture controls from across those eight Logins are individually controlled, with up to thirty-two Undo and Redo levels for each control or Object.

A simple application, 'Undo.Tester', and its source, are included, to show how the Undo Manager works in practice.

'Undo.Tester' contains two basic word processors that you might find useful for simple text processing or testing.

Programming

How the Undo Manager works

To activate the Undo command for any LineEdit, TextEdit, or Picture controls within your application, the controls in most cases should be within windows handled by the doModalWindow call. This will probably be the case for most modern applications. On Login to the Undo Manager, the Manager will temporarily change the seven Edit Menu item IDs from their defaults, to ones you have supplied. This means they are isolated from the normal Edit keypresses handled by the Scrap Manager, and then from within the doModalWindow loop, the Undo Manager will trap and handle those keypresses and Menu selections for the seven Menu items, as if the System had handled them.

The Undo Manager maintains its own private clipboard for each of your LineEdit, TextEdit, or Picture controls. Before text or a picture is changed by OA-X, Clear, Delete, or OA-V, a private Clip will be kept of the entire text from that control, and in the case of a TextEdit control, any styles that may have been used. The Undo menu item will then be activated. When the OA-Z key is pressed, or you select the Undo menu item, the Undo Manager will then restore the text or picture to how it was before those Edit keys were pressed, and any selection that had been active will be restored. If you have pressed OA-Z, then the Redo menu item will become active, and by pressing OA-Y, or selecting the Redo menu item, it will Redo the last Undo action. Up to thirty-two levels of Undo and Redo can be stored for each control.

If you have a standard speed IIgs, you may find a small slowdown when the OA-Z, OA-Y, OA-X, Clear, Delete, or OA-V keys are pressed, especially if you are working with a very large amount of text in a TextEdit control. For shorter amounts of text, in either a LineEdit or TextEdit control, you should not see any slowdown at all.

If the "No editing allowed" or "User cannot select text" flags are set for a TextEdit control, the related Edit menu items will be dimmed.

The private Clips are normally kept within memory, but as TextEdit controls can hold very large quantities of text, if a TextEdit “clip” or ObjectHandle is larger than 4K, it will be saved as a temporary file, in Teach format for text, to the “Undo.Data” folder in the System folder, rather than be kept in memory. If there is insufficient space on the hard disk to do this, the clip will not be saved, and the Undo command will not be activated.

Only thirty-two levels of Undo and Redo are kept on the Clips stack, so the Tool can only restore the last thirty-two changes that were made. As new clips are saved, older ones will be rolled off that stack. Please refer to the “Using the Calls” section for more detail.

Using the Calls

Please refer to the section on Coding Examples for a more detailed explanation of how the calls are integrated into your code.

Basic Strategy:

These are the sequence of calls you will need to use the Undo Manager:

```
At application start:
     LoadOneTool          ; Load the Undo Manger into memory
     UMStartUp            ; Start the Undo Manager
     UMLogIn              ; LogIn to the Undo Manager
In a doModalWindow call Loop:
     UMKeyAction          ; Called from within the doModalWindow EventHook
     UMMenuAction         ; Called from within the main doModalWindow loop
At application Quit:
     UMLogOut             ; LogOut from the Undo Manager
     UMShutDown           ; Shut the Undo Manager down
     UnloadOneTool        ; Unload the Undo Manager from memory
```

The advantages of this basic strategy is that you can hold saved Clips for up to thirty-two different controls across your application. It is important however that all the windows containing those controls are kept open, as the ControlHandle value may change if windows are closed, and then re-opened later. This would mean that saved Clips could no longer be matched up to the same control if its window was re-opened later on.

To avoid saved Clips from becoming orphaned with this strategy, call `UMClearWindowClips` just before a window is closed.

Optional calls (Refer to each call for more details of what the calls do):

```
     UMVersion            ; Returns the version number of the Undo Manager
     UMStatus             ; Returns the Status of the Undo Manager
     UMSaveClip           ; Manually saves a Clip from the current active control
     UMClearClips         ; Clears all the Clips for the current active control
     UMClear              ; Clears all the Clips for the passed refID
     UMClearWindowClips ; Clears all the clips for the front window
     UMLogOutAll          ; LogOut, and clears Clips, for all the applications
                            or windows
     UMClipStatus       ; Returns data about the Clips stack for a control
     UMKeyAction2       ; Alternative call to UMKeyAction
     UMPushUndoObject   ; Saves a supplied Object data Handle to the Undo stack
     UMPopUndoObject    ; Retrieves an Object data Handle from the Undo stack
```

Alternate Strategy:

This alternate strategy can be used:

```
At application start:
     LoadOneTool         ; Load the Undo Manager into memory
     UMStartUp           ; Start the Manager Tool

After each window has been opened:
     UMLogIn             ; LogIn to the Manager Tool

In a doModalWindow call Loop:
     UMKeyAction         ; Called from within the doModalWindow EventHook
     UMMenuAction        ; Called from within the main doModalWindow loop

Before each window is closed:
     UMLogOut            ; LogOut from the Undo Manager

At application Quit:
     UMShutDown          ; Shut the Undo Manager down
     UnloadOneTool       ; Unload the Undo Manager from memory
```

This strategy makes sure that all saved Clips are cleared whenever a window is closed. If your application has a large number of controls, this will ease memory use from the space required to save a multitude of Clips.

To sum up:

Up to thirty-two Clips can be saved for each of thirty-two active controls, or data Object. If more than thirty-two Clips are saved, older Clips will be lost as they are rolled off from the bottom of the Clips stack for that control.

You can only call `UMLogIn` eight times in total, with up to thirty-two active controls. You should leave at least one `UMLogIn` free for any NDA that may also wish to call the Tool.

If you use `UMLogIn` at the start of your application, and `UMLogOut` at Quit, you need to call either `UMClear` or `UMClearWindowClips` before each doModalWindow window is closed. The saved Clips are control specific, so to minimise memory use, and to clear memory, one of these calls needs to be made when the clips are no longer available.

Note: Only the picture content from a Picture control is cleared or replaced. The control itself will not change. If the picture being changed is of a different size to the original picture, it will be resized to fit the control. This may produce unexpected results.

Passed Parameters:

A `refID` needs to be passed for many of the calls. This can be either the application UserID, or a new ID created using GetNewID. The `refID` is used to identify which application, NDA or window has logged in and is making the call, amongst the eight possible `UMLogIn` calls. It is permissible for an application to make multiple calls to `UMLogIn` using unique `refIDs`, thus being able to handle multiple windows as if they were from different applications. This has the advantage of being able to use different menus for each call. If you do this, just make sure every `UMLogIn` is balanced with an `UMLogOut` call.

A pointer to a `DataTable` in your application is passed at `UMLogIn`. The DataTable is constructed thus:

```
DataTable entry
          dc    i4'EventRecord'  long     Pointer to your EventRecord
          dc    i'$01FA,$00FA'   word     New/Old Undo Menu Item ID = OA-Z
          dc    i'$01FB,$00FB'   word     New/Old Cut Menu Item ID = OA-X
          dc    i'$01FC,$00FC'   word     New/Old Copy Menu Item ID = OA-C
          dc    i'$01FD,$00FD'   word     New/Old Paste Menu Item ID = OA-V
          dc    i'$01FE,$00FE'   word     New/Old Clear Menu Item ID
          dc    i'$0100,$0100'   word     New/Old Select All Menu Item ID = OA-A
          dc    i'$0101,$0101'   word     New/Old Rdedo Menu Item ID = OA-Y
```

The `EventRecord` pointer must point to an extended `EventRecord` in your application code (Refer to Coding Samples for more detail).

The seven MenuIDs follow in pairs. The first ID of each is the new MenuID that the Undo Manager will use to replace the existing MenuID or current MenuID that your application has defined. The second ID is the existing or default ID defined in your application. If you define in your code the five Edit MenuIDs as custom MenuIDs, that are outside the $FA to $FE range, the two values can be of the same value if you wish. Just make sure that the basic five Edit Menu IDs of $FA to $FE are not used for the new MenuIDs that the Undo Manager will allocate. This is to make sure that the Scrap Manager will not trap those keys before the Undo Manager can do its work.

If you use a value of $0000 for both MenuIDs, then that Menu item will be ignored at `UMLogIn`. This allows you to use custom menus, that only have those calls that you wish to make available.

As you will see from the `Undo.Tester` application, the Undo Manager can support not only the usual five Edit Menu items, but Select All and Redo as well. Undo and Redo work together, so Redo is only available after an Undo call has been made.

$0184 UMBootInit

Initialises the Undo Manager; called only by the Tool Locator.

Warning
An Application must never make this call.

Parameters The stack is not affected by this call. There are no input or output parameters.

Errors None

C Call must not be made by an application.

$0284 UMStartUp

Starts up the Undo Manager for use by an application.

Important
Your Application must make this call before it makes any other Undo Manager calls.

Parameters The stack is not affected by this call. There are no input or output parameters.

Errors None

C

```
extern pascal void UMStartUp ();
```

Note: This call must be made before any other Undo Manager calls are made, and must be balanced with a call to UMShutDown at application Quit.

$0384 UMShutDown

Shuts down the Undo Manager.

Important

If your Application has started up the Undo Manager, the application must make this call before it quits.

Parameters The stack is not affected by this call. There are no input or output parameters.

Errors None

C

```
extern pascal void UMShutDown ();
```

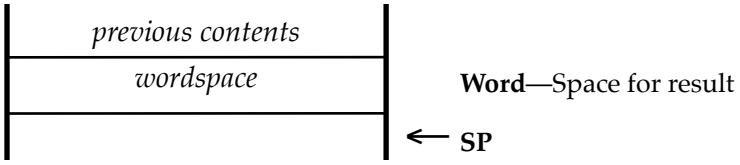
Note: This call must be made at application Quit, and balances the UMStartup call.

$0484 UMVersion

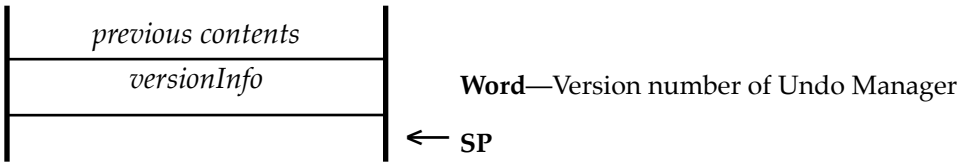
Returns the version number of the Undo Manager.

Parameters

Stack before call



Stack after call



Errors None

C

```
extern pascal Word UMVersion ();
```

$0584 UMReset

Resets the Undo Manager; called only when the system is reset.

Warning
An Application must never make this call.

Parameters The stack is not affected by this call. There are no input or output parameters.

Errors None

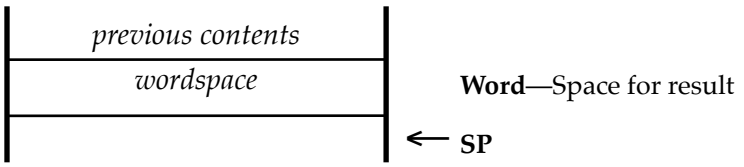
C Call must not be made by an application.

$0684 UMStatus

Indicates whether the Undo Manager is active.
UMStatus returns TRUE if UMStartup has been called and UMShutDown has not been called. The routine returns FALSE if UMStartUp has not been called at all or if UMShutDown has been called since the last time UMStartUp was called.

Parameters

Stack before call



Stack after call

Errors None

C

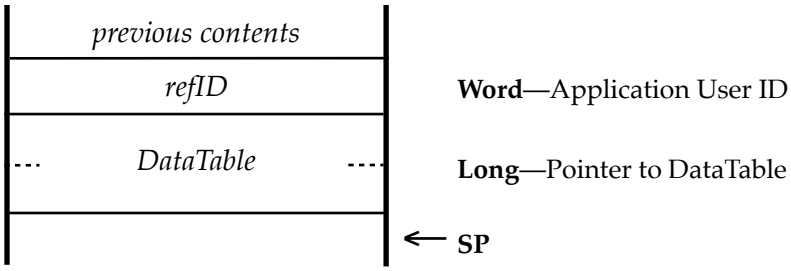
```
extern pascal Boolean UMStatus ();
```

$0984 UMLogIn

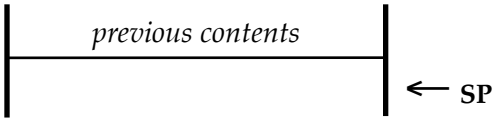
Logs into the Manager either by an Application, NDA, or doModalWindow loop.

Parameters

Stack before call



Stack after call



Errors

$8402	umAlreadyLoggedIn	Window closed
$840B	umToolsNotStarted	Required Tools not started (See Page 5)

C

```
extern pascal void UMLogIn (refID,DataTable);
Word      RefID;
Pointer   DataTable;
```

Note: Every application, must make this call before any Undo Manager action calls are made, and must be balanced with a call to UMLogOut when the window is closed or the application Quits.The call uses the information from the 'DataTable' Pointer, to retrieve a Pointer to the applications 'EventRecord' table, and to the original and replacement menu IDs for the six Edit Menu items. Please refer to 'Coding Examples' for more information.

The application can either make this call this once at application Start, balanced with an UMLogOut at Quit, or before each doModalWindow loop, with a balanced UMLogOut when that window closes.

The refID will normally be the application User ID, but the application may make this call more than once, as long as it uses a unique refID each time. This allows you to use unique Edit menu IDs for each doModalWindow if you wish.

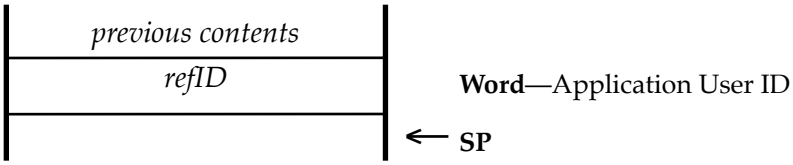
Only 8 applications or windows may call UMLogIn at any one time, with up to 32 controls. You should not use more than around 7 UMLogIn calls from within your application, and no more than 30 controls at once, to allow for any NDAs that may also wish to use the Undo Manager. Using UMLogIn around each doModalWindow, will allow for more controls to be handled overall within your application.

$0A84 UMLogOut

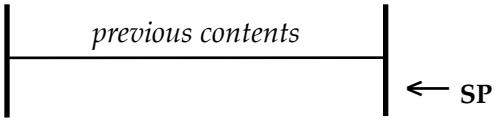
Logs out of the Manager either by an Application, NDA, or doModalWindow loop.

Parameters

Stack before call



Stack after call



Errors

$8403	umNotLoggedIn	Not logged in

C

```
extern pascal void UMLogOut (refID);
Word      RefID
```

Note: Every application, must make this call either when the application Quits, or when a doModalWindow is closed. It must always be balanced with an UMLogIn call.

Any saved clips for this refID will be cleared when this call is made, and the seven Edit Menu items will be disabled.

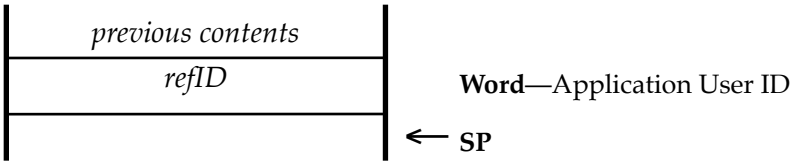
If you need to keep Clips across multiple windows being opened and closed while other windows are still open, then only call UMLogIn at the Start of the application, and call UMLogOut at application Quit.

$0B84 UMKeyAction

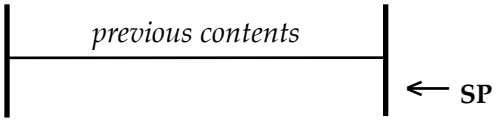
Traps keypresses from the doModalWindow EventHook.

Parameters

Stack before call



Stack after call



Errors

$8403	umNotLoggedIn	Not logged in
$8404	umNoActiveControl	No active LineEdit or TextEdit control found
$8406	umClipBlockFull	Too many clips already saved
$8407	umLowMemory	Not enough memory to save Clip
$8409	umNotEnoughDiskSpace	Not enough disk space to store Clip

GS/OS Errors returned unchanged

C

```
extern pascal void UMKeyAction (refID);
Word     RefID
```

Note: This call must be made from within the EventHook for a doModalWindow. All keypresses must be passed through to the Manager, but the Undo Manager will only trap the Delete key, and OA-Delete key presses.

The call filters and only acts on those two keypresses. Usually this will be the only call made within the EventHook code for a specific doModalWindow, but it is permissible for you to do your own filtering if you wish. Just make sure that as a minimum, you allow Mouse down, the ASCII keys, the Delete key, and the OA-Delete key actions to be passed through to the UMKeyAction call.

A Clip for the currently active control will be saved before the normal Scrap and Clipboard action are made for the Delete keys. The Undo menu will then be enabled.

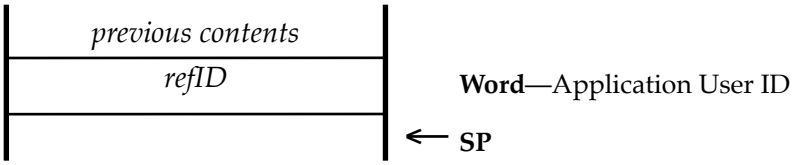
To allow the Manager to work correctly, you must use values of either $003F, $002F, $003D, or $002D for the '*flags*' value in the doModalWindow call.

$0C84 UMMenuAction

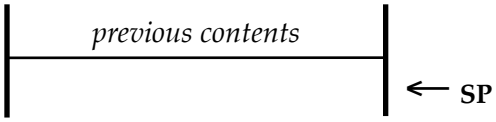
Traps Edit Menu actions from the doModalWindow loop.

Parameters

Stack before call



Stack after call



Errors

$8403	umNotLoggedIn	Not logged in
$8404	umNoActiveControl	No active LineEdit or TextEdit control found
$8405	umNoSavedClip	No Undo Clip was found
$8406	umClipBlockFull	Too many clips already saved
$8407	umLowMemory	Not enough memory to save Clip
$8409	umNotEnoughDiskSpace	Not enough disk space to store Clip

GS/OS Errors returned unchanged

C

```
extern pascal void UMMenuAction (refID);
Word     RefID
```

Note: This call is made from within the main doModalWindow loop. It will only handle the seven Edit Menu keypresses, of Undo, Redo, Cut, Copy, Paste, Clear and Select All.

The routine filters and only acts on those seven keypresses. Usually you will append this call to the end of your main doModalWindow loop, so you can first trap any other Menu items, or clicks on buttons or other controls, before this call is made.

It is permissible to first apply filtering yourself, as long as you allow those seven menu items through to the UMMenuAction call. Remember that the seven Edit Menu item IDs will have been changed from their default values to those supplied in the DataTable pointed to at UMLogin.

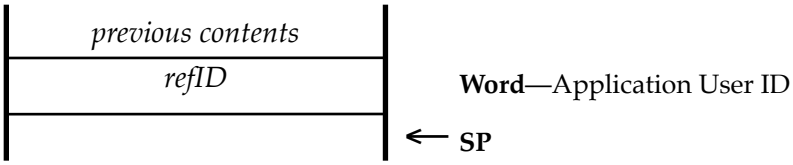
Refer to the Coding Examples for an example of how this call is used.

$0D84 UMSaveClip

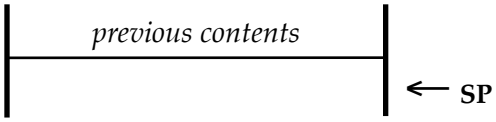
Optionally saves a Clip for the currently active control.

Parameters

Stack before call



Stack after call



Errors

$8403	umNotLoggedIn	Not logged in
$8404	umNoActiveControl	No active LineEdit or TextEdit control found
$8406	umClipBlockFull	Too many clips already saved
$8407	umLowMemory	Not enough memory to save Clip
$8409	umNotEnoughDiskSpace	Not enough disk space to store Clip

GS/OS Errors returned unchanged

C

```
extern pascal void UMSaveClip (refID);
Word     RefID
```

Note: This call can be made at any time after UMLogIn and before UMLogOut. It will save a Clip to the Undo stack for the currently active control of the contents of that control without clearing or otherwise changing the text.

Remember that each control can only have 32 levels of Clip saved before the first Clip will be rolled off from the bottom of the stack to make way for the new Clip.

You might for instance call UMSaveClip before any font changes you make in a TextEdit control. The Undo.Tester actually uses this call in that way, so you can backtrack on any font changes that you have made.

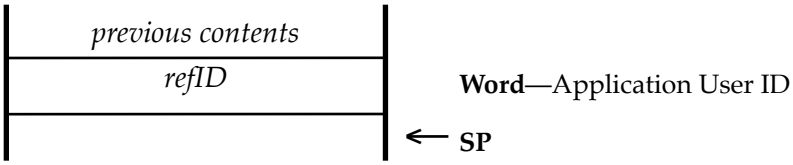
Refer to the Coding Examples for an example of how this call is used.

$0E84 UMClearClips

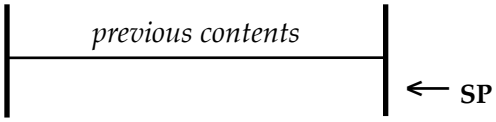
Optionally clears any Clips for the currently active control.

Parameters

Stack before call



Stack after call



Errors

$8403 umNotLoggedIn Not logged in
GS/OS Errors returned unchanged

C

```
extern pascal void UMClearClips (refID);
Word      RefID
```

Note: This call clears all the saved Clips for the currently active control.

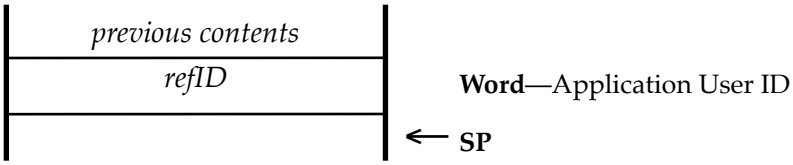
Refer to the Coding Examples for an example of how this call is used.

$0F84 UMClear

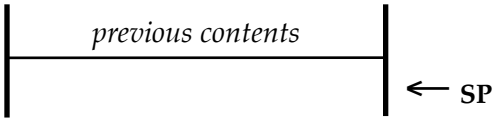
Optionally clears all the saved clips for the User ID.

Parameters

Stack before call



Stack after call



Errors

$8403 umNotLoggedIn Not logged in
GS/OS Errors returned unchanged

C

```
extern pascal void UMClear (refID);
Word      RefID
```

Note: This call clears any saved Clips for all the controls associated with the refID.

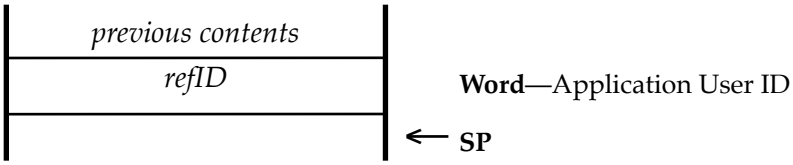
Refer to the Coding Examples for an example of how this call is used.

$1084 UMClearWindowClips

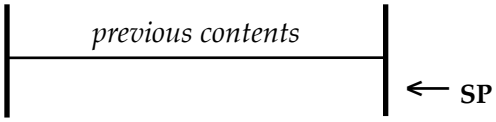
Optionally clears all the Clips for controls in the front Window.

Parameters

Stack before call



Stack after call



Errors

$8408	umNoWindowFound	No front window or controls found

GS/OS Errors returned unchanged

C

```
extern pascal void UMClearWindow (refID);
Word      RefID
```

Note: This call clears all the saved Clips for every LineEdit, TextEdit, and Picture control within the currently active front window, and then Disables the seven Edit Menu Items.

Usually it will be called before you close the front active window.

Note: If you use UMLogIn and UMLogOut at application Start and Quit, you need to call UMClearWindowClips before a controlled window is closed. This will make sure that any pending Clips for controls in that window that will no longer be available after that window is closed will be cleared from memory.

It does not clear saved Clips for any other windows or data Objects related to that refID.

Refer to the Coding Examples for an example of how this call is used.

$1184 UMLogOutAll

Optionally Logs out of all User IDs, and clears all their Clips.

Parameters

The stack is not affected by this call. There are no input or output parameters.

Errors

GS/OS Errors returned unchanged

C

```
extern pascal void UMLogOutAll ();
```

Note: This call clears all the saved Clips for every refID used to call UMLogIn, and then calls UMLogOut for all those refIDs

Note: Other applications or NDAs may well have also called UMLogIn, so this call will have logged them out as well. **Use with care.**

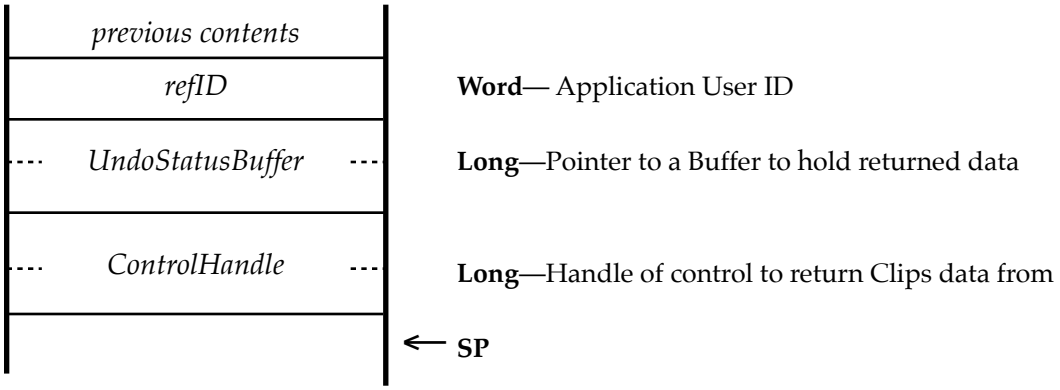
Refer to the Coding Examples for an example of how this call is used.

$1284 UMClipStatus

Returns information about the Clips stack for a control.

Parameters

Stack before call

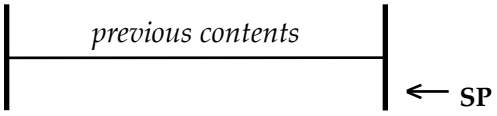


Word— Application User ID

Long—Pointer to a Buffer to hold returned data

Long—Handle of control to return Clips data from

Stack after call



Errors

$8404	umNoActiveControl	No active LineEdit or TextEdit control found

C

```
extern pascal void UMClipStatus (refID, UndoStatusBuffer,
                                                 ControlHandle);

Word     RefID;
Pointer  UndoStatusBuffer;
Long     ControlHandle;

UndoStatusBuffer
NumberLoggedInApps          word  Total 0-16
NumberControlsThisApp       word  Total 0-64
NumberClipsThisControl      word  Stack size 0-10
StackPointerThisControl     word  Undo Stack Pointer 0-10
LastKeypressThisControl     word  $17F, $7F, $100, OA-Z, OA-Y,
                                  OA-X, OA-C, OA-V, or OA-A
```

Allows an application to find out how many applications are currently logged in, how many Clips have been saved for the control or ObjectType, as well as the current state of the Clips stack pointer for the control.

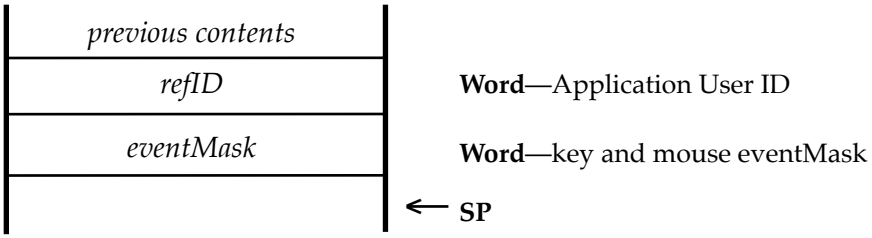
Refer to the Alternate Strategy section of the Coding Examples for further details on this call.

$1384 UMKeyAction2

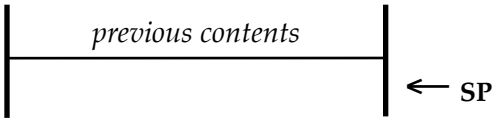
Traps key and menu item presses from the program loop.

Parameters

Stack before call



Stack after call



Errors

$8403	umNotLoggedIn	Not logged in
$8404	umNoActiveControl	No active LineEdit or TextEdit control found
$8406	umClipBlockFull	Too many clips already saved
$8407	umLowMemory	Not enough memory to save Clip
$8409	umNotEnoughDiskSpace	Not enough disk space to store Clip

GS/OS Errors returned unchanged

C

```
extern pascal void UMKeyAction2 (refID,eventMask);

Word      RefID;
Word      eventMask;

EventMask:
    Bit 2 & Bit 1 = Mouse events
    Bit 5 & Bit 3 = Key events
    Bit 14 = Delete key
    Bit 15 = OA-Delete key
```

Note: This is an alternative call to UMKeyAction. The events are intercepted without the need for a menu to be active, so for this call to work correctly, the Edit menu items must be OA-Z, OA-Y, OA-X, OA-C, OA-V and OA-A. As 'Clear' does not normally have a key equivalent, it is not supported.

Refer to the Coding Examples for further details on this call.

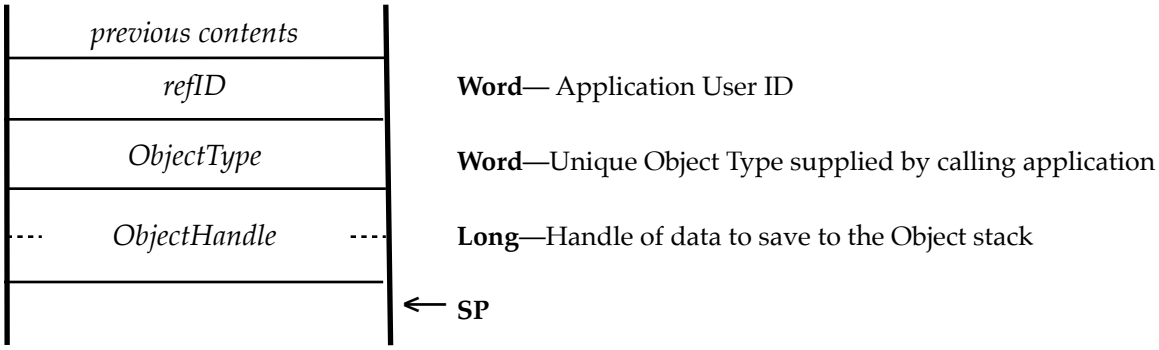
Note: This call was introduced in Undo Manager v1.0.1, so if you rely on its presence, check for this minimum version of the tool at your application startup.

$1484 UMPushUndoObject

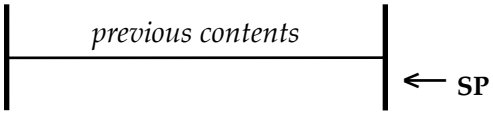
Pushes a supplied Object Handle to the Undo Object stack.

Parameters

Stack before call



Stack after call



Errors

$8403	umNotLoggedIn	Not logged in
$8406	umClipBlockFull	Too many clips already saved
$8407	umLowMemory	Not enough memory to save Clip
$8409	umNotEnoughDiskSpace	Not enough disk space to store Clip
$840A	umObjectHandleInvalid	objectHandle is invalid

GS/OS Errors returned unchanged

C

```
extern pascal void UMPushUndoObject (refID, objectType,
                                                objectHandle);
Word      RefID; objectType;
Long      objectHandle;
```

This call allows an application to temporarily save data from a supplied Handle to a private Object clips stack. This is balanced by the UMPopUndoObject call.

Note: The only other calls that the private Object clips stack is affected by are UMClear, UMClipStatus, and UMLogout.

Refer to the section on "Undo Object Data Calls" for further details on this call.

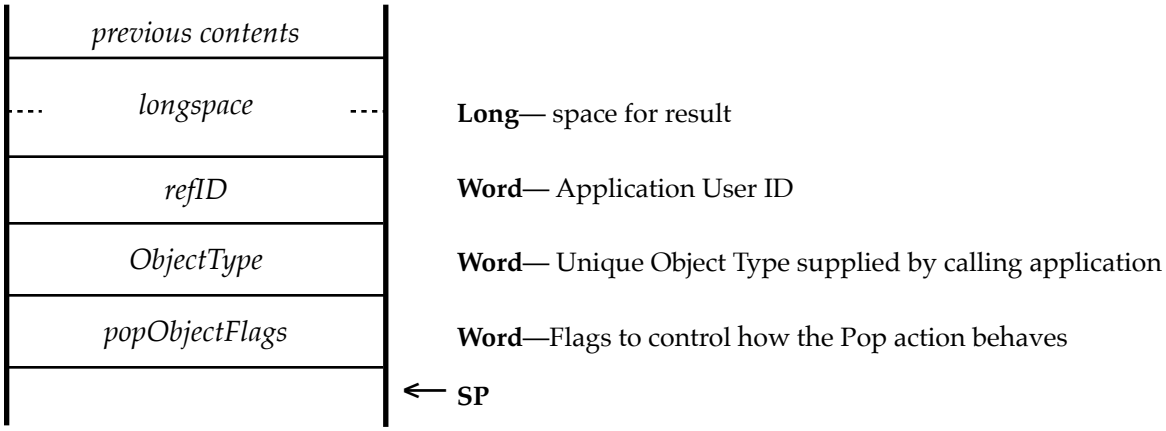
Note: This call was introduced in Undo Manage v1.0.2, so if you rely on its presence, it is advisable to check for a minimum of this version at your application startup.

$1584 UMPopUndoObject

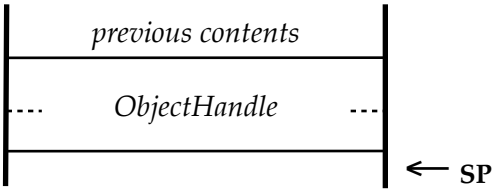
Pops an Object Handle from the Undo stack

Parameters

Stack before call



Stack after call



Errors

$8403	umNotLoggedIn	Not logged in
$8405	umNoSavedClip	No Undo Clip was found

GS/OS Errors returned unchanged

C

```
extern pascal long UMPopUndoObject (refID, ObjectType,
                                                popObjectFlags);
Word      RefID; ObjectType; popObjectFlags
Long      ObjectHandle;
```

popObjectFlags

```
Bit 15  0 = Retrieves entry from top of stack
Bit 15  1 = Retrieves entry indicated by Bits 0-6 from stack
Bit 14  0 = Clears entry and all clips from this entry and above
Bit 14  1 = Don't clear any clips
Bits 0-6 = (Value = 1-32) Recovers target clip (Bit 15 must be set)
```

This call allows an application to pop or recover, saved data from the private Object clips stack to a Handle. This must be balanced by the UMPushUndoObject call.

Refer to the section on "Undo Object Data Calls" for further details on this call.

Note: This call was introduced in Undo Manage v1.0.2, so if you rely on its presence, it is advisable to check for a minimum of this version at your application startup.

Coding Examples

These are ORCA/M examples of how to use the Undo Manager calls. Refer to the ORCA/M, C and Pascal source files for Undo.Tester as an example of how to use the calls in practice:

```
*** Main code

* At application startup:

         pea   $0084            ; Load the Undo Manager
         pea   $0100
         _LoadOneTool
         _UMStartUp

* Either at application startup, or bracketed around doModalWindows:

         PushWord refID         ; User ID or custom ID
         PushLong #DataTable    ; Pointer to required DataTable (See below)
         _UMLogIn               ; Logs in to the Undo Manager

* Open a Window, setup any LineEdit or TextEdit controls, then:

doModalLoop anop

         pha
         pha
         PushLong #EventRecord ; The EventRecord pointed to in the DataTable
         pea   0
         pea   0
         lda   #^EventHook      ; The EventHook (See below)
         ora   #$8000
         pha
         lda   #EventHook
         pha
         pea   0                ; Beep procedure
         pea   0
         pea   $003F            ; or $002F, $003D, or $002D (flags)
         _doModalWindow
         pla
         plx

* In the doModalLoop, your code can first check for any control buttons etc.,
then sends the seven OA menu key presses to the Undo Manager for processing:

         pea   0                ; Clear the menu Hilite
         lda   TaskData+2       ; From your EventRecord
         pha
         _HiliteMenu

         PushWord refID         ; Handles the six Menu Items
         _UMMenuAction

         bra   doModalLoop      ; Return to the start of the main loop
```

```
* Called when a doModalWindow is closed or at application Quit:

         PushWord refID
         _UMLogOut             ; Log out from the Undo Manager

* At Quit, unload the Tool from memory:

         _UMShutDown
         pea   $0084           ; Unload the Undo Manager

         _UnloadOneTool
```

Note: The UMLogIn and UMLogOut calls must be balanced, but may be called up to 16 times within one application, as long as a unique refID is used each time.

```
*** The EventHook routine example:

EventHook anop                 ; Sends all keypresses to the Undo Manager

OrigD    equ  1
OrigB    equ  OrigD+2
RTLAdr   equ  OrigB+1
Ptr      equ  RTLAdr+3

         phd
         phb
         phk
         plb

         PushWord refID        ; processes keypresses, and traps the Delete Keys
         _UMKeyAction

         plb
         pld

         lda  1,s
         sta  5,s
         lda  2,s
         sta  6,s
         tsc
         clc
         adc  #4
         tcs
         rtl
```

Note: This call is referenced from the doModalWindow call, and processes mouse down and key events.

Note: This call can be replaced with UMKeyAction2, when UMMenuAction will no longer be required. Refer to the entry for UMKeyAction2 for more details.

*** The Passed Pointers and DataTables

* Used to LogIn to the Manager, and then for most of the subsequent calls:

```
        refID                ; Either the UserID or a unique ID
```

* At UMLogIn, a Pointer is passed to a DataTable holding a pointer to the EventRecord in your code, this is followed by a Table of Unique menu IDs, and original menu IDs, that will replace the five standard Edit Menu IDs. At UMLogOut, the original or old menu IDs will be restored. The actual menu IDs will vary depending on your application's requirements:

```
DataTable entry
        dc   i4'EventRecord'  long     Pointer to your EventRecord
        dc   i'$01FA,$00FA'   word     New/Old Undo Menu Item ID = OA-Z
        dc   i'$01FB,$00FB'   word     New/Old Cut Menu Item ID = OA-X
        dc   i'$01FC,$00FC'   word     New/Old Copy Menu Item ID = OA-C
        dc   i'$01FD,$00FD'   word     New/Old Paste Menu Item ID = OA-V
        dc   i'$01FE,$00FE'   word     New/Old Clear Menu Item ID
        dc   i'$0100,$0100'   word     New/Old Select All Menu Item ID = OA-A
        dc   i'$0101,$0101'   word     New/Old Redo Menu Item ID = OA-Y
```

* The Undo Manager changes the seven Edit Menu IDs, so it can trap the keypresses before the System has a chance to act on them itself. You simply list IDs that are not otherwise being used in your application as the 'New' IDs, and then following that value with the existing ID, which as shown here, will usually be the default ID for that Menu Item.

  Both IDs may be the same, but the New IDs must be unique, and not within the standard $FA to $FE range.

  Use a value of $0000 for both IDs, if you wish the Undo Manager to ignore that Menu Item.

* A Pointer to an extended EventRecord is passed at the head of the DataTable:

```
        EventRecord    anop
        EventWhat      ds 2
        EventMessage   ds 4
        EventWhen      ds 4
        EventWhere     ds 4
        EventModifiers ds 2
        TaskData       ds 4
        TaskMask       dc i4'$001fffef'
        LastClickTick  ds 4
        ClickCount     ds 2
        TaskData2      ds 4
        TaskData3      ds 4
        TaskData4      ds 4
        LastClickPoint ds 4
```

* When controlling Picture controls, bit 14 fCtlCanBeTarget, of moreFlags for the Picture control must be set. This allows the control to be made a target, and so can be seen by the Undo Manager calls.

As Picture controls do not normally respond to keyboard events to activate the control, you may need to use further code in the doModalWindow EventHook to activate the control on a mouse click before the Undo Manager call:

```
EventHook anop                  ; Sends all keypresses to the Undo Manager

OrigD    equ  1
OrigB    equ  OrigD+2
RTLAdr   equ  OrigB+1
Ptr      equ  RTLAdr+3

         phd
         phb
         phk
         plb

         lda  EventWhat
         cmp  #1                ; Mouse down event
         bne  call_um

         PushLong #MousePos     ; Temporary variable
         _GetMouse

         pha
         PushLong #Temp_Handle ; Temporary variable
         PushLong MousePos
         PushLong WindowHandle
         _FindCursorCtl
         pla

         pha
         pha
         PushLong Temp_Handle
         _GetCtlID
         pla
         plx

         cmp  #PictureID        ; Target Picture control ID
         bne  call_um

         PushLong Temp_Handle
         _MakeThisCtlTarget

call_um  anop

         PushWord MyID
         _UMKeyAction
```

Finish the routine as shown in the earlier EventHook example.

*** Optional calls that can also be made:

```
        pha
        _UMVersion
        pla                 ; Returns the Version number of the Undo Manager

        pha
        _UMStatus
        pla                 ; Boolean; True if Undo Manager is active,
                              False if inactive
```

* Optional call for your custom routines that might be about to change text:

```
        PushWord refID
        _UMSaveClip         ; Saves a Clip for the currently active tool
```

* Optional call for your custom routines to clear any pending Undos for the currently active control:

```
        PushWord refID
        _UMClearClips       ; Clears Clips for the currently active control
```

* Optional call to Clear all the Saved Clips for the passed app refID:

```
        PushWord refID
        _UMClear            ; Clear all Clips for all controls for this refID
```

* Optional call to Clear all saved clips for the active front window:

```
        PushWord refID
        _UMClearWindowClips ; Clears all Clips for the front active window
```

* Optional call to LogOut and Clear all the Clips for all LoggedIn refIDs:

```
        _UMLogOutAll        ; LogOut and Clears Clips for all the refIDs
```

* Optional call to return information about the Clips stack for a control:

```
        PushWord refID
        PushLong UndoStatusBuffer
        PushLong ControlHandle
        _UMClipStatus       ; Returns Clips stack data for a control
```

The call is supplied with the refID or UserID of the calling application, a Pointer to a UndoStatusBuffer, and the ControlHandle of the target control:

UndoStatusBuffer

NumberLoggedInApps	word	Total 0-8
NumberControlsThisApp	word	Total 0-32
NumberClipsThisControl	word	Stack size 0-32
StackPointerThisControl	word	Undo Stack Pointer 0-32
LastKeypressThisControl	word	$17F, $7F, $100, $5A, $59, $58, $43, $56, $41

The UndoStatusBuffer returns the number of applications currently logged in to the Tool, the number of controls that have saved Clips for the supplied refID, the number of saved Clips for the supplied ControlHandle or ObjectType, the current Undo stack pointer into that stack, and the last keypress that the Undo Manager saw for that control.

Note that the Last Keypress value returned will be one of these values:

```
$017F   =    Clear Menu selected, or OA-Delete pressed
$007F   =    Delete pressed
$0100   =    An ASCII key pressed
$0200   =    Arrow key pressed
$005A   =    Undo menu selected, or OA-Z pressed
$0059   =    Redo menu selected, or OA-Y pressed
$0058   =    Cut menu selected, or OA-X pressed
$0043   =    Copy menu selected, or OA-C pressed
$0056   =    Paste menu selected, or OA-V pressed
$0041   =    Select All menu selected, or OAS-A pressed
```

Alternate strategy

An alternative strategy to the preferred method of using TMKeyAction and UMMenuAction with a doModalWindow loop, and a strategy that can be used with applications only using TaskMaster, is to use UMKeyAction2 instead. UMKeyAction2 can also be used where the Edit Menu is not active or unavailable.

If you use UMKeyAction2 with a doModalWindow call, place the call into the EventHook. With TaskMaster, you will need to use the call in your Key-down code, and may also need to use UMMenuAction within your Menu calls. You will probably need to experiment to get the desired result.

UMKeyAction2 is an alternative call to UMKeyAction, and in addition to the the keypresses detailed in UMKeyAction, it traps the Open Apple equivalent keys of Undo, Redo, Cut, Copy, Paste, and Select All as well. The keypresses are intercepted without the need for a menu to be active, so for this call to work correctly, the Edit menu items must be OA-Z, OA-Y, OA-X, OA-C, OA-V and OA-A. As 'Clear' does not normally have a key equivalent, it is not supported.

```
PushWord refID
PushWord eventMask
_UMKeyAction2
```

Which events are acted on can be controlled by the passed eventMask:

```
Bit 1 = Mouse-down events
Bit 2 = Mouse-up events
Bit 3 = Key-down events
Bit 5 = Auto-key events
Bit 14 = Delete key
Bit 15 = OA-Delete key
```

Note: If you are using the call with TaskMaster, the Delete and OA-Delete keys will already have been intercepted, so the character or selection to the left of the cursor will already have been deleted before TaskMaster exits and reaches this call. This means UMKeyAction2 is unable to capture to a Clip the text as it was before the keypress. This will result in subsequent Undo calls returning an incorrect result. Where possible, use a doModalWindow loop to avoid this happening.

Undo Object Data Calls

The data saved from the UMPushUndoObject call, and recovered with PopUndoObject, is not controlled by the Edit Menu commands, and apart from UMClear, UMLogout, and UMClipStatus, the saved stacks are not controlled by the other Undo Manager calls. The Object data clips stack is though counted as one of the thirty-two stacks that can be saved by the Undo Manager, and like the other stacks, can have up to 32 levels in the clips stack.

An application after calling UMLogIn, can use these two calls as a simple way of storing changing versions of data, and then be able to retrieve that data as required. Data is passed to the UMPushUndoObject call in a Handle, and is then later returned to the application as a Handle using the UMPopUndoObject call. By default, the passed data is stored on the Object stack in the same way as with the other calls, and after 32 Handles have been placed on the stack, the next call to UMPushUndoObject will roll off the first item to have been saved. The passed Handle is either saved as a Handle using a local ID, or if it is greater than 4K in size, to a file on disk as is the case with the other calls. When the data is later "Popped", it is returned in a Handle, using the ID passed in the "`refID`" field of the UMPopUndoObject call.

UMClear, as it clears all clips for a passed Application ID, will also clear any saved Object data clips. Similarly, UMLogout, will clear any saved Object data clips for that Application ID. UMClipStatus can be called to return the number of clips saved for a specific "`ObjectType`", but only "`NumberClipsThisControl`" will be valid for the Object stack.

Saving Object Data Handle

Object Data must first be saved to the stack using the UMPushUndoObject call:

```
Pushword refID        ; Application User ID
PushWord ObjectType   ; Unique value for this stack
PushLong ObjectHandle ; Handle holding data to be saved
_UMPushUndoObject
```

ObjectType is a unique value given by the calling application, and is the same value that must be used when calling UMPopUndoObject. This identifies the correct stack to return a saved data Handle from. The value can be of any value, so it is possible to save up to 32 data stacks using different ObjectTypes. As an Object data stack counts as one of the 32 maximum stacks that Undo Manager can control, it is advisable not to save too many stacks, thus leaving space for any other controls you may wish to be handled by the Undo Manager.

ObjectHandle must be a standard Handle, can be of any size, and can hold any form of data that you wish to temporarily save. Once passed, the Handle will be given a new ID, so can no longer be controlled by the calling application.

The UMPushUndoObject call will always place new clips at the top of the stack.

Restoring Object Data Handle

Object Data is returned from the saved stack using the UMPopUndoObject call:

```
pha                   ; space for returned Handle
pha                   ; space for returned Handle
Pushword refID        ; Application User ID
PushWord ObjectType   ; Unique value for this stack
PushWord popObjectFlags ; flags control how the stack is to be handled
_UMPopUndoObject
PullLong ObjectHandle ; Handle holding returned data
```

ObjectType must be the same unique value that was used for UMPushUndoObject when saving a related Handle to the stack.

popObjectFlags control where on the stack the ObjectHandle will be retrieved from, and how the stack will be left after the ObjectHandle has been retrieved (see examples and description below). If Bit 15 of popObjectFlags is set, and a target clip is being recovered, if there are insufficient clips on the stack, the umNoSavedClip Error will be returned.

ObjectHandle is the Handle to the returned data. It will be created with the same refID as is passed by the call. The Handle can be handled again by the calling application, and Deleted as necessary.

popObjectFlags

```
Bit 15    0 = Retrieves entry from top of stack
Bit 15    1 = Retrieves entry indicated by Bits 0-6 from stack
Bit 14    0 = Clears entry and all clips from this entry and above
Bit 14    1 = Don't clear any clips
Bits 0-6  (Value = 1-32) Recovers target clip (Bit 15 must be set)
```

Sample calls:

```
pha
pha
Pushword refID
PushWord ObjectType
pea  $0000            ; popObjectFlags
_UMPopUndoObject
PullLong ObjectHandle
```

With a value of $0000 for popObjectFlags, the call will return the Object Handle from the top of the clips stack, clear the entry from the top of the stack, and reduce the number of saved clips by one.

```
pha
pha
Pushword refID
PushWord ObjectType
pea  $C00A            ; popObjectFlags
_UMPopUndoObject
PullLong ObjectHandle
```

With a value of $800A for popObjectFlags, the call will return the Object Handle from position 10 on the stack. It will then clear that entry from the stack, and

any newer entries above it from the stack, reduce the number of saved clips to 9, and set the stack pointer to 9.

```
pha
pha
Pushword refID
PushWord ObjectType
pea  $4000            ; popObjectFlags
_UMPopUndoObject
PullLong ObjectHandle
```

With a value of $4000 for popObjectFlags, the call will return the Object Handle from the top of the clips stack. It will not clear the entry from the stack, and will leave the stack pointer, and the number of saved clips as it was. This allows for a clip to be retrieved multiple times if necessary.

```
pha
pha
Pushword refID
PushWord ObjectType
pea  $C008            ; popObjectFlags
_UMPopUndoObject
PullLong ObjectHandle
```

With a value of $C008 for popObjectFlags, the call will return the Object Handle from position 8 on the stack. It will not clear the entry from the stack, and will leave the stack pointer, and the number of saved clips as it was. This allows for a clip to be retrieved multiple times if necessary from anywhere on the stack.

Note: If you do not know how many clips are on the stack, call UMClipStatus to return the number of clips in the stack. To identify the correct stack, pass ObjectType+$80000000 instead of the ControlHandle value, and the number of clips will be returned in the NumberClipsThisControl field.

To sum up: UMPushUndoObject always places new Handles on the top of the stack. With the default value of $0000 passed for popObjectFlags, when the ObjectHandle is retrieved using UMPopUndoObject, it will be returned from the top of the stack.

By setting Bit 15 of popObjectFlags, an entry can be retrieved from within the stack. This allows you to randomly recover a data Handle from the stack. Bits 0-6 of popObjectFlags holds the entry to recover. Valid values are 1-32.

The default is for the retrieved entry, and any entries above that entry on the stack to be erased from the stack. By setting Bit 14 of popObjectFlags, the retrieved entry, and any others above it in the stack will not be erased. This allows you to save up to 32 Object data Handles, and randomly and repeatedly retrieve them as you wish. If you wish to clear the stack completely, just call UMPopUndoObject for the required stack, with popObjectFlags set to a value of $8001. The application must delete the returned Handle when it is done with it.

If there are no more entries on the stack, or the entry number passed is higher than the top of the stack, you will get the umNoSavedClip error. To avoid such an error, call UMClipStatus before the UMPopUndoObject call, and see how many clips the stack holds.

Undo Manager Error Codes

umFailedLogin	$8401	16 apps already logged in
umAlreadyLoggedIn	$8402	This refID already logged in
umNotLoggedIn	$8403	This refID not logged in
umNoActiveControl	$8404	No active LineEdit, TextEdit, or Picture control found
umNoSavedClip	$8405	No Undo Clip was found
umClipBlockFull	$8406	64 controls already have Clips saved
umLowMemory	$8407	Not enough memory to save Clip
umNoWindowFound	$8408	No active front window or controls found
umNotEnoughDiskSpace	$8409	Not enough disk space to store Clip
umObjectHandleInvalid	$840A	Object Handle is invalid
umFailedCall	$84FF	Generic failure

Version History:

v1.0.0 First release 29th May 2018
v1.0.1 Updated 20th July 2018
Added UMKeyAction2 call (required for Spectrum 2.5.5)
v1.0.2 Updated September 2018
Added UMPushUndoObject and UMPopUndoObject calls
v1.0.3 Updated August 2019
UMLogIn now checks for required Tools being started

Contacts - Problems

Hopefully you will have none, but if you do, and they cannot be answered by reading these notes, please contact me on:

spectrumdaddy@speccie.uk

Contacts - Other information

Please check out my web site for a number of other utilities, Tools, and programs that I have written:

http://speccie.uk

If you do not already know about Spectrum™, please drop by my web site and read more. Apart from all the other wonderful things it does, Spectrum™ offers many useful tools for processing files, such as post processing text files that you have received that may have obstinate formatting.

Spectrum™ is now Freeware, and with all my many other applications, is available from my web site:

http://speccie.uk

Someone once said to me, 'Spectrum™ does everything!'