Manual: FDO91 Manual
Chapter 2: Action Protocol defines the action protocol atoms and provides information for associating actions with FDO91 objects.
Last updated: October 1997

CHAPTER 2
Action (ACT) Protocol

The Action (ACT) protocol (protocol ID 2) consists of atoms that allow actions to be associated with objects on forms. For example, you can set up an action to occur as the result of a member selecting an object, such as an icon on a form.

Action Criteria

There are two components to setting up an object's action: the action criterion, which is specified using **atom$act_set_criterion**, and the atom stream (sometimes referred to as an action stream), which is specified by a series of atoms that begins with **atom$uni_start_stream** and ends with **atom$uni_ end_stream**.

Action criteria let you assign an action ID that indicates when an object's action will occur, while the atom stream lets you define what action will occur. In most cases, an action is set up when its associated object is created.

Table 2.1 lists the title, the action ID, the hexadecimal equivalent of the action ID, and a description for each action criterion:

Table 2.1 – Action Criteria

Action Criterion/ Action ID	Description
SELECTION 1 (01x)	Action occurs when an object is selected.
CLOSE 2 (02x)	Action occurs when an object is closed.

Table 2.1 – Action Criteria (Continued)

Action Criterion/ Action ID	Description
GAIN_FOCUS 4 (04x)	Action occurs when a window is brought to the front.
LOSE_FOCUS 5 (05x)	Action occurs when another window is brought to the front of the current window.
CANCEL 6 (06x)	Action occurs when ESC is pressed while the window is the top-most window.
ENTER_FREE 7 (07x)	Action occurs when a free area is entered. This criterion applies to independent groups (windows) and affects the default behavior of switching between free and paid areas.
ENTER_PAID 8 (08x)	Action occurs when a paid area is entered. This criterion applies to independent groups (windows) and affects the default behavior of switching between free and paid areas.
CREATE 9 (09x)	Action occurs when a window is first drawn.
SET_ONLINE 10 (0Ax)	Action occurs when a member goes online. This criterion applies to independent groups (windows).
SET_OFFLINE 11 (0Bx)	Action occurs when a member goes offline. This criterion applies to independent groups (windows).
RESTORE 12 (0Cx)	Action occurs when a window is restored.
MINIMIZE 14 (0Ex)	Action occurs when a window is minimized.

Table 2.1 – Action Criteria (Continued)

Action Criterion/ Action ID	Description
RESTORE_FROM_ MAXIMIZE 15 (0Fx)	Action occurs when a window is restored to its original size after being maximized.
RESTORE_FROM_ MINIMIZE 16 (10x)	Action occurs when a window is restored to its original size after being minimized.
TIMEOUT 17 (11x)	Action occurs when the host fails to respond. This criterion lets you override the default behavior for what happens when the host does not respond.
SCREEN_NAME_ CHANGED 18 (12x)	Action occurs when a screen name is changed. This criterion applies to independent groups (windows) and affects all windows open when the action occurs.
MOVIE_OVER 19 (13x)	Action occurs when a **.avi** file has finished playing.
DROP 20 (14x)	Action occurs when data is dropped into an object (field or view) and the mouse button is released.
URL_DROP 21 (15x)	Action occurs when an URL is dropped into an object and the mouse button is released.
USER_DELETE 22 (16x)	Action occurs when a user presses DELETE or clicks a mouse to delete something.
TOGGLE_UP 23 (17x)	Action occurs when a user activates the up action of a toggle switch.
ACTIVATED 24 (18x)	Actions occurs when MDI_ACTIVATE is true.
DEACTIVATED 25 (19x)	Action occurs when MDI_ACTIVATE is false.

Table 2.1 – Action Criteria (Continued)

Action Criterion/ Action ID	Description
POPUPMENU 26 (1Ax)	Action occurs when a user invokes a pop-up menu.
DESTROYED 27 (1Bx)	Action occurs when a window is cleared from the screen.
ACTION_TOOL 128-255 (80x-FFx)	Reserved for protocol-independent action criterion, e.g., action ID 128 in the MORG protocol is ACTION_VIEW_UPDATED. Action ID 128 in the WWW protocol is ACTION_URL_CHANGED.

Action Protocol Atoms

The Action (ACT) protocol functions and their associated atoms follow:

Function	Atoms
Saving action streams	atom$act_copy_action_to_reg atom$act_set_action_in_reg
Selecting action streams	atom$act_do_action atom$act_replace_action atom$act_replace_action_from_reg atom$act_replace_popup_menu_action atom$act_replace_select_action
Setting triggers for action streams	atom$act_set_criterion
Searching for action streams	atom$act_set_inheritance

Using database records	atom$act_get_db_record atom$act_get_db_value atom$act_set_db_id atom$act_set_db_length atom$act_set_db_offset atom$act_set_db_record
Making sound actions	atom$act_sound_beep
Setting new user and guest flags	atom$act_set_guest_flag atom$act_set_newuser_flag

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

atom$act_append_action
32 ($20)

This atom is no longer supported. This atom appended the specified data to the existing action for the current object.

Alternative: Redefine the entire action stream with **atom$act_replace_action**.

atom$act_append_action_from_reg
40 ($28)

This atom is no longer supported. This atom appended the data that is stored in the specified register to the existing action for the current object.

Alternative: Redefine the entire action stream with **atom$act_replace_action**.

atom$act_append_select_action
35 ($23)

This atom is no longer supported. This atom affected only actions that have a criterion of SELECTION, and appended the specified action to the existing action for the current object.

Alternative: Redefine the entire action stream with **atom$act_replace_action**.

atom$act_copy_action_to_reg
38 ($26)

Description

atom$act_copy_action_to_reg copies the current action for the current object to the raw data area of a specified register.

Registers A through D are available and can be set to hold any binary raw data. These registers exist for the life of the application and are not influenced by stream openings or closings. The purpose of these registers is to hold temporary data such as an atom stream.

Once you have stored data in a register, you can copy it using **atom$act_replace_action_from_reg** or **atom$act_change_action_from_reg**.

Syntax

```
atom$act_copy_action_to_reg [<register>]
```

`[<register>]`	Specifies an optional register. Values are B, C or D. The default is register A.

Return Value

None.

Example

The following example copies an action to a raw data register:

```
atom$man_start_object <ind_group>
atom$act_set_criterion <4>
atom$act_replace_action
   <
   atom$uni_start_stream
   .
   .  (atom stream 1 goes here...)
   .
   atom$uni_end_stream
   >
➪ atom$act_copy_action_to_reg <B>
```

The result of this example is that the current window's action, which is atom stream 1, is copied to register B where it is stored.

atom$act_do_action
1 ($01)

Description

atom$act_do_action invokes the atom stream related to the specified action ID or criterion based on the current context. For a list of action IDs, see **atom$act_set_criterion**.

If an action is not found, the inheritance byte of the action ID is used to determine if the scope of the search should be extended. The upper byte of all action IDs defines the scope of the inheritance for the specified ID. For more information, see **atom$act_set_inheritance**.

Syntax

```
atom$act_do_action <action_ID>
```

`<action_ID>` Specifies a 1-byte action ID value.

Return Value

None.

Example

The following example invokes the current object's atom stream:

```
atom$man_start_object <trigger>
atom$act_set_criterion <1>
atom$act_replace_action
  <
  atom$uni_start_stream
  .
  .  (atom stream 1 goes here...)
  .
  atom$uni_end_stream
  >
.
.
.
atom$act_do_action <1>
```


atom$act_get_db_record
47 ($2F)

Description

atom$act_get_db_record indicates the database record to retrieve. This atom command is used in conjunction with **atom$act_set_db_length**, **atom$act_set_db_offset**, **atom$act_get_db_value**, **atom$act_get_db_record**, and **atom$act_set_db_id**.

Syntax

```
atom$act_get_db_record <dword>
```

`<dword>` Specifies a database global ID.

Return Value

The data from the specified record.

Examples

The following examples perform database record extraction. For the examples, database record 23 = "This is test case 27."

```
  atom$act_set_db_offset <6>
  atom$act_set_db_length <15>
➪ atom$act_get_db_record <23>
```

The result of this example is that "is test case 27" is returned.

```
  atom$act_set_db_offset <18>
  atom$act_set_db_length <2>
➪ atom$act_get_db_record <23>
```

The result of this example is that "27" is returned.

atom$act_get_db_value
53 ($35)

Description

atom$act_get_db_value retrieves the value stored in a database record. A database record is identified by a global ID.

This atom command is used in conjunction with **atom$act_set_db_length**, **atom$act_set_db_offset**, **atom$act_set_db_record**, **atom$act_get_db_record**, and **atom$act_set_db_id**.

Syntax

```
atom$act_get_db_value <global_ID>
```

`<global_ID>` Specifies the database record global ID.

Return Value

Value stored in the database record global ID.

Examples

The following example sounds a beep if the value stored in a database record is non-zero:

```
atom$act_get_db_value <20-0-1234>
.
.
.
atom$if_last_return_true_then <1>
.
.
.
atom$act_sound_beep
atom$uni_sync_skip <1>
```


atom$act_prepend_action
33 ($21)

This atom is no longer supported. This atom inserted the specified data before the existing action for the current object

Alternative: Redefine the entire action stream with **atom$act_replace_action**.

atom$act_prepend_action_from_reg
41 ($29)

This atom is no longer supported. This atom inserted the data that is stored in the specified register before the existing action for the current object.

Alternative: Redefine the entire action stream with **atom$act_replace_action**.

atom$act_prepend_select_action
36 ($24)

This atom is no longer supported. This atom affected only actions that have a criterion of SELECTION, and inserted the specified data before the existing action for the current object

Alternative: Redefine the entire atom stream with **atom$act_replace_action**.

atom$act_replace_action
3 ($03)

Description

atom$act_replace_action replaces the existing action with the specified data for the current object. If an existing action is not found, the specified action data is still applied to the current object. If no replacement action data is specified, the existing action is removed from the object.

Syntax

```
atom$act_replace_action [<data>]
```

`[<data>]`	Specifies optional variable length binary data that normally consists of an action atom stream.

Return Value

None.

Example

The following example replaces the existing action for the current object:

```
  atom$man_start_object <ind_group>
  atom$act_set_criterion <2>
➪ atom$act_replace_action
    <
    atom$uni_start_stream
    .
    . (atom stream 1 goes here...)
    .
    atom$uni_end_stream
    >
    .
    .
    .
➪ atom$act_replace_action
    <
    atom$uni_start_stream
    .
    . (atom stream 2 goes here...)
    .
    atom$uni_end_stream
    >
```


atom$act_replace_action_from_reg
39 ($27)

Description

atom$act_replace_action_from_reg replaces the existing action for the current object with the data that is stored in the specified register. If an existing action is not found, the action data in the register is still applied to the current object.

The raw data areas of registers A, B, C and D are available and can be set to hold any binary raw data. These registers exist for the life of the application and are not influenced by stream openings or closings. The purpose of these registers is to hold temporary data such as entire atom streams.

If no register is specified, register A is the default. For information about how to set action data in a register, see **atom$act_set_action_in_reg**. For information about how to copy action data to a register, see **atom$act_copy_action_to_reg**.

Syntax

```
atom$act_replace_action_from_reg [<register>]
```

`[<register>]`	Specifies an optional register. Values are B, C or D. The default is register A.

Return Value

None.

Example

The following example replaces the current object's action using data stored in a register. For this example, register B = Atom Stream 2:

```
    atom$man_start_object <ind_group>
    atom$act_set_criterion <4>
    atom$act_replace_action
      <
      atom$uni_start_stream
      .
      . (atom stream 1 goes here...)
      .
      atom$uni_end_stream
      >
    .
    .
    .
➪   atom$act_replace_action_from_reg <B>
```


atom$act_replace_popup_menu_action
55 ($37)

Description

atom$act_replace_popup_menu_action affects only actions that have a criterion of POPUPMENU, and replaces the current object's existing action with a pop-up menu action. If an existing action is not found, the specified action is still applied to the current object.

This atom is often used as a shorthand atom for creating an action, since it eliminates the use of **atom$act_set_criterion** <**26**>.

Syntax

```
atom$act_replace_popup_menu_action [<data>]
```

`[<data>]`	Specifies optional variable length binary data that normally consists of an action atom stream.

Return Value

None.

Example

The following example defines a pop-up menu action that will occur when the current object is selected with a right mouse click:

```
atom$man_start_object <view, “”>
atom$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$act_replace_select_action
4 ($04)

Description

atom$act_replace_select_action affects only actions that have a criterion of SELECTION, and replaces the current object's existing action with the specified data. If an existing action is not found, the specified action is still applied to the current object.

This atom is often used as a shorthand atom for creating an action, since it eliminates the use of **atom$act_set_criterion <1>**.

Syntax

```
atom$act_replace_select_action [<data>]
```

`[<data>]`	Specifies optional variable length binary data that normally consists of an action atom stream.

Return Value

None.

Example

The following example defines an action that will occur when the current object is selected:

```
  atom$man_start_object <trigger>
➪ atom$act_replace_select_action
    <
    atom$uni_start_stream
    .
    .  (atom stream 1 goes here...)
    .
    atom$uni_end_stream
    >
```


atom$act_set_action_in_reg
43 ($2B)

Description

atom$act_set_action_in_reg stores specified action data in the raw data area of a register.

The raw data areas of registers A through D are available and can be set to hold any binary raw data. These registers exist for the life of the application and are not influenced by stream openings or closings. The purpose of these registers is to hold temporary data such as entire atom streams.

Once you have stored data in a register, you can copy it using **atom$act_replace_action_from_reg** or **atom$act_change_action_from_reg**.

Syntax

```
atom$act_set_action_in_reg <register, data>
```

`<register>`	Specifies a register. Values are A, B, C or D.
`<data>`	Specifies variable length data that normally consists of an action atom stream.

Return Value

None.

Example

The following example stores action data in a raw data area of a register. For this example, register B currently contains atom stream 3:

```
atom$act_set_action_in_reg <B,
  atom$uni_start_stream
  .
  .  (atom stream 4 goes here...)
  .
  atom$uni_end_stream
  >
```

The result of this example is that atom stream 3, which is currently stored the raw data area of register B, is replaced by atom stream 4. Atom stream 4 exists in the register until it is replaced or the application is terminated.

Note that the action data, which is the second argument of **atom$act_set_action_in_reg**, is defined by an atom stream that begins with **atom$uni_start_stream** and ends with **atom$uni_end_stream**.

atom$act_set_criterion
0 ($00)

Description

atom$act_set_criterion specifies the action ID for atom streams. Action IDs determine what causes an action to occur.

Common IDs include CLOSE, LOSE_FOCUS, GAIN_FOCUS, and SELECTION. Protocol-independent IDs have a value of 128 or greater, which let developers create their own action criteria.

Syntax

```
atom$act_set_criterion <action_ID>
```

`<action_ID>` Specifies an action ID. Values are:

`1 (01x)`	SELECTION — action occurs when an object is selected.
`2 (02x)`	CLOSE — action occurs when an object is closed.
`4 (04x)`	GAIN_FOCUS — action occurs when a window is brought to the front.
`5 (05x)`	LOSE_FOCUS — action occurs when another window is brought to the front of the current window.
`6 (06x)`	CANCEL — action occurs when ESC is pressed while the window is the top-most window.
`7 (07x)`	ENTER_FREE — action occurs when a free area is entered. This criterion applies to independent groups (windows) and affects the default behavior of switching between free and paid areas.
`8 (08x)`	ENTER_PAID — action occurs when a paid area is entered. This criterion applies to independent groups (windows)

	and affects the default behavior of switching between free and paid areas.
`9 (09x)`	CREATE — action occurs when a window is first drawn.
`10 (0Ax)`	SET_ONLINE — action occurs when a member goes online. This criterion applies to independent groups (windows).
`11 (0Bx)`	SET_OFFLINE — action occurs when a member goes offline. This criterion applies to independent groups (windows).
`12 (0Cx)`	RESTORE — action occurs when a window is restored.
`14 (0Ex)`	MINIMIZE — action occurs when a window is minimized.
`15 (0Fx)`	RESTORE_FROM_MAXIMIZE — action occurs when a window is restored to its original size after being maximized.
`16 (10x)`	RESTORE_FROM_MINIMIZE — action occurs when a window is restored to its original size after being minimized.
`17 (11x)`	TIMEOUT — action occurs when the host fails to respond. This criterion lets you override the default behavior for what happens when the host does not respond.
`18 (12x)`	SCREEN_NAME_CHANGED — action occurs when a screen name is changed. This criterion applies to independent groups (windows) and affects all windows open when the action occurs.
`19 (13x)`	MOVIE_OVER — action occurs when an **.avi** file has finished playing.

`20 (14x)`	DROP — action occurs when data is dropped into an object (field or view) and the mouse button is released.
`21 (15x)`	URL_DROP — action occurs when an URL is dropped into an object (once the mouse button is released).
`22 (16x)`	USER_DELETE — action occurs when a user presses DELETE or uses a mouse to delete something.
`23 (17x)`	TOGGLE_UP — action occurs when a user activates the up action of a toggle switch.
`24 (18x)`	ACTIVATED — action occurs when MDI_ACTIVATE is true.
`25 (19x)`	DEACTIVATED — action occurs when MDI_ACTIVATE is false.
`26 (1Ax)`	POPUPMENU — action occurs when a user invokes a pop-up menu.
`27 (1Bx)`	DESTROY — action occurs when a window is cleared from the screen.
`128-255 (80-FFx)`	ACTION_TOOL — reserved for protocol-independent criterion, e.g., action ID 128 in the MORG protocol is ACTION_VIEW_UPDATED. Action ID 128 in the WWW protocol is ACTION_URL_CHANGED.

Return Value

Action ID.

Example

The following example sets the action ID to 9 so that the action occurs when a window is first drawn:

```
  atom$man_start_object <ind_group>
➪ atom$act_set_criterion <9>
  atom$act_replace_action
    <
    atom$uni_start_stream
    .
    . (atom stream goes here...)
    .
    atom$uni_end_stream
    >
```


atom$act_set_db_id
48 ($30)

Description

atom$act_set_db_id specifies the database record to save the object data to.

Syntax

```
atom$act_set_db_id <dword>
```

`<dword>` Specifies a database global ID.

Return Value

Database global ID.

Example

The following example saves data to database record 20-0-1234:

```
atom$act_set_db_id <20-0-1234>
```

atom$act_set_db_length
46 ($2E)

Description

atom$act_set_db_length specifies the chunk size of data to extract from or store in a database record.

Syntax

```
atom$act_set_db_length <dword>
```

`<dword>`	Specifies the chunk size of data to extract from or store in a record.

Return Value

None.

Example

The following example extracts a 15-byte chunk of data from database record 23:

```
  atom$act_set_db_offset <6>
➪ atom$act_set_db_length <15>
  atom$act_get_db_record <23>
```


atom$act_set_db_offset
52 ($34)

Description

atom$act_set_db_offset sets an offset value from the start of the database record area. The default offset is 0, which points to the start of the record.

Syntax

```
atom$act_set_db_offset <dword>
```

`<dword>` Specifies an offset value.

Return Value

None.

Example

The following example offsets the start of the storage of data in a record by 2 bytes. In this example, record 20-0-1234 initially contains 00x 01x.

```
atom$act_set_db_id <20-0-1234>
atom$act_set_db_offset <2>
atom$act_set_db_record <02x 03x>
```

The result of this example is record 20-0-1234 contains 00x 01x 02x 03x.

atom$act_set_db_record

49 ($31)

Description

atom$act_set_db_record saves an object to a database record. The database record is identified by a global ID.

This atom command is used in conjunction with **atom$act_set_db_length**, **atom$act_set_db_offset**, **atom$act_get_db_record**, **atom$act_set_db_id**, and **atom$act_get_db_value**.

Syntax

```
atom$act_set_db_record <object_data>
```

`<object_data>`	Specifies the object data to store. The data is stored in the form of a sequence of bytes.

Return Value

Database record global ID.

Example

The following example saves the object data to a database record:

```
  atom$act_set_db_id <20-0-1234>
  atom$act_set_db_offset <0>
➪ atom$act_set_db_record <00x 01x 02x 03x>
```


atom$act_set_guest_flag
50 ($32)

Description

atom$act_set_guest_flag sets the state of the flag fGuestAccount. An argument must be specified to set the state of fGuestAccount to true.

atom$if_guest_then and **atom$if_owner_then** can be used to perform conditional operations on the state of the fGuestAccount flag.

Syntax

```
atom$act_set_guest_flag [<boolean>]
```

`[<boolean>]` Specifies the flag state. Values are:

`1` fGuestAccount is true.

`0` fGuestAccount is false. (Default)

Return Value

None.

Examples

The following examples set the state of the fGuestAccount flag:

```
atom$act_set_guest_flag <1>
```

The result of this example is that the fGuestAccount is set to true.

```
atom$act_set_guest_flag <0>
```

The result of this example is that the fGuestAccount is set to false.

```
atom$act_set_guest_flag
```

The result of this example is that the fGuestAccount is set to false.

atom$act_set_inheritance
2 ($02)

Description

atom$act_set_inheritance determines the scope of a search for an action or variable for the current object. The search begins on the current object. If an action or variable is not found, the search continues as far as the inheritance flag allows. List box items usually have an inheritance flag ID set to inherit all actions from their parent. This means only one action is defined for all child items.

To inherit actions or variables directly from an object, use 128 (DIRECT_FROM flag) in conjunction with another inheritance flag ID. For example, to inherit actions or variables from the root, use 128 (DIRECT_FROM flag) plus 5 (ROOT). This causes the action protocol to ignore all other objects in the tree except for the one specified.

Syntax

```
atom$act_set_inheritance <flag_id>
```

`<flag_id>` Specifies a 1-byte flag ID to determine where the action is inherited from. Values are:

`0`	NONE — searches the current object.
`1`	INDEPENDENTS_PARENT — searches the current independent group (window object), then the parent.
`2`	PARENT — searches the current object, then the parent.
`3`	INDEPENDENT — searches the current object, then the independent group (window object).
`4`	INDEPENDENT_AND_ABOVE — searches the current independent group (window object), then the next parent, and the next parent, all the way up to the root.

`5`	ROOT — searches the current object, then the root.
`6`	OBJECT_ID — searches the current object, then the global ID.
`7`	RELATIVE_ID — searches the current object, then the relative ID.
`128`	DIRECT_FROM — used in conjunction with another inheritance flag and searches the object specified by that inheritance flag.

Return Value

The flag ID.

Examples

The following examples show how actions are inherited:

```
atom$act_set_inheritance <5>
```

The result of this example is that a search for an action begins on the current object. If a match is not found, the parent object is searched, followed by the window object, followed by the root.

```
atom$act_set_inheritance <134>
```

The result of this example is that the action is inherited directly from the global ID of the current object. This is derived by adding 128 (the value of the DIRECT_FROM flag) and 6 (the value of the OBJECT_ID flag).

atom$act_set_newuser_flag
51 ($33)

Description

atom$act_set_newuser_flag sets the state of the fNewUser flag. An argument must be specified to set the state of fNewUser to true.

atom$if_newuser_then can be used to perform conditional operations on the state of the fNewUser flag.

Syntax

```
atom$act_set_newuser_flag [<boolean>]
```

`[<boolean>]` Specifies the state of the flag. Values are:

`1` fNewUser is true.

`0` fNewUser is false. (Default)

Return Value

None.

Examples

The following examples set the state of the fNewUser flag:

```
atom$act_set_newuser_flag <1>
```

The result of this example is that fNewUser is set to true.

```
atom$act_set_newuser_flag <0>
```

The result of this example is that fNewUser is set to false.

```
atom$act_set_newuser_flag
```

The result of this example is that fNewUser is set to false.

atom$act_sound_beep
5 ($05)

Description

atom$act_sound_beep causes the internal speaker of the client computer to make a beep sound. The client computer must be equipped with a speaker.

Syntax

```
atom$act_sound_beep [<sound>]
```

`[<sound>]` Specifies an optional 1-byte sound index, which is one of the default Windows sounds, to be played if a sound driver is enabled. If no argument is specified, a standard beep sound is produced using the computer speaker. Values are:

- `40x` Plays the sound identified by the SystemAsterisk entry in the sounds section of **win.ini**.
- `30x` Plays the sound identified by the SystemExclamation entry in the sounds section of **win.ini**.
- `20x` Plays the sound identified by the SystemQuestion entry in the sounds section of **win.ini**.
- `10x` Plays the sound identified by the SystemHand entry in the sounds section of **win.ini**.
- `00x` Plays the sound identified by the SystemDefault entry in the sounds section of **win.ini**.

Return Value

None.

Examples

The following examples make a beep sound on the client computer:

```
atom$act_sound_beep
```

The result of this example is that the client computer makes a beep sound from its internal speaker.

```
atom$act_sound_beep <30x>
```

The result of this example is that the client computer plays the sound identified by the SystemExclamation entry in the sounds section of the Windows **win.ini** file.

