Chapter 6:

Display Manager Protocol

The Display Manager (MAN) protocol (protocol ID 1) consists of atoms that are responsible for handling objects displayed on online service forms. Display Manager protocol atoms control the creation and management of objects and basic form display. Multiple hierarchies of display objects are maintained.

Display Manager Protocol Atoms

The Display Manager (MAN) protocol functions and their associated atoms follow:

Table 6-1: Display Manager Protocol Functions and Atoms

Function	Atoms	Page
Object definition, creation, start or end	atom$man_end_object	6-29
	atom$man_insert_object_after	6-44
	atom$man_insert_object_before	6-45
	atom$man_preset_gid	6-63
	atom$man_preset_relative	6-64
	atom$man_start_alpha	6-83
	atom$man_start_first	6-86
	atom$man_start_last	6-89
	atom$man_start_object	6-92
	atom$man_start_sibling	6-94
Object context setting or ending	atom$man_change_context_relative	6-10
	atom$man_check_and_set_context_rid	6-11
	atom$man_end_context	6-26
	atom$man_set_context_globalid	6-70
	atom$man_set_context_index	6-72
	atom$man_set_context_relative	6-74

Table 6-1: Display Manager Protocol Functions and Atoms

Function	Atoms	Page
	atom$man_set_context_response_id	6-75
Object tree navigation	atom$man_get_child_count	6-33
	atom$man_get_first_window	6-36
	atom$man_get_next_window	6-39
Objects or windows closing	atom$man_close	6-15
	atom$man_close_children	6-16
Window minimization	atom$man_is_window_iconic	6-49
Focus on objects setting or detecting	atom$man_do_magic_response_id	6-21
	atom$man_do_magic_token_arg	6-22
	atom$man_get_top_window	6-42
	atom$man_is_rendered	6-46
	atom$man_make_focus	6-57
	atom$man_place_cursor	6-60
Response ID objects management	atom$man_do_magic_response_id	6-21
	atom$man_get_first_response_id	6-35
	atom$man_get_next_response_id	6-38
	atom$man_get_request_window	6-40
	atom$man_get_response_window	6-41
	atom$man_ignore_response	6-43
	atom$man_is_response_ignored	6-47
	atom$man_is_response_pending	6-48
	atom$man_response_pop	6-68
	atom$man_set_context_response_id	6-75
	atom$man_set_response_id	6-80
Object data content — clearing or appending	atom$man_append_data	6-4
	atom$man_clear_file_name	6-12
	atom$man_clear_object	6-13
	atom$man_clear_relative	6-14
	atom$man_end_data	6-27
	atom$man_replace_data	6-67
Secure IP sessions management	atom$man_append_secure_data	6-5
	atom$man_end_ip_session	6-28
	atom$man_set_append_secure_data	6-69
	atom$man_start_ip_session	6-88
Object display update	atom$man_close_update	6-17
	atom$man_update_display	6-97
	atom$man_update_end_object	6-98
	atom$man_update_fonts	6-99

Table 6-1: Display Manager Protocol Functions and Atoms

Function	Atoms	Page
	atom$man_update_woff_end_stream	6-100
	atom$man_post_update_gid	6-61
	atom$man_force_update	6-31
Pop-up menu items—appending	atom$man_build_favorites_menu	6-6
	atom$man_build_savemail_menu	6-8
Object edit management	atom$man_do_edit_menu	6-19
	atom$man_set_edit_position	6-77
	atom$man_set_edit_position_to_end	6-78
Object titles management	atom$man_get_index_by_title	6-37
	atom$man_preset_title	6-65
	atom$man_set_default_title	6-76
	atom$man_use_default_title	6-101
Object attributes of the Attribute Manager (MAT) atom protocol management	atom$man_get_attribute	6-32
Form management	atom$man_display_popup_menu	6-18
	atom$man_spell_check	6-82
	atom$man_preset_authoring_form	6-62
	atom$man_build_font_list	6-7
DOD object rendering	atom$man_force_old_style_dod	6-30
Timer management	atom$man_enable_continuous_timer	6-24
	atom$man_enable_one_shot_timer	6-25
	atom$man_kill_timer	6-54
Session and logging management	atom$man_item_get	6-50
	atom$man_item_set	6-52
	atom$man_logging_command	6-55
	atom$man_set_item_type	6-79
Domain management	atom$man_set_edit_position	6-77
URL preset management	atom$man_preset_url	6-66

The MAN protocol atoms are described in alphabetical order in the rest of this chapter.

atom$man_append_data

20 ($14)

atom$man_append_data appends data to or inserts data in the current object. Both text and binary data, which are used to display downloading graphic images, are supported.

If the given object has already been rendered, a visual update for the current object is implied after each **atom$man_append_data**. Otherwise, the appended data is buffered and displayed immediately upon visual creation of the object (the first explicit atom$man_update_display on page 6-97 for the independent ancestor of the object).

Syntax

```
atom$man_append_data <data>
```

`<data>`	Specifies variable-length data to be added to the object's content.

Return Value

None.

Example

The following example adds text [Enter word(s):] to the Keyword form:

```
atom$man_start_sibling <org_group, “”>
atom$man_start_object <ornament, “”>
atom$mat_size <10, 1>
atom$mat_vertical_spacing <5>
atom$mat_font_id <arial>
atom$mat_font_size <10>
atom$mat_font_style <bold>
atom$man_append_data <"Enter words(s):">
atom$man_start_sibling <edit_view, “”>
```


atom$man_append_secure_data
102 ($66)

atom$man_append_secure_data decrypts incoming data from the host and displays the secure data in the current field. Typically, this atom is used for transmitting secure IP data.

Syntax

```
atom$man_append_secure_data <data>
```

`<data>`	Specifies the encrypted data to be decrypted and displayed in the current field.

Return Value

None.

Example

The following example decrypts the incoming data (for example, an account or social security number) from the host and displays the secure data in the field:

```
atom$man_start_object <view, “”>
atom$man_append_secure_data <”--~“ÚÚx -Æ#$j7”>
atom$mat_bool_writeable <no>
atom$man_end_object
```

atom$man_build_favorites_menu

106 ($6A)

atom$man_build_favorites_menu dynamically adds Favorite Places menu items to the current object (popup menu). This atom is used in the action stream that is invoked when the popup menu sends the `WM_INITPOPUPMENU` message.

Syntax

```
atom$man_build_favorites_menu
```

Return Value

None.

Example

The following example creates the dynamic menu of your favorite places:

```
atom$man_start_object <dss_list, “Favorite Places”>
atom$mat_relative_tag <15>
atom$mat_bool_inactive_for_guest <yes>
atom$act_replace_select_action
atom$uni_start_stream
atom$man_set_context_relative <15>
atom$man_build_favorites_menu
atom$man_end_context
atom$uni_end_stream
atom$man_end_object
```


atom$man_build_font_list

94 ($5E)

atom$man_build_font_list dynamically adds a font list to a single selectable list dropdown menu on a rich toolbar (or wherever you are putting it).

Syntax

```
atom$man_build_font_list
```

Return Value

None.

Example

The following example dynamically adds a font list to a single selectable list dropdown menu on a rich toolbar (or wherever you are putting it):

```
atom$man_start_object <dynamic_list, "">
atom$mat_orientation <vff>
atom$mat_size <21, 1>
atom$mat_relative_tag <22>
atom$mat_relative_tag <22>
atom$mat_title_pos <left | center>
atom$mat_horizontal_spacing <0>
atom$man_build_font_list <>
atom$mat_sort_order <reverse>
atom$act_replace_select_action
```

atom$man_build_savemail_menu

105 ($69)

atom$man_build_savemail_menu dynamically adds all the mail folder items to the current object such as a Personal Filing Cabinet (PFC) popup menu. This atom is used in the action stream that is invoked when the popup menu sends the `WM_INITPOPUPMENU` message.

Syntax

```
atom$man_build_savemail_menu
```

Return Value

None.

Example

The following example adds a mail folder to the PFC popup menu:

```
atom$man_start_object <dss_list, “Save to Personal Filing
Cabinet”>
atom$mat_relative_tag <14>
atom$mat_bool_inactive_for_guest <yes>
atom$act_replace_select_action
atom$uni_start_stream
atom$man_set_context_relative <14>
atom$man_build_savemail_menu
atom$man_end_context
atom$uni_end_stream
atom$man_start_object <trigger, “Incoming/Saved Mail”>
atom$act_replace_select_action
atom$uni_start_stream
atom$mip_action_command <44>
atom$uni_end_stream
atom$man_end_object
```


atom$man_build_signatures_menu
108 ($6C)

atom$man_build_signatures_menu creates a dynamic popup menu of the member's current signature files to place the signature file directly into the body of an e-mail.

Syntax

```
atom$man_build_signatures_menu
```

Return Value

None.

Example

The following example creates a dynamic popup menu of the member's current signature files to place the signature file directly into the body of an e-mail:

```
atom$man_start_object <dss_list> ""
atom$mat_relative_tag 505
atom$mat_bool_popup_menu <yes>
atom$act_replace_select_action
atom$uni_start_stream
atom$man_set_context_relative <505>
atom$man_build_signatures_menu
atom$man_end_context
atom$uni_end_stream
atom$man_end_object
```

atom$man_change_context_relative
12 ($0C)

atom$man_change_context_relative is a shorthand atom that changes the context from the current object to one of its siblings without the use of atom$man_end_context on page 6-26. Context is a term used to indicate on which object processing occurs. All processing acts on the object in context.

Syntax

```
atom$man_change_context_relative <relative_ID>
```

`<relative_ID>`	Specifies the relative ID of the new context object.

Return Value

The relative ID of the new context object.

Example

The following example changes the context from ID 13 to various objects on the form and specifically enables or disables the selection of these objects:

```
atom$man_set_context_relative <13>
atom$mat_bool_disabled <yes>
atom$man_change_context_relative <16>
atom$mat_bool_disabled <yes>
atom$man_change_context_relative <18>
atom$mat_bool_disabled <yes>
atom$man_change_context_relative <11>
atom$mat_bool_disabled <no>
atom$man_end_context
```


atom$man_check_and_set_context_rid
68 ($44)

atom$man_check_and_set_context_rid checks the existence (typically in a window) of a relative ID specified in the argument and, when found, sets the context to that relative ID. If the specified object is not found, the context is not set.

Syntax

```
atom$man_check_and_set_context_rid <relative_ID>
```

`<relative_ID>` Specifies a relative ID to put into context.

Return Value

The relative ID when found. If the relative ID is not found, the value is 0.

Example

The following example searches for an object with a relative ID of 3 and changes the context to that object when it is found:

```
atom$man_check_and_set_context_rid <3>
atom$if_last_return_true_then <1>
atom$man_append_data <"New text for field 3...">
atom$uni_sync_skip <1>
```

atom$man_clear_file_name

69 ($45)

atom$man_clear_file_name clears the file's instance for the current window. This atom is typically used by client developers.

Syntax

```
atom$man_clear_file_name
```

Return Value

None.

Example

The following example clears the file's instance for the current window:

```
atom$man_clear_file_name
```


atom$man_clear_object

15 ($0F)

atom$man_clear_object clears the content of the current object. Most frequently used with views, this atom clears any text displayed in the view. The object content is not actually cleared until a visual update (explicit through atom$man_update_display on page 6-97 or implicit through the use of atom$man_append_data on page 6-4) takes place for the current object.

Syntax

```
atom$man_clear_object
```

Return Value

None.

Example

The following example clears the content from the current object:

```
atom$man_clear_object
```

atom$man_clear_relative

14 ($0E)

atom$man_clear_relative is a shorthand atom that clears the content of a view with a specific relative ID in the current context.

Syntax

```
atom$man_clear_relative <relative_ID>
```

`<relative_ID>`	Specifies the relative ID of the object whose content is to be cleared.

Return Value

None.

Example

The following example clears the content of field 4 for the current window:

```
atom$man_clear_relative <4>
```


atom$man_close
3 ($03)

atom$man_close marks an object to be deleted. The next time a display update is triggered for the subtree that contains the object, the object and all of its children are deleted.

When an object is deleted, it is removed from the display tree, and its visual representation is removed from the screen. Deleting an object in a window automatically triggers a full update for the window, and the positions of objects in the window are adjusted to fill the space occupied by the departing object.

The most common use of this atom is to close (pop) an entire window.

Syntax

```
atom$man_close [<global_ID>]
```

`[<global_ID>]` Specifies the optional global ID of the object to close. If the ID is omitted, the current object is assumed.

Return Value

None.

Example

The following example closes the current window:

```
atom$uni_start_stream
atom$man_close
atom$uni_end_stream
```

atom$man_close_children

4 ($04)

atom$man_close_children marks children of an object to be deleted. The object itself is not marked. For more information about deleting objects, see atom$man_close on page 6-15.

Syntax

```
atom$man_close_children [<global_ID>]
```

`[<global_ID>]`	Specifies the optional global ID of the parent object. If the ID is omitted, the current object is assumed.

Return Value

None.

Example

The following example marks the children of an object for deletion:

```
atom$man_close_children <0-0-52>
```

The result of this example is that all children belonging to object 0-0-52, which is the online cluster, are marked for deletion. Use this atom when a member signs off of the online service.

atom$man_close_update
28 ($1C)

atom$man_close_update is a shorthand atom that marks the current object to be deleted and automatically performs a display update. For more information about deleting objects, see atom$man_close on page 6-15.

Syntax

```
atom$man_close_update [<global_ID>]
```

`[<global_ID>]`	Specifies the optional global ID of the object to be deleted. If the ID is omitted, the global ID defaults to the current object.

Return Value

None.

Example

The following example marks an object for deletion and then updates the display:

```
atom$uni_start_stream
atom$man_close_update <32-243>
atom$uni_end_stream
```

atom$man_display_popup_menu
99 ($63)

atom$man_display_popup_menu invokes the current object as a popup menu. In order to display the menu, it must have been previously defined (with **mat_bool_popup_menu** <**yes**>).

Syntax

```
atom$man_display_popup_menu
```

Return Value

None.

Example

The following example invokes a defined list with assigned RID as a popup menu:

```
atom$man_start_object <trigger>
atom$act_replace_popup_menu_action
atom$uni_start_stream
atom$man_set_context relative <10>
atom$man_display_popup_menu
atom$man_end_context
atom$uni_end_stream
atom$man_end_object
```


atom$man_do_edit_menu

35 ($23)

atom$man_do_edit_menu executes a specified item in the Edit menu. The Edit menu is selectable from the menu bar, located across the top of the main window of the online service.

Syntax

```
atom$man_do_edit_menu <menu_command>
```

`<menu_command>` Specifies one of the menu items of the Edit menu. Values are as follows:

- `0` `DVL_EDIT_MENU_INIT`—Checks the object to determine what should be highlighted on the Edit menu.
- `1` `DVL_EDIT_MENU_CUT`—Executes the Edit menu's **Cut** command for the current object.
- `2` `DVL_EDIT_MENU_COPY`—Executes the Edit menu's **Copy** command for the current object.
- `3` `DVL_EDIT_MENU_PASTE`—Executes the Edit menu's **Paste** command for the current object.
- `4` `DVL_EDIT_MENU_SELECT_ALL`—Executes the Edit menu's **Select All** command for the current object.
- `5` `DVL_EDIT_MENU_UNDO`—Executes the Edit menu's **Undo** command for the current object.
- `6` `DVL_EDIT_MENU_DELETE`—Executes the Edit menu's **Delete** command for the current object.
- `9` `DVL_EDIT_MENU_FIND`—Executes the Edit menu's **Find in Top Window** command for the current object.

10 DVL_EDIT_MENU_REDO—Executes the Edit menu's **Redo** command for the current object.

11 DVL_EDIT_MENU_SPELL—Executes the Edit menu's **Spell Check** command for the current object.

12 DVL_EDIT_MENU_IMAGE—Executes the Edit menu's **View Pictures** or **Insert Picture** command for the current object.

13 DVL_EDIT_MENU_BACKIMAGE—Executes the Edit menu's **Insert Background Picture** command for the current object.

Return Value

None.

Example

The following example executes the Cut command in the Edit menu:

```
atom$uni_start_stream
atom$man_do_edit_menu <1>
atom$uni_end_stream
```


atom$man_do_magic_response_id
6 ($06)

atom$man_do_magic_response_id performs explicit *magic* to invoke a window. Magic occurs when the inactive window is made active again (the topmost form) without redrawing it from the host so that multiple instances of the same window do not occur. If the window being invoked already exists on the screen, the atom stream that normally creates the window is terminated.

This atom is used in conjunction with atom$man_set_response_id on page 6-80 to associate a trigger that invokes the window.

Syntax

```
atom$man_do_magic_response_ID [<response_ID>]
```

`[<response_ID>]`	Specifies the optional response ID to use. If the argument is omitted, the response ID defaults to the object that triggered the action.

Return Value

If magic was performed, the value is non-zero.

Example

The following example examines the entire screen for the existence of a window on the screen (based on a trigger action) and, if it exists, brings the window to the front:

```
atom$uni_start_stream_wait_on
atom$man_do_magic_response_id <8422>
atom$uni_invoke_no_context <40-5792>
atom$uni_start_stream
atom$man_set_context_global_id <1>
atom$man_set_response_id <8422>
atom$man_start_object <ind_group, “Keyword”>
atom$mat_orientation <vff>
```

atom$man_do_magic_token_arg
5 ($05)

atom$man_do_magic_token_arg performs *magic* based on a token or a token and argument. Magic occurs when an inactive window is made active again (the topmost form) without redrawing it from the host so that multiple instances of the same window do not occur.

This atom checks the magic table for a window that resulted from the token/argument specified. If found, the window is brought to the front, and the current stream is terminated.

For information about how a token/argument becomes associated with a window, see atom$man_set_response_id on page 6-80.

Syntax

```
atom$man_do_magic_token_arg <token [gid]>
```

`<token [gid]>`	Specifies a 2-byte token for routing to a host server (a hex value that is a code of two ASCII characters). If the length of the atom's argument is greater than 2 bytes, the Display Manager protocol assumes that a 4-byte global ID argument is included after the token. For example, a 4-byte hex value 14x 00x 01x 4Fx defines a global ID of 20-0-00355.

Return Value

If magic was performed, the value is non-zero.

Example

The following example examines the screen for the existence of a window based on a host server routing window of (66x 6Cx) and a global ID 20-0-00355 (14x 00x 01x 4Fx) and, if it exists, the window is brought to the front of the screen:

```
atom$man_do_magic_token_arg <66x 6Cx 14x 00x 01x 4Fx>
```

The result of this example is that an action resulting from the argument causes magic to be performed. If a window (identified by a response ID) is

associated with that token argument in the magic table, the inactive window is brought to the front.

atom$man_enable_continuous_timer
76 ($4C)

atom$man_enable_continuous_timer enables a timer that generates an event continuously. The timer has a specified duration. When the duration expires, a specified event takes place. Timer attributes are specified by **atom$mat_timer_duration** and **atom$mat_timer_event**.

For a continuous timer, initial timer attributes need only be set before the timer is first enabled. You can change timer attributes at any time.

atom$man_enable_one_shot_timer on page 6-25 also enables a timer but it cannot run continuously. It must be rearmed with timer attributes each time it is invoked.

Syntax

```
atom$man_enable_continuous_timer
```

Return Value

None.

Example

The following example sets up a continuous timer:

```
atom$mat_timer_duration <1000>
atom$mat_timer_event <200>
atom$man_enable_continuous_timer
```

The result of this example is that a continuous timer's attributes are set up and the timer is enabled twice. First, the timer's duration and event are established. Later, the continuous timer is enabled. After 1000 milliseconds (1 second), event 200 occurs. Later, the continuous timer is again enabled with the same attributes as before.

atom$man_enable_one_shot_timer

75 ($4B)

atom$man_enable_one_shot_timer enables a timer that generates an event once. The timer has a specified duration. When the duration expires, a specified event takes place. Timer attributes are specified by **atom$mat_timer_duration** and **atom$mat_timer_event**.

For a one-shot timer, timer attributes must be set each time before the timer is enabled.

atom$man_enable_continuous_timer on page 6-24 also enables a timer but it can run continuously. It does not have to be rearmed with timer attributes each time it is invoked.

Syntax

```
atom$man_enable_one_shot_timer
```

Return Value

None.

Example

The following example sets up a one-shot timer:

```
atom$mat_timer_duration <1000>
atom$mat_timer_event <200>
atom$man_enable_one_shot_timer
```

The result of this example is that a one-shot timer's attributes are set up and the timer is enabled. First, the timer's duration and event are established, then later the timer is enabled. Once enabled, the timer lasts 1000 milliseconds (1 second) and then event 200 occurs.

atom$man_end_context

29 ($1D)

atom$man_end_context returns context to the object that was in context before the last context-setting atom, such as atom$man_set_context_globalid on page 6-70.

Syntax

```
atom$man_end_context
```

Return Value

None.

Example

The following example ends the context associated with global ID 32-123 and returns the context to the previous object in context:

```
atom$man_set_context_globalid <32-123>
.
.
.
atom$man_end_context
```

The result of this example is that context returns to the object that was in context before global ID 32-123.

atom$man_end_data

73 ($49)

atom$man_end_data indicates the end of a data download of a graphic image that is rendered as it downloads. Such on-the-fly graphic images, for example gif, jpeg, and art files, appear in the window as they are being downloaded.

Syntax

```
atom$man_end_data
```

Return Value

None.

Example

The following example displays an on-the-fly graphic image during its download cycle:

```
atom$man_set_context_relative <1>
atom$man_append_data <Binary data goes here...>
atom$man_append_data <More binary data goes here...>
atom$man_end_data
```

atom$man_end_ip_session

104 ($68)

atom$man_end_ip_session marks the end of a secure IP session, which signals the host to conclude the IP processing that was initiated with atom$man_start_ip_session on page 6-88.

Syntax

```
atom$man_end_ip_session <gid>
```

`<gid>`	Specifies the global ID (4-byte hex value) of the cache record in the database. For example, a value of 14x 00x 01x 4Fx defines a global ID of 20-0-00335, where 14x equals 20, 00x equals 0, and 01x 4Fx equals 00335.

Return Value

None.

Example

The following example ends the IP session using GID 20-0-00335 (14x 00x 01x 4Fx) for the cache database record:

```
atom$man_start_ip_session <59x 23x 14x 00x 01x 4Fx>
atom$man_start_object <edit_view, "">
atom$mat_size <3Cx 0Ex>
atom$mat_secure_field <0>
atom$mat_bool_writeable <yes>
atom$man_end_object
atom$man_update_display
atom$man_end_ip_session <14x 00x 01x 4Fx>
atom$uni_wait_off
atom$uni_end_stream
```


atom$man_end_object
2 ($02)

atom$man_end_object marks the end of the definition of the current object. The context returns back to the object that was in context before atom$man_start_object on page 6-92, which opened the current object.

Syntax

```
atom$man_end_object
```

Return Value

None.

Example

The following example marks the end of the definition of a window that has a global ID of 32-123:

```
atom$man_start_object <32-123>
.
.
.
atom$man_end_object
```

atom$man_force_old_style_dod

92 ($5C)

atom$man_force_old_style_dod specifies that the requested artwork will be rendered using old-style data on demand (DOD) and that the progressive rendering method of DOD will not be used. Old-style DOD displays the blue update gauges and the *Please wait while we add new art...* message. This atom is sent by the client.

Syntax

```
atom$man_force_old_style_dod
```

Return Value

None.

Example

The following example specifies that artwork data will be rendered using an old-style DOD:

```
atom$uni_start_stream
.
.
.
atom$man_start_object <independent>
atom$man_force_old_style_dod
.
.
.
atom$uni_end_stream
```


atom$man_force_update

78 ($4E)

atom$man_force_update causes the client to complete processing of all deferred events (`show_window`'s and `set_redraw`'s) before causing the current (top) form to be updated. (This atom was created specifically to ensure the main menu gets updated before the welcome screen appears.)

Syntax

```
atom$man_force_update
```

Return Value

None.

Example

The following example causes the client to complete the processing of all deferred events before updating the top form:

```
atom$uni_start_stream <00x>
atom$act_get_db_value <14x, 00x, 00x, 42x>
if_last_return_true_then <1>
atom$uni_invoke_no_context <32-5483>
atom$man_force_update
atom$uni_sync_skip <1>
atom$uni_wait_on <>
```

atom$man_get_attribute

64 ($40)

atom$man_get_attribute checks the state of the specified attribute for the current object. It changes a MAT atom's set functionality to a get functionality.

Syntax for VP Designer

```
atom$man_get_attribute <atom>
```

Syntax for form_edit

```
atom$man_get_attribute prot$<protocol> atom$<attribute atom>
```

`<protocol>`	This argument value is not used for **VP Designer**. For **form_edit**, this argument value specifies the MAT protocol name **proto_mat**.
`<attribute atom>`	Specifies a MAT atom that sets an object attribute to a get object attribute function so that the existing object attribute value is now retrieved.

Return Value

The existing 4-byte value of the atom's attribute value.

Examples

The following example examines the value of the current object to see if the object is selectable (Yes or 1) or disabled (No or 0):

```
atom$man_get_attribute <atom$mat_bool_disabled>
```

The following example examines the art animation rate. If the rate is 0, the rate is changed to 150:

```
atom$uni_start_stream
atom$man_set_context_relative <61>
atom$man_get_attribute <atom$mat_art_animation_rate>
atom$if_last_return_false_then <1>
atom$mat_art_animation_rate <150>
atom$mat_bool_repeat_animation <yes>
atom$man_update_display
atom$uni_sync_skip <1>
atom$man_end_context
atom$uni_end_stream
```


atom$man_get_child_count

67 ($43)

atom$man_get_child_count determines the number of children that belong to an object.

Syntax

```
atom$man_get_child_count
```

Return Value

The number of children that belong to the object.

Example

The following example determines the number of children in the current object:

```
atom$man_get_child_count
```

atom$man_get_display_characteristics
107 ($6B)

atom$man_get_display_characteristics gets the width and height of the client area.

Syntax

```
atom$man_get_display_characteristics <x y>
```

`<x y>`

The first byte (*x*) indicates which kind of information is needed, where *x* is:

- 0 indicates client width
- 1 indicates client height
- 2 indicates the query of the desktop horizontal resolution
- 3 indicates the vertical resolution

The second byte (*y*) is the percentage of the information that is required in the first byte.

Return Value

2d byte (percentage of attribute)

Example

The following example gets the client width and sizes the form to 80% of that width:

```
atom$man_get_display_characteristics 0 80
atom$uni_use_last_atom_value prot$mat atom$mat_width
```


atom$man_get_first_response_id

55 ($37)

atom$man_get_first_response_id looks up the first (oldest) response ID listed in the magic table. Note that in addition to response IDs, the magic table contains global IDs, tokens, and active/ignored states of all the open AOL windows on the screen. This atom is commonly used in a loop with atom$man_get_next_response_id on page 6-38.

Syntax

```
atom$man_get_first_response_id
```

Return Value

The first response ID found in the top of the magic table. If no response ID is found, the value is 0.

Example

The following example looks up the oldest response ID in the magic table:

```
atom$man_get_first_response_id
atom$uni_save_result
atom$uni_start_loop
atom$uni_get result
atom$if_last_return_true_then <1>
.
.
.
(atom functions to perform for each window)
.
.
.
atom$man_get_next_response_id
atom$uni_save_result
atom$uni_end_loop
atom$uni_sync_skip <1>
```

atom$man_get_first_window

60 ($3C)

atom$man_get_first_window looks up the first independent object (window) located under the root of the object tree. This atom is commonly used in a loop with atom$man_get_next_window on page 6-39.

Syntax

```
atom$man_get_first_window
```

Return Value

The global ID of the first window found in the object tree. If no window is found, the value is 0.

Example

The following example looks up the first window in the object tree:

```
atom$man_get_first_window
atom$uni_save_result
atom$uni_start_loop
atom$uni_get result
atom$if_last_return_true_then <1>
.
.
.
(atom functions to perform for each window)
.
.
.
atom$man_get_next_window
atom$uni_save_result
atom$uni_end_loop
atom$uni_sync_skip <1>
```


atom$man_get_index_by_title

37 ($25)

atom$man_get_index_by_title performs a case-sensitive search for a string in the children of a group or list, returning the index of the item in the list. If no object with the specified title is found, the atom returns 0. The current object must be a group or list existing at the time this atom is invoked.

Syntax

```
atom$man_get_index_by_title <string>
```

`<string>` Specifies the search string.

Return Value

The index of the item. If none is found, the value is 0.

Example

The following example searches the children's objects for a particular text string:

```
atom$man_get_index_by_title <"New User">
```

The result of this example is that 2 is returned if the text string New User is the second item in a group.

atom$man_get_next_response_id

56 ($38)

atom$man_get_next_response_id finds the next response ID in the magic table. Note that in addition to response IDs, the magic table contains global IDs, tokens, and active/ignored states of all the open AOL windows on the screen. This atom must be preceded by atom$man_get_first_response_id on page 6-35, which looks up the first response ID located in the magic table. This atom is commonly used in a loop when a series of atom functions must be performed on more than one window.

Syntax

```
atom$man_get_next_response_id
```

Return Value

The next response ID found in the magic table. If no response ID is found, the value is 0.

Example

The following example finds the next response ID in the magic table:

```
atom$man_get_first_response_id
atom$uni_save_result
atom$uni_start_loop
atom$uni_get result
atom$if_last_return_true_then <1>
.
.
.
(atom functions to perform for each window)
.
.
.
atom$man_get_next_response_id
atom$uni_save_result
atom$uni_end_loop
atom$uni_sync_skip <1>
```


atom$man_get_next_window
61 ($3D)

atom$man_get_next_window finds the next window in the object tree. This atom must be preceded by atom$man_get_first_window on page 6-36, which looks up the first window located under the root in the object tree. This atom is commonly used in a loop when a series of atom functions must be performed on more than one window.

Syntax

```
atom$man_get_next_window
```

Return Value

The global ID of the next window found. If no window is found, the value is 0.

Example

The following example finds the next window in the object tree:

```
atom$man_get_first_window
atom$uni_save_result
atom$uni_start_loop
atom$uni_get result
atom$if_last_return_true_then <1>
.
.
.
(atom functions to perform for each window)
.
.
.
atom$man_get_next_window
atom$uni_save_result
atom$uni_end_loop
atom$uni_sync_skip <1>
```

atom$man_get_request_window

58 ($3A)

atom$man_get_request_window finds the window that initiated a particular request. This atom is commonly used in conjunction with atom$man_get_first_response_id on page 6-35 and atom$man_get_next_response_id on page 6-38.

Syntax

```
atom$man_get_request_window <response_ID>
```

`<response_ID>` Specifies the responding window's response ID.

Return Value

The global ID of the window that initiated a request. If no initiating window is found, the value is 0.

Example

The following example finds the global ID of a window that initiated a request:

```
atom$man_get_request_window <5>
atom$uni_use_last_atom_value <atom$man_set_context_globalid>
```


atom$man_get_response_window
57 ($39)

atom$man_get_response_window finds the global ID of a window that responded to a particular request. This atom is commonly used in conjunction with atom$man_get_first_response_id on page 6-35 and atom$man_get_next_response_id on page 6-38.

Syntax

```
atom$man_get_response_window <response_ID>
```

`<response_ID>` Specifies the responding window's response ID.

Return Value

The global ID of the responding window. If no responding window is found, the value is 0.

Example

The following example finds the global ID of a window that responded to a request:

```
atom$man_get_response_window <5>
atom$uni_use_last_atom_value <atom$man_set_context_globalid>
```

atom$man_get_top_window

54 ($36)

atom$man_get_top_window finds the topmost window on the screen.

Syntax

```
atom$man_get_top_window
```

Return Value

The global ID of the topmost window. If no window is found, the value is 0.

Example

The following example searches for the topmost window:

```
atom$man_get_top_window
atom$uni_save_result
atom$if_last_return_true_then <1>
atom$uni_get_result
atom$uni_use_last_atom_value <atom$man_close_update>
atom$uni_sync_skip <1>
```


atom$man_ignore_response
59 ($3B)

atom$man_ignore_response sets the state of a particular response to be ignored. An atom stream has one of three states: pending, ignored, or magic (in focus).

Syntax

```
atom$man_ignore_response <response_id>
```

`<response_id>`	Specifies the response ID of the stream to be ignored.

Return Value

Non-zero (true) if the response is set to be ignored. If the response ID is not recognized, the value is 0 (false).

Example

The following example sets the window with a response ID of 5 to the ignored state:

```
atom$man_ignore_response <5>
```

atom$man_insert_object_after
44 ($2C)

atom$man_insert_object_after creates an object and inserts it as the next sibling of the current object. Like atom$man_start_object on page 6-92, this atom sets context to the object created so that attributes can be set for the object. Use atom$man_end_object on page 6-29 to close the object definition.

Syntax

```
atom$man_insert_object_after [<global_ID>]
```

`[<global_ID>]`	Specifies the optional global ID of the existing object.

Return Value

None.

Example

The following example creates and inserts an object as the next sibling to the current object:

```
atom$man_set_context_relative <5>
atom$man_insert_object_after
atom$mat_object_type <trigger>
.
.
.
atom$man_end_object
```


atom$man_insert_object_before

52 ($34)

atom$man_insert_object_before creates an object and inserts it as the previous sibling of the current object. Like atom$man_start_object on page 6-92, this atom sets context to the object created so that attributes can be set for the object. Use atom$man_end_object on page 6-29 to close the object definition.

Syntax

```
atom$man_insert_object_before [<global_ID>]
```

`[<global_ID>]`	Specifies the optional global ID of the existing object.

Return Value

None.

Example

The following example creates and inserts an object as the preceding sibling to the current object:

```
atom$man_set_context_relative <5>
atom$man_insert_object_before
atom$mat_object_type <trigger>
.
.
.
atom$man_end_object
```

atom$man_is_rendered

48 ($30)

atom$man_is_rendered checks the context and determines whether the current object (window) is rendered on the screen. This atom is used in special cases where you need to test to see if an object is on the screen.

Syntax

```
atom$man_is_rendered
```

Return Value

Non-zero (true) if the window is rendered. If the object is not rendered (not found), the value is 0 (false).

Example

The following example checks to see if the current object is on the screen:

```
atom$man_is_rendered
```


atom$man_is_response_ignored

63 ($3F)

atom$man_is_response_ignored checks the magic table, which keeps track of the state of all atom streams, to see if a response is ignored. A stream has one of three states: pending, ignored, or magic (in focus).

This atom is commonly used in a loop with atom$man_get_first_response_id on page 6-35 and atom$man_get_next_response_id on page 6-38.

Syntax

```
atom$man_is_response_ignored <response_id>
```

`<response_id>`	Specifies a response ID to determine whether the state of the response is ignored.

Return Value

Non-zero (true) if the response is ignored. If the response is not ignored, the value is 0 (false).

Example

The following example checks the magic table to see if the response with an ID of 7 is ignored:

```
atom$man_is_response_ignored <7>
```

atom$man_is_response_pending
62 ($3E)

atom$man_is_response_pending checks the magic table, which keeps track of the state of all atom streams, to see if a certain response is pending. A stream has one of three states: pending, ignored, or magic (in focus).

This atom is commonly used in a loop with atom$man_get_first_response_id on page 6-35 and atom$man_get_next_response_id on page 6-38.

Syntax

```
atom$man_is_response_pending <response_id>
```

`<response_id>`	Specifies a response ID to determine whether the state of the response is pending.

Return Value

Non-zero (true) if the response is pending. If the response is not pending, the value is 0 (false).

Example

The following example checks the magic table to see if response ID 7 is pending:

```
atom$man_is_response_pending <7>
```


atom$man_is_window_iconic

71 ($47)

atom$man_is_window_iconic indicates if the window is minimized.

Syntax

```
atom$man_is_window_iconic <global_ID>
```

`<global_ID>` Specifies the global ID of the window to check.

Return Value

Non-zero (true) if the window is minimized. If it is not minimized, the value is 0 (false).

Example

The following example checks a window to see if it is minimized:

```
atom$man_is_window_iconic <32-2541>
```

atom$man_item_get

32 ($20)

atom$man_item_get checks the current value of the Display Manager protocol register that you specify. This atom is used to determine the current Display Manager protocol environment.

If no argument is specified, the last register type set by a preceding atom$man_set_item_type on page 6-79 atom is used.

Syntax

```
atom$man_item_get [<dm_register>]
```

`[<dm_register>]` Specifies the Display Manager protocol register whose value you are checking. The register types are:

`dmi_session_log_id`	Session log ID
`dmi_chat_log_id`	Chat log ID
`dmi_logging_chat`	Chat session logging state
`dmi_logging_session`	Entire session logging state
`dmi_logging_ims`	Instant Message™ logging state
`dmi_on_line`	Online state
`dmi_paid_area`	Paid area state
`dmi_font_size`	Font size
`dmi_scroll_text`	Text scroll state

Return Value

The current value of the register specified. The possible values are:

Register Type	Data Type	Value Description
`dmi_session_log_id`	Numeric	The ID of the session log.
`dmi_chat_log_id`	Numeric	The ID of the chat log.
`dmi_logging_chat`	Boolean	True if a chat session is being logged; false if not.
`dmi_logging_session`	Boolean	True if an entire session is being logged; false if not.
`dmi_logging_ims`	Boolean	True if Instant Messages are being logged; false if not.
`dmi_on_line`	Boolean	True if online; false if not.
`dmi_paid_area`	Boolean	True if in a paid area; false if not.
`dmi_font_size`		The size of the font.
`dmi_scroll_text`	Boolean	True if text is scrollable; false if not.

Example

The following example checks to see if a member is online:

```
atom$man_item_get <dmi_on_line>
```

The result of this example is that non-zero (true) is returned if the member is online, and 0 (false) is returned if the member is offline.

atom$man_item_set

33 ($21)

atom$man_item_set sets the value of a particular Display Manager protocol register. This companion atom must be preceded with atom$man_set_item_type on page 6-79.

Syntax

```
atom$man_item_set <dm_reg_value>
```

`<dm_reg_value>` Specifies the value that you want to set the register to. Depending on the register type specified by **atom$man_set_item_type**, the possible values are:

Register Type	Data Type	Value Description
`dmi_session_log_id`	Numeric	The ID of the session log.
`dmi_chat_log_id`	Numeric	The ID of the chat log.
`dmi_logging_chat`	Boolean	True if a chat session is being logged; false if not.
`dmi_logging_session`	Boolean	True if an entire session is being logged; false if not.
`dmi_logging_ims`	Boolean	True if Instant Messages are being logged; false if not.
`dmi_on_line`	Boolean	True if online; false if not.
`dmi_paid_area`	Boolean	True if in a paid area; false if not.
`dmi_font_size`		The size of the font.
`dmi_scroll_text`	Boolean	True if text is scrollable; false if not.

Return Value

None.

Example

The following example sets the value of the `dmi_on_line` register to a true (1) state:

```
atom$man_set_item_type <dmi_on_line>
atom$man_item_set <1>
```

atom$man_kill_timer

77 ($4D)

atom$man_kill_timer disables the timer for the current object.

Syntax

```
atom$man_kill_timer
```

Return Value

None.

Example

The following example disables the timer for the current object:

```
atom$man_set_context_relative <1>
atom$mat_timer_duration <1000>
atom$mat_timer_event <200>
.
.
.
atom$man_kill_timer
atom$man_end_context
```


atom$man_logging_command

36 ($24)

atom$man_logging_command determines what type of logging to perform. Members access logging from the File menu of the online service.

Syntax

```
atom$man_logging_command <command>
```

`<command>` Specifies the logging command to execute. Logging command values are:

`0`	`man_logging_open`	Opens a logging file.
`1`	`man_logging_append`	Appends to a logging file.
`2`	`man_logging_close`	Closes a logging file.
`3`	`man_log_type_session`	Indicates that succeeding **atom$man_logging_command** atoms refer to the logging of the session (see example below).
`4`	`man_log_type_chat`	Indicates that succeeding **atom$man_logging_command** atoms refer to the logging of chat conversation.
`5`	`man_log_type_im`	Begins logging Instant Messages.
`6`	`man_log_type_im_off`	Stops logging Instant Messages.
`7`	`man_logging_flush`	Clears all logging files.

Return Value

None.

Example

The following example opens a log file for session logging:

```
atom$fm_create_file
atom$man_logging_command <3>
atom$man_logging_command <0>
atom$man_set_context_relative <1>
atom$fm_item_get <filename>
atom$uni_use_last_atom_string <atom$mat_title>
atom$man_end_context
```


atom$man_make_focus

53 ($35)

atom$man_make_focus marks an object to receive focus the next time a display update of the subtree is performed. When a window receives focus, the window moves in front of all other online service windows. When an object in a window receives focus, a dotted line is typically drawn around the object.

Syntax

```
atom$man_make_focus [<global_ID>]
```

`[<global_ID>]`	Specifies the optional global ID of the object to receive focus.

Return Value

None.

Example

The following example sets focus on window 32-234:

```
atom$man_set_context_globalid <32-234>
atom$man_make_focus
atom$man_update_display <32-234>
```

atom$man_obj_stack_pop

98 ($62)

atom$man_obj_stack_pop pops, peeks, or clears the top object on the global stack. If the argument is omitted, this atom removes (pops) the top value off the stack and returns the value.

Syntax

```
atom$man_obj_stack_pop [<word]
```

`[<word>]` Specifies the optional stack operation type. Values are:

`0`	Peek	Returns the value from the top of the stack without removing it.
`1`	Pop	Returns the value at the top and removes it from the stack (default).
`2`	Clear	Clears the entire stack of all values.

Return Value

Returns the 32-bit value or global ID from the top of the stack. If the stack is cleared (empty), the value is 0.

Example

The following example pops the value at the top of the stack:

```
atom$man_start_ip_session <59x 23x 14x 00x 01x 4Fx>
atom$man_obj_stack_pop
atom$uni_end_stream
```


atom$man_obj_stack_push
97 ($61)

atom$man_obj_stack_push pushes a 32-bit value onto a global stack. If the argument is omitted, the value pushed onto the stack is the global ID (GID) of the current object (form).

Syntax

```
atom$man_obj_stack_push [<dword>]
```

`[<dword>]`	Specifies the 32-bit value to be pushed to the stack. If the optional argument is omitted, the global ID of the current object is pushed to the stack.

Return Value

Returns the 32-bit value or global ID pushed to the stack. If there is no object in context or an error has occurred, the value is 0.

Example

The following example pushes GID 20-0-00335 (14x 00x 01x 4Fx) to the stack:

```
atom$man_start_ip_session <59x 23x 14x 00x 01x 4Fx>
atom$man_obj_stack_push <14x 00x 01x 4Fx>
atom$uni_end_stream
```

atom$man_place_cursor

24 ($18)

atom$man_place_cursor is a shorthand atom that immediately sets focus to an object that does not require a display update.

Syntax

```
atom$man_place_cursor <relative_ID>
```

`<relative_ID>`	Specifies the relative ID of the object to receive focus. Context must be set to a parent or ancestor of the desired object. (The independent object is usually used.)

Return Value

The relative ID of the object in focus.

Example

The following example immediately sets focus on the object that has a relative ID of 4:

```
atom$man_set_context_globalid <32-234>
atom$man_place_cursor <4>
```


atom$man_post_update_gid
72 ($48)

atom$man_post_update_gid does a post version of **atom$man_update_display**. This atom allows the current stream, all other streams, all pending client messages, and all currently pending client post messages to be processed by the client before performing a routine of **atom$man_update_display** on the argument.

Unlike **atom$man_update_display**, this atom has a mandatory GID argument of the form to be updated (a non-argument does not imply the current form).

Syntax

```
atom$man_post_update_gid <$gid>
```

`<$gid>` Specifies the gid.

Return Value

None.

Example

The following example posts the update gid:

```
atom$man_post_update_gid <$gid>
```

atom$man_preset_authoring_form

83 ($53)

atom$man_preset_authoring_form designates an authoring form for the next object to be created. This atom is used only by Visual Publisher Designer (VPD) developers.

Syntax

```
atom$man_preset_authoring_form <string>
```

`<string>` Specifies the text for the next object to be created.

Return Value

None.

Example

The following example designates an authoring form for the next object to be created:

```
atom$man_present_authoring_form
```


atom$man_preset_gid

22 ($16)

atom$man_preset_gid designates a global ID for the next object to be created. If the specified global ID is already assigned to an object in the display tree, the context is set to that object.

Syntax

```
atom$man_preset_gid <global_ID>
```

`<global_ID>` Specifies the global ID for the next object.

Return Value

The global ID when the object already exists. If the global ID does not exist, the value is 0.

Example

The following example presets the global ID for the next object that is created:

```
atom$man_preset_gid <32-123>
atom$man_start_object <ind_group>
```

atom$man_preset_relative

51 ($33)

atom$man_preset_relative designates the relative ID to be used for the next object created. If an object with that relative ID already exists as a child or descendent of the current object, the context is set to that object.

Syntax

```
atom$man_preset_relative <relative_ID>
```

`<relative_ID>`	Specifies the relative ID to be used for the next object.

Return Value

The relative ID when the object already exists. If the relative ID does not exist, the value is 0.

Example

The following example presets the relative ID for the next object that is created:

```
atom$man_preset_relative <4>
atom$man_start_object <ornament>
```


atom$man_preset_title

23 ($17)

atom$man_preset_title designates a title to be used for the next object created.

Syntax

```
atom$man_preset_title <string>
```

`<string>` Specifies the title.

Return Value

The title string.

Example

The following example presets the title (People Connection) for the next object that is created:

```
atom$man_preset_title <"People Connection">
atom$man_start_object <ind_group>
```

atom$man_preset_url

90 ($5A)

atom$man_preset_url designates an URL to be used for the next object created.

Syntax

```
atom$man_preset_url <URL>
```

`<URL>` Specifies the URL.

Return Value

None.

Example

The following example designates an URL to be used for the next object created:

```
atom$uni_start_stream <00x>
atom$man_preset_gid <32-6123>
atom$uni_start_stream <>
atom$man_set_default_title <"View Buddies">
atom$man_preset_url <"aol://1722:buddyview">
atom$uni_end_stream <>
```


atom$man_replace_data

21 ($15)

atom$man_replace_data replaces the content of the current object (usually a view). If the object has already been rendered, the replaced data does not take effect until a display update of the associated subtree is performed.

Syntax

```
atom$man_replace_data <data>
```

`<data>`	Specifies the data in any number of bytes.

Return Value

None.

Example

The following example replaces the data for the current object:

```
atom$man_replace_data <"202-555-0956">
```

The result of this example is that the text in the current view is replaced with 202-555-0956.

atom$man_response_pop
27 ($1B)

atom$man_response_pop destroys the window containing the trigger that initiated a request. Typically, it is sent by the host as part of a response to a request to pop a form. An atom$man_set_response_id on page 6-80 atom must precede this atom in the stream so that the response is associated with the request.

Syntax

```
atom$man_response_pop [<response_ID>]
```

`<response_ID>` Specifies an optional response ID to override the preceding `atom$man_set_response_id` atom.

Return Value

The response ID; otherwise, 0.

Example

The following example destroys a window (ID 2014) that initiated a request:

```
atom$man_set_response_id <2014>
.
.
.
atom$man_response_pop
```


atom$man_set_append_secure_data
101 ($65)

atom$man_set_append_secure_data displays the information provider's (IP) brand ID in a field on a form. This atom lets the online service send secure field data between an information provider and client applications through the **rmg** server on the host.

Syntax

```
atom$man_set_append_secure_data <brand_id>
```

`<brand_id>`	Specifies the brand ID for the form. Brand IDs can be up to 80 characters in length.

Return Value

None.

Example

The following example displays the brand ID (AolVisa) in the current field:

```
atom$man_start_object <view, "">
atom$man_set_append_secure_data <"AolVisa">
atom$mat_bool_writeable <no>
atom$man_end_object
```

atom$man_set_context_globalid
9 ($09)

atom$man_set_context_globalid sets context to the object specified with the global ID. If the object is not found, the stream is terminated.

Syntax

```
atom$man_set_context_globalid <global_ID>
```

`<global_ID>` Specifies the global ID.

Return Value

The global ID.

Example

The following example sets the context to an object that has a global ID of 32-123:

```
atom$man_set_context_globalid <32-123>
atom$man_set_context_relative <50>
.
.
.
atom$man_end_context
```


atom$man_set_context_first_selection

111 ($6F)

atom$man_set_context_first_selection is used for handling the walking through of tree controls or list boxes.

Syntax

```
atom$man_set_context_first_selection <RID>
```

`<RID>` Specifies the RID for tree controls or list boxes.

Return Value

atom$man_set_context_first_selection returns true if successful and false if failed.

Object(s) Handled

The objects handled are `treecontrols` and `listboxes`.

Example

The following example sets the conext for the first command in the tree control action for the object with the specified RID:

```
atom$uni_start_stream
atom$man_set_context_first_selection 10
atom$if_last_return_true_then <20>
atom$uni_start_loop
atom$man_treectrl_action_command <1>
atom$man_end_context
atom$man_set_context_next_selection
atom$if_last_return_true_then <10>
atom$uni_end_loop
atom$uni_sync_skip <10>
atom$man_end_context
atom$uni_sync_skip <20>
atom$uni_end_stream
atom$man_end_object
```

atom$man_set_context_index

11 ($0B)

atom$man_set_context_index sets context to one of the children of the current object or list using the index number of the child. The current object must be a group or list. The stream is terminated if the index entry is out of range.

Syntax

```
atom$man_set_context_index <index_#>
```

`<index_#>` Specifies the index number of the child object.

Return Value

The index number.

Example

The following example sets the context to the third object in the current group:

```
atom$man_set_context_index <3>
.
.
.
atom$man_end_context
```


atom$man_set_context_next_selection
112 ($70)

atom$man_set_context_next_selection is used for handling the walking through of tree controls or list boxes.

Syntax
```
atom$man_set_context_next_selection
```

Return Value

None.

Object(s) Handled

The objects handled are `treecontrols` and `listboxes`.

Example

The following example sets the conext for the next command in the tree control action:

```
atom$uni_start_stream
atom$man_set_context_first_selection 10
atom$if_last_return_true_then <20>
atom$uni_start_loop
atom$man_treectrl_action_command <1>
atom$man_end_context
atom$man_set_context_next_selection
atom$if_last_return_true_then <10>
atom$uni_end_loop
atom$uni_sync_skip <10>
atom$man_end_context
atom$uni_sync_skip <20>
atom$uni_end_stream
```

atom$man_set_context_relative

10 ($0A)

atom$man_set_context_relative sets the context to one of the children or descendants of the current object using a relative ID. This atom is used with group, independent, or list objects. The stream is terminated if no object with the specified relative ID is found.

Syntax

```
atom$man_set_context_relative <relative_ID>
```

`<relative_ID>` Specifies the relative ID.

Return Value

The relative ID.

Example

The following example sets the context to an object that has a relative ID of 2:

```
atom$man_set_context_relative <2>
```

The result of this example is that context is set to the object with an ID of 2 on the current window.

atom$man_set_context_response_id
8 ($08)

atom$man_set_context_response_id sets the context to the independent ancestor of the object that initiated a request. The request is identified by the response ID defined by atom$man_set_response_id on page 6-80 which was executed earlier in the stream. Optionally, an argument may be supplied to set context to a different response ID.

Syntax

```
atom$man_set_context_response_id [<response_ID>]
```

`[<response_ID>]` Specifies an optional override response ID.

Return Value

The response ID given in the argument; otherwise, 0.

Example

The following example sets the context to the independent ancestor of an object that triggered a response:

```
atom$man_set_response_id <1234>
.
.
.
atom$man_set_context_response_id
```

atom$man_set_default_title

66 ($42)

atom$man_set_default_title changes the text of the default title, which is stored in a buffer.

Normally, when a trigger (namely, a button) is selected, the text of the trigger becomes the default title for the next window. For example, if a member selects a button with a title of Unread Articles, the title of the resulting window also says Unread Articles. **atom$man_set_default_title** can be used to override the default title.

Syntax

```
atom$man_set_default_title <string>
```

`<string>`	Specifies the new title of the current object.

Return Value

None.

Example

The following example replaces the current object's default title with the new title, Exciting Articles to Read:

```
atom$man_set_default_title <"Exciting Articles to Read">
```


atom$man_set_edit_position

81 ($51)

atom$man_set_edit_position, along with **atom$man_set_edit_position_to_end,** can only be used on an input_box object; these atoms replace the cursor at the position specified. **atom$man_set_edit_position** takes a word argument specifying the number of characters from the beginning of the current string in the **input_box.**

Syntax

```
atom$man_set_edit_position <$dword>
```

`<$dword>`	Specifies the text string in which the cursor is placed.

Return Value

None.

Example

The following example specifies that the cursor falls after the 14th character of the string in the input_box:

```
atom$man_set_edit_position <$dwordxxxxxxxxx>
```

atom$man_set_edit_position_to_end
82 ($52)

atom$man_set_edit_position_to_end, along with **atom$man_set_edit_position**, can only be used on an input_box object; these atoms replace the cursor at the position specified. **atom$man_set_edit_position_to_end** takes a word argument specifying the number of characters from the beginning of the current string in the input_box.

Syntax

```
atom$man_set_edit_position_to_end <$dword>
```

`<$dword>`	Specifies the text string at whose end the cursor falls.

Return Value

None.

Example

The following example specifies that the cursor falls at the end of the string in the input_box:

```
atom$man_set_edit_position <$dword>
```


atom$man_set_item_type

65 ($41)

atom$man_set_item_type specifies a particular Display Manager protocol register whose value is to be set or changed. This companion atom must precede atom$man_item_set on page 6-52.

Syntax

```
atom$man_set_item_type <dm_register>
```

`<dm_register>` Specifies the particular register whose value you want to set. Register types and their function are as follows:

`dmi_session_log_id`	Session log ID
`dmi_chat_log_id`	Chat log ID
`dmi_logging_chat`	Usage state of the chat log
`dmi_logging_session`	Usage state of the session log
`dmi_logging_ims`	Usage state of the Instant Messages log
`dmi_on_line`	Online state
`dmi_paid_area`	Paid area state
`dmi_font_size`	Font size
`dmi_scroll_text`	Text scroll state

Return Value

None.

Example

The following example defines the type of a register as `dmi_on_line` to be set to a true (1) value.

```
atom$man_set_item_type <dmi_on_line>
atom$man_item_set <1>
```

atom$man_set_response_id
7 ($07)

atom$man_set_response_id notifies the client computer that the current stream is a response to the client computer's request. Each object has an associated response ID. When a special buffer flag is set (or any shorthand action is used), the response ID precedes any data that is sent to the host.

Additionally, if a magic action is used, an entry is added to the magic table indicating that a response is expected for the requesting object's response ID. When a stream is later received containing an a**tom$man_set_response_id** atom for that response ID, the next independent object to be created is associated in the magic table with the requesting object.

This association allows a number of operations to be performed:

- Magic is facilitated.
- The requesting window can be popped (destroyed).
- The requesting window can be modified.

The magic table entry is retained until the resulting independent object is deleted.

Syntax

```
atom$man_set_response_id <response_ID>
```

`<response_ID>` Specifies the response ID.

Return Value

None.

Example

The following example associates response ID 1234 with a trigger object on the requesting window:

```
atom$uni_start_stream
atom$man_start_object <trigger>
atom$man_set_response_id <1234>
atom$man_end_object
atom$uni_end_stream
```


atom$man_sort_items
109 ($6D)

atom$man_sort_items is used for generic sorting of objects. Atom used to sort

Syntax

```
atom$man_sort_items <sorting_flags>
```

`<sorting_flags>` Specifies the sorting flags as follows:

`byte0`	Ascending/descending
`byte1`	Field number
`byte2`	Delimiter
`byte3`	Permanent flag (default is No)

Return Value

None.

Objects Handled

The objects handled are `Tree Control` and `Listbox`.

Example

The following example sets the sort flag by field number:

```
atom$man_start_object <trigger> "Sort"
atom$act_replace_select_action
atom$uni_start_stream
atom$man_set_context_relative <30>
atom$man_sort_items <1>
atom$man_end_context
atom$uni_end_stream
atom$man_end_object
```

atom$man_spell_check

96 ($60)

atom$man_spell_check invokes the current object as a spell-check form from a rich view.

Syntax

```
atom$man_spell_check
```

Return Value

None.

Example

The following example invokes a spell-check form:

```
atom$uni_start_stream
atom$man_set_context relative <12>
atom$man_spell_check
atom$man_end_context
atom$uni_end_stream
```


atom$man_start_alpha

40 ($28)

atom$man_start_alpha creates an object as a child of the current object. The position of the object relative to its siblings is determined alphabetically. This atom requires that the object's title be specified as part of the atom or in a preceding atom$man_preset_title on page 6-65.

Syntax

```
atom$man_start_alpha <object_type[, string]>
```

`<object_type>` Specifies an object type. Values are as follows:

`0`	`org_group`	An organizational group
`1`	`ind_group`	An independent group (for example, a window)
`2`	`dms_list`	A dynamic, multi-select list (for example, a list box where more than one item is selectable)
`3`	`sms_list`	A static, multi-select list (for example, a check box)
`4`	`dss_list`	A dynamic, single-select list (for example, a list box where only one item is selectable)
`5`	`sss_list`	A static, single-select list (for example, radio buttons)
`6`	`trigger`	An actionable object (for example, a button, icon, or menu item)
`7`	`ornament`	A static or decorative object (for example, a picture, or static text)

8	view	A view field (used for display of larger amounts of text or a graphic where, if necessary, scroll bars are available)
9	edit_view	An editable view field (used for input text fields where members can enter text)
12	range	A range (used for display of graphic representation of a value, such as a bar gauge)
13	select_range	A selectable range (used for an easy user interface for entering numeric data, such as a spin gadget)
17	tool_group	A toolbar used on any form or frame window including tab controls
18	tab_group	A tab group used for tab control of a group of tabbed pages, frames or forms (for example, list boxes with tabs)
19	tab_page	A tab page used as a tabbed page, frame or form such as a list box with a labeled tab

`[<string>]` Specifies an optional title.

Return Value

None.

Example

The following example creates a new object as a child of the current object and inserts it alphabetically among its siblings:


```
atom$man_start_alpha <trigger, "The Wine Club">
```

The result of this example is that a trigger object with the title The Wine Club is created and inserted alphabetically among its siblings.

atom$man_start_first

34 ($22)

atom$man_start_first creates an object and inserts it as the first child of the current object.

Syntax

```
atom$man_start_first <object_type[, string]>
```

`<object_type>` Specifies an object type. Values are as follows:

`0`	`org_group`	An organizational group
`1`	`ind_group`	An independent group (for example, a window)
`2`	`dms_list`	A dynamic, multi-select list (for example, a list box where more than one item is selectable)
`3`	`sms_list`	A static, multi-select list (for example, a check box)
`4`	`dss_list`	A dynamic, single-select list (for example, a list box where only one item is selectable)
`5`	`sss_list`	A static, single-select list (for example, radio buttons)
`6`	`trigger`	An actionable object (for example, a button, icon, menu item)
`7`	`ornament`	A static or decorative object (for example, a picture, static text)
`8`	`view`	A view field (used for display of larger amounts of text or a graphic where, if necessary, scroll bars are available)

9	edit_view	An editable view field (used for input text fields where members can enter text)
12	range	A range (used for display of graphic representation of a value, such as a bar gauge)
13	select_range	A selectable range (used for an easy user interface for entering numeric data, such as a spin gadget)
17	tool_group	A toolbar used on any form or frame window including tab controls
18	tab_group	A tab group used for tab control of a group of tabbed pages, frames or forms (for example, list boxes with tabs)
19	tab_page	A tab page used as a tabbed page, frame or form such as a list box with a labeled tab

`[<string>]` Specifies an optional title.

Return Value

None.

Example

The following example creates a new object as a child of the current object and inserts it before its siblings:

```
atom$man_start_first <trigger, "The Wine Club">
```

The result of this example is that a trigger object with the title The Wine Club is created and inserted first, before its siblings.

atom$man_start_ip_session
103 ($67)

atom$man_start_ip_session establishes a secure information provider (IP) session between the client and the host. This atom is sent by the host as part of a form definition and must be paired with atom$man_set_append_secure_data on page 6-69 to create an environment that can be completed with a handshake between the client and the IP through the **rmg** server.

Syntax

```
atom$man_start_ip_session <token/gid>
```

`<token>` Specifies a hex token value in the first 2 bytes of the argument. This value defines a routing token to the host server the client should use regarding this session. The value is a code of 2 ASCII characters. For example, the value 59x 23x defines a routing token of Y#.

`<gid>` Specifies a hex global ID (GID) value in the last 4 bytes of the argument. This value defines the GID of the cache record in the database that the client must use regarding this session. For example, the value 14x 00x 01x 4Fx defines a global ID of 20-0-00335, where 14x equals 20, 00x equals 0, and 01x 4Fx equals 00335.

Return Value

None.

Example

The following example establishes an IP session to transmit secure data to the client application using a Y# routing token and a GID of 20-0-00335:

```
atom$man_start_ip_session <59x 23x 14x 00x 01x 4Fx>
atom$man_start_object <edit_view, "">
atom$mat_size <3Cx 0Ex>
atom$mat_secure_field <0>
atom$mat_bool_writeable <yes>
atom$man_end_object
```


atom$man_start_last

41 ($29)

atom$man_start_last creates an object and inserts it as the last child of the current object. This is the default behavior of **atom$man_start**. The command **atom$man_start_last** is provided as a means of overriding a list's sort order when necessary.

Syntax

```
atom$man_start_last <object_type[, string]>
```

`<object_type>` Specifies an object type. Values are as follows:

`0`	`org_group`	An organizational group
`1`	`ind_group`	An independent group (for example, a window)
`2`	`dms_list`	A dynamic, multi-select list (for example, a list box where more than one item is selectable)
`3`	`sms_list`	A static, multi-select list (for example, a check box)
`4`	`dss_list`	A dynamic, single-select list (for example, a list box where only one item is selectable)
`5`	`sss_list`	A static, single-select list (for example, radio buttons)
`6`	`trigger`	An actionable object (for example, a button, icon, menu item)
`7`	`ornament`	A static or decorative object (for example, a picture, static text)

8	view	A view field (used for display of larger amounts of text or a graphic where, if necessary, scroll bars are available)
9	edit_view	An editable view field (used for input text fields where members can enter text)
12	range	A range (used for display of graphic representation of a value, such as a bar gauge)
13	select_range	A selectable range (used for an easy user interface for entering numeric data, such as a spin gadget)
17	tool_group	A toolbar used on any form or frame window including tab controls
18	tab_group	A tab group used for tab control of a group of tabbed pages, frames or forms (for example, list boxes with tabs)
19	tab_page	A tab page used as a tabbed page, frame or form such as a list box with a labeled tab

`[<string>]` Specifies an optional title.

Return Value

None.

Example

The following example creates a new object as a child of the current object and inserts it after its siblings:


```
atom$man_start_last <trigger, "The Wine Club">
```

The result of this example is that a trigger object with the title The Wine Club is created and inserted last, after its siblings.

atom$man_start_object
0 ($00)

atom$man_start_object creates an object as a child of the current object. The position of the new object relative to other children of the current object is dependent on the sort order of the current object. If no sort order has been explicitly set for the object, children are added after all other siblings (the equivalent of atom$man_start_first on page 6-86 behavior).

Syntax

```
atom$man_start_object <object_type[, string]>
```

`<object_type>` Specifies an object type. Values are as follows:

`0`	`org_group`	An organizational group
`1`	`ind_group`	An independent group (for example, a window)
`2`	`dms_list`	A dynamic, multi-select list (for example, a list box where more than one item is selectable)
`3`	`sms_list`	A static, multi-select list (for example, a check box)
`4`	`dss_list`	A dynamic, single-select list (for example, a list box where only one item is selectable)
`5`	`sss_list`	A static, single-select list (for example, radio buttons)
`6`	`trigger`	An actionable object (for example, a button, icon, menu item)
`7`	`ornament`	A static or decorative object (for example, a picture, static text)
`8`	`view`	A view field (used for display of larger amounts of text or a graphic where, if necessary, scroll bars are available)

9	edit_view	An editable view field (used for input text fields where members can enter text)
12	range	A range (used for display of graphic representation of a value, such as a bar gauge)
13	select_range	A selectable range (used for an easy user interface for entering numeric data, such as a spin gadget)
17	tool_group	A toolbar used on any form or frame window including tab controls
18	tab_group	A tab group used for tab control of a group of tabbed pages, frames or forms (for example, list boxes with tabs)
19	tab_page	A tab page used as a tabbed page, frame or form such as a list box with a labeled tab

`[<string>]` Specifies an optional title.

Return Value

None.

Example

The following example defines a new trigger object:

```
atom$man_start_object <trigger>
.
.
.
atom$man_end_object
```

atom$man_start_sibling
1 ($01)

atom$man_start_sibling creates an object and attaches it to the display tree as the next sibling of the current object. An atom$man_end_object on page 6-29 for the current object is implied.

Syntax

```
atom$man_start_sibling <object_type[, string]>
```

`<object_type>` Specifies an object type. Values are as follows:

`0`	`org_group`	An organizational group
`1`	`ind_group`	An independent group (for example, a window)
`2`	`dms_list`	A dynamic, multi-select list (for example, a list box where more than one item is selectable)
`3`	`sms_list`	A static, multi-select list (for example, a check box)
`4`	`dss_list`	A dynamic, single-select list (for example, a list box where only one item is selectable)
`5`	`sss_list`	A static, single-select list (for example, radio buttons)
`6`	`trigger`	An actionable object (for example, a button, icon, menu item)
`7`	`ornament`	A static or decorative object (for example, a picture, static text)
`8`	`view`	A view field (used for display of larger amounts of text or a graphic where, if necessary, scroll bars are available)
`9`	`edit_view`	An editable view field (used for input text fields where members can enter text)

12	range	A range (used for display of graphic representation of a value, such as a bar gauge)
13	select_range	A selectable range (used for an easy user interface for entering numeric data, such as a spin gadget)
17	tool_group	A toolbar used on any form or frame window including tab controls
18	tab_group	A tab group used for tab control of a group of tabbed pages, frames or forms (for example, list boxes with tabs)
19	tab_page	A tab page used as a tabbed page, frame or form such as a list box with a labeled tab

`[<string>]` Specifies an optional title.

Return Value

None.

Example

The following example creates a new object as a sibling of the current object and inserts it next in the object tree:

```
atom$man_start_sibling <trigger, "Software Center">
```

The result of this example is that a trigger object with the title Software Center is created and inserted after the current object.

atom$man_treectrl_action_command
110 ($6E)

atom$man_treectrl_action_command is used for `tree_control` to create various actions.

Syntax

```
atom$man_treectrl_action-command <object_type[, string]>
```

`<command>` Specifies the commands such as edit, add, or delete with an integer as follows:

`1`	action: edit the item's label
`2`	action: add an item to the tree control
`N`	action: as defined

Return Value

None.

Object(s) Handled

This atom handles the `tree_control` object.

Example

The following example creates the action to edit the item’s label:

```
atom$man_start_object <trigger> "Edit"
atom$act_replace_select_action
atom$uni_start_stream
atom$man_treectrl_action_command < 1>
atom$uni_end_stream
atom$man_end_object
```


atom$man_update_display
17 ($11)

atom$man_update_display visually updates the objects contained in a subtree of the display tree.

Syntax

```
atom$man_update_display [<global_ID>]
```

`[<global_ID>]`	Specifies the optional global ID of the subtree to be updated. If the argument is omitted, the root of the subtree (to update) defaults to the current object.

Return Value

None.

Example

The following example visually updates the display of objects in the current object (ID 61):

```
atom$uni_start_stream
atom$man_set_context_relative <61>
atom$mat_bool_repeat_animation <no>
atom$mat_art_animation_rate <0>
atom$mat_art_frame <0>
atom$man_update_display
atom$man_end_context
atom$uni_end_stream
```

atom$man_update_end_object

19 ($13)

atom$man_update_end_object performs a visual update of the subtree of the display tree and also ends the current object context. The current object is used as the root of the subtree to update.

Syntax

```
atom$man_update_end_object
```

Return Value

None.

Example

The following example ends the current object and visually updates the display of objects in a particular subtree:

```
atom$uni_start_stream
atom$man_set_context_relative <61>
atom$mat_bool_repeat_animation <no>
atom$mat_art_animation_rate <0>
atom$mat_art_frame <0>
atom$man_update_end_object
atom$uni_end_stream
```


atom$man_update_fonts
74 ($4A)

atom$man_update_fonts reloads font information from the database record that contains it. The database record is changed using atoms from the Database Manager command set, and then **atom$man_update_fonts** is invoked.

Syntax

```
atom$man_update_fonts
```

Return Value

None.

Example

The following example reloads font information:

```
atom$man_update_fonts
```

atom$man_update_woff_end_stream
18 ($12)

atom$man_update_woff_end_stream is a shorthand atom that performs a visual update of the current object and all of its descendants. Then it turns the global wait state off and terminates the stream.

Syntax

```
atom$man_update_woff_end_stream
```

Return Value

None.

Example

The following example visually updates the current object and all of its descendants, turns off the global wait state, and terminates the atom stream:

```
atom$man_start_sibling <trigger, “?”>
atom$act_replace_select_action
atom$uni_start_stream_wait_on
atom$sm_send_k1 <2-4936>
atom$mat_trigger_style <rectangle>
atom$mat_font_sis <arial, 12, bold>
atom$man_end_object
atom$man_update_woff_end_stream
```


atom$man_use_default_title

16 ($10)

atom$man_use_default_title presets the title for an object about to be created to the saved title. For example, when a trigger's action is invoked, the title of the trigger is saved for later reference.

Syntax

```
atom$man_use_default_title
```

Return Value

The default title string.

Example

The following example presets the title of an object to a title that has already been saved:

```
atom$man_use_default_title
atom$man_start_object <ind_group>
.
.
.
atom$man_end_object
```

MAN_SORT_ITEMS

ID#: 109

Return Value: None

Objects Handled: Tree Control, Listbox

Description: This atom is used for generic sorting. Atom used to sort

objects by:

Arguments: <sorting_flags> dword specifying the sorting flags.

byte0: ascending | descending

byte1: field number

byte2: delimiter

byte3: Permanent flag, Default= NO

Example of Usage:

atom$man_start_object <trigger> "Sort"

atom$act_replace_select_action

atom$uni_start_stream

atom$man_set_context_relative <30>

atom$man_sort_items <1>

atom$man_end_context

atom$uni_end_stream

atom$man_end_object

MAN_TREECTRL_ACTION_COMMAND

ID#: 110

Return Value: None

Object Handled: Tree Control

Description: This atom is used for tree control to create various actions.

Arguments: <command> integer representing actions like:

edit | add | delete

we may add other actions later.

command =1 means action: edit the item's label

command =2 means action: add an item to the tree control.

and so on.

Example of Usage:


```
atom$man_start_object <trigger> "Edit"
atom$act_replace_select_action
  atom$uni_start_stream
    atom$man_treectrl_action_command < 1>
  atom$uni_end_stream
atom$man_end_object
```

atom$man_sort_items
109 ($6D)

atom$man_sort_items.

Syntax

```
atom$man_
```

Return Value

Example

The following example

MAN_SORT_ITEMS

ID#: 109

Return Value: None

Objects Handled: Tree Control, Listbox

Description: This atom is used for generic sorting. Atom used to sort objects by:

Arguments: <sorting_flags> dword specifying the sorting flags.

byte0: ascending | descending

byte1: field number

byte2: delimiter

byte3: Permanent flag, Default= NO

Example of Usage:

atom$man_start_object <trigger> "Sort"

atom$act_replace_select_action

atom$uni_start_stream

atom$man_set_context_relative <30>

atom$man_sort_items <1>

atom$man_end_context

atom$uni_end_stream

atom$man_end_object

atom$man_
? ($?)

atom$man_use_.

Syntax

```
atom$man_
```

Return Value

Example

The following example

