Manual: FDO91 Manual
Chapter 18: File Manager Protocol defines file manager protocol atoms and provides information for handling file management.
Last updated: February 1995

C H A P T E R   1 8

File Manager Protocol

The File Manager protocol (protocol ID 8) consists of atoms that manage online service files on the client computer. The functions provided in the File Manager Tool can be accessed through inter-tool messaging, as well as with atom commands. The File Manager Tool's primary responsibilities include the following:

- Support for the commands on the standard File menu and extendibility for future non-text views.
- File type registration and notification for tools.
- Atom- and message-based interface to the Windows Common Dialog Library's Open, Print, and Save As dialog boxes.
- Low-level session log and chat log support, including basic word wrap.
- Windows drag-and-drop functionality.
- The ability to read and modify any .INI file.
- Basic file operations such as creation, deletion, and renaming.

Context of the File Manager Tool

The online service application supports multiple, concurrent streams of data, which requires the File Manager Tool to be available for simultaneous operations. To support simulataneous operations, the File Manager Tool maintains multiple contexts. A context is the current state of the File Manager Tool and consists mainly of approximately twenty context registers, each of which contains a single value or string representing data to be passed to an operation, the result of an operation, or some other piece of information. The **atom$fm_start** atom is used to create a File Manager Tool context for a stream, while the **atom$fm_end** atom should be invoked before the stream terminates in order to dispose of the context. A similar set of messages exists for tools that need to make use of File Manager Tool services.

Once a context has been created, other command atoms and messages may be used to manipulate the context registers. Since all File Manager Tool atoms have been designed to take no more than a single argument, setting the value of a register directly is a two-step process. First, use the **atom$fm_item_type** atom to select the register to be used for subsequent operations. Then, supply the new value of the register via the **atom$fm_item_set** atom. To query the value of a register, imbed the **atom$fm_item_get** atom in your stream. The value of the selected register is returned as the last atom value of the stream. For information on how to use the value of the selected register, see Chapter 5, "Universal Protocol."

File Types

In a tool-based environment such as that used by the online service application, it is impossible to anticipate all of the document types that might be introduced in the future by new tools. In fact, even attempting to do so would go against the grain of the tool approach, where extendibility must be ensured. With that in mind, The File Manager Tool was designed to make it possible to seamlessly add new document types without changing any existing code or forms. When a tool that supports a document type is present, File menu operations automatically handle that type of document, passing requests to the appropriate tool as needed. Similarly, drag-and-drop operations involving the new document type are forwarded to the proper tool.

The bulk of this is accomplished through file type registration. During its initialization, a tool that manages a type of document must register its document type by sending a message to the File Manager Tool that includes the 3-character file type extension shared by all documents of the given type, tag text describing the document type, and a set of flags that determine the behavior of such documents in generic terms (i.e., whether such documents can be loaded, created, and/or saved from the File menu). The File Manager Tool returns a file type ID to the tool to be used for all future transactions involving that document type.

Tools that register file types can also include handlers for any notification messages broadcast by the File Manager Tool. These include requests to open a specific file, determine which file type(s) to use for a Save As, and save the current file. All File Manager Tool messages are sent to tools that that have registered for the messages. A file type ID is passed with all broadcast messages to allow a tool to ignore messages that do not pertain to one of its document types. Broadcasting to all registered tools makes it possible for tools to share file types and is also necessary for operations where the file type is ambiguous or simply not known. For example, many tools listen for the MSG_NOTIFY_INQUIRE_SAVE_AS message. The tool that determines that one of its documents is displayed in the current window responds to the message; all other tools ignore it, letting it pass to the next tool in the dispatch tree.

Related file types may be grouped for the purpose of reducing the number of items displayed in the pop-up menu on file selection dialog boxes. For example, "*.PCX" and "*.LBM" files might be grouped under the heading "Art Files." When a file group is registered, a file type ID is returned that can be used to reference the group. For more information on using file type grouping, see the atom descriptions later in this chapter and message descriptions in the America Online *Windows Client Manual*.

Dialog Boxes

The File Manager Tool employs the standard Windows 3.1 Open, Save As, and Print dialog boxes. In addition, a locally-defined dialog box is used to select the type for New operations. Prior to using the atoms that invoke the dialog boxes, the File Manager tool usually performs a small amount of initialization. The initial directory can be specified, and, for Save As, a default filename may be needed. By default, the only file type selectable from the pop-up menu is "All Files (*.*)." The File Manager Tool provides several atoms that add registered file types and file groups to the context file list, which is the list of file types displayed in the next dialog box. In the Open dialog box, the file type list is merely a convenience to the user; however, for Save As, the file type highlighted when the transaction is complete determines the type of document created, as well as the default extension used if the user did not specify one.

After the user completes a dialog box transaction, several context registers may be set. The most important register – and the first to check – is fmi_error_code, which summarizes the overall result of the operation. A result of 0 indicates that the transaction was successful. A non-zero value indicates that the user canceled the operation or that a disk error occurred. The **atom$fm_handle_error** atom can be used to provide default error handling; it does nothing if no error occurred, but optionally displays an error message, broadcasts the error to all registered tools, and/or terminates the current stream if something went awry. Other registers updated include the fmi_filename, fmi_path, and fmi_filespec. The File Manager Tool also deduces a file type from the extension of the file selected (or from the highlighted file type if no extension was specified) and updates the file type ID, extension, and description registers.

Several options are available once the user selects a file. The **atom$fm_broadcast** atom can be used to announce the event to any registered tools. The argument to that atom is the type of broadcast message to be generated. Alternatively, the value of any register(s) can be used as the argument(s) of subsequent atoms in any protocol. For example, the filespec could be displayed in a text field on a screen or saved in an action variable. Beyond that, the File Manager Tool provides several atoms that carry out common file operations on the current file.

Initialization Files

Windows applications use initialization (*.INI) files to keep track of user preferences and other settings that must be maintained between application sessions. Since the online service application employs a much more sophisticated database system, the .INI file is not often used. However, .INI files are still useful for providing hooks for options and functionality that need to be made available to members on an emergency basis only. An example of this includes special debugging flags that cause the online service application to create detailed crash reports.

The File Manager Tool provides atoms that allow any .INI file to be read or modified. The default .INI file is **AOL.INI**, and a register is provided to allow a stream to switch to a different .INI file, such as **WIN.INI**, when necessary. .INI file settings are identified by group and key; to access a particular setting, you must set the fmi_ini_group and fmi_ini_key registers. The online service application supports the following data types in .INI files: string (FM_INI_STRING), integer (FM_INI_INT), and boolean (FM_INI_BOOL). The fmi_ini_data_type context register is used to specify the data type for an **atom$fm_write_data** or **atom$fm_read_data** atom.

File Operations

A number of advanced file and disk operations were added to the tool as atoms and messages to provide additional functionality. These include creating, opening, and closing files, deleting files, renaming files, checking a disk volume for space, and adding data to and reading data from files at any offset. This functionality, coupled with the power of the action language, allows fairly complex file management tasks to be carried out through atom streams.

A stream may have only one file open at once. The fmi_handle context register contains the handle for the currently open file. When no file is open, the value of that register is guaranteed to be 0. Open files are not automatically closed when a context is destroyed or the stream ends; they must be explicitly closed. File operations that require a filespec, such as open and create, use the one contained in the fmi_filespec context register.

Files can be opened in binary mode or text mode. In binary mode, the argument to an **atom$fm_append_data** atom is appended verbatim to the file. In text mode, some simple interpretation of the data can be done. Word wrap can be applied based on the line width and indentation registers, escape sequences are expanded to the current date and time or the text of a register, and chat logs can be properly formatted. The fmi_text_flags context register dictates which translations are used.

File Manager Protocol Atoms

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

atom$fm_add_file_type

7 ($07)

Description

atom$fm_add_file_type adds a registered file type to the context file type list. This atom is used with **atom$fm_dialog_get_file** and **atom$fm_dialog_put_file**. For more information about registering file types, see **atom$fm_register**.

If no argument is specified, the value of the fmi_file_type_id context register is used.

Syntax

```
atom$fm_add_file_type [<file_type_id>]
```

`[<file_type_id>]`	Specifies a registered file type.

Return Value

Unchanged.

Example

The following is an example of how to add a registered file type to the context file type list:

```
  atom$fm_start
  atom$fm_item_type <fmi_file_type_ext>
  atom$fm_item_set <JPG>
  atom$fm_item_type <fmi_file_type_desc>
  atom$fm_item_set <JPG Files>
  atom$fm_register <FMR_FILE_TYPE>
➪ atom$fm_add_file_type
  atom$fm_dialog_put_file <Save As>
```

```
.
.
.
atom$fm_end
```

The example is explained as follows:

`atom$fm_start`

Creates a File Manager Tool context.

`atom$fm_item_type <fmi_file_type_ext>`

Specifies the fmi_file_type_ext context register.

`atom$fm_item_set <JPG>`

Sets the value of the current context register to JPG.

`atom$fm_item_type <fmi_file_type_desc>`

Specifies the fmi_file_type_desc context register.

`atom$fm_item_set <JPG Files>`

Sets the value of the current context register to JPG Files.

`atom$fm_register <FMR_FILE_TYPE>`

Registers a new file type and creates a file type ID.

➪ `atom$fm_add_file_type`

Adds the registered file type ID stored in the fmi_file_type_id context register to the context file type list.

`atom$fm_dialog_put_file <Save As>`

Displays a standard Windows file dialog box and retrieves a filename and full path of a file to be created.

`atom$fm_end`

Disposes of the File Manager Tool context.

atom$fm_add_file_type_mask
9 ($09)

Description

atom$fm_add_file_type_mask adds all registered file types matching a mask criteria to the context file type list. This atom is used with **atom$fm_dialog_get_file** and **atom$fm_dialog_put_file**. For more information about registering file types, see **atom$fm_register**.

The mask criteria requires that the type flag value of each registered file type be exclusive-*or*ed with the value of the fmi_xor_mask context register and *and*ed with the value of the fmi_and_mask context register. If the result is non-zero, the file type is added to the context file list. To add all registered file types to the file list, set the value of the fmi_xor_mask register to <FFx FFx> and the value of the fmi_and_mask register to <0>. For more information about setting context registers, see **atom$fm_item_type** and **atom$fm_item_set**.

Syntax

```
atom$fm_add_file_type_mask
```

Return Value

Unchanged.

Example

The following is an example of how to add all registered file types matching a mask criteria to the context file type list:

```
  atom$fm_start
  atom$fm_item_type <fmi_xor_mask>
  atom$fm_item_set <FFx FFx>
  atom$fm_item_type <fmi_and_mask>
  atom$fm_item_set <0>
➪ atom$fm_add_file_type_mask
  atom$fm_dialog_get_file <Open a file>
```

```
.
.
.
atom$fm_end
```

The example is explained as follows:

`atom$fm_start`

Creates a File Manager Tool context.

`atom$fm_item_type <fmi_xor_mask>`

Specifies the fmi_xor_mask context register.

`atom$fm_item_set <FFx FFx>`

Sets the value of the current context register to FFx FFx.

`atom$fm_item_type <fmi_and_mask>`

Specifies the fmi_and_mask context register.

`atom$fm_item_set <0>`

Sets the value of the current context register to 0.

➪ `atom$fm_add_file_type_mask`

Adds all registered file types matching the mask criteria to the context file type list.

`atom$fm_dialog_get_file <Open a file>`

Displays a standard Windows file dialog box and retrieves a filename and full path of an existing file.

`atom$fm_end`

Disposes of the File Manager Tool context.

atom$fm_add_type_to_group
14 ($0E)

Description

atom$fm_add_type_to_group adds a registered file type to the current group. The fmi_file_group_id context register contains the ID of the current group.

If no argument is specified, the value of the fmi_file_type_id context register is used.

Syntax

```
atom$fm_add_type_to_group [<file_type_id>]
```

`[<file_type_id>]`	Specifies a registered file type or group identifier.

Return Value

Unchanged.

Example

The following is an example of how to register two file types and a group and add the two registered file types to the group:

```
atom$fm_start
atom$fm_item_type <fmi_file_type_ext>
atom$fm_item_set <GIF>
atom$fm_item_type <fmi_file_type_desc>
atom$fm_item_set <GIF Files>
atom$fm_register <FMR_FILE_TYPE>
atom$fm_item_get <fmi_file_type_id>
atom$var_number_set_from_atom <NUMA>
atom$fm_item_type <fmi_file_type_ext>
atom$fm_item_set <JPG>
atom$fm_item_type <fmi_file_type_desc>
atom$fm_item_set <JPG Files>
atom$fm_register <FMR_FILE_TYPE>
atom$fm_item_get <fmi_file_type_id>
atom$var_number_set_from_atom <NUMB>
atom$fm_item_type <fmi_file_type_desc>
atom$fm_item_set <Graphics Files>
atom$fm_register <FMR_FILE_GROUP>
atom$var_number_get <NUMA>
```

```
➪ atom$uni_use_last_atom_value <prot$file>
    <atom$fm_add_type_to_group>
  atom$var_number_get <NUMB>
➪ atom$uni_use_last_atom_value <prot$file>
    <atom$fm_add_type_to_group>
  .
  .
  .
  atom$fm_end
```

The example is explained as follows:

`atom$fm_start`

Creates a File Manager Tool context.

`atom$fm_item_type <fmi_file_type_ext>`

Specifies the fmi_file_type_ext context register.

`atom$fm_item_set <GIF>`

Sets the value of the current context register to GIF.

`atom$fm_item_type <fmi_file_type_desc>`

Specifies the fmi_file_type_desc context register.

`atom$fm_item_set <GIF Files>`

Sets the value of the current context register to GIF Files.

`atom$fm_register <FMR_FILE_TYPE>`

Registers a new file type and creates a file type ID.

`atom$fm_item_get <fmi_file_type_id>`

Retrieves the value of the fmi_file_type_id context register.

`atom$var_number_set_from_atom <NUMA>`

Sets the value of the NUMA variable register to the file type ID assigned to the GIF file type.

`atom$fm_item_type <fmi_file_type_ext>`

Specifies the fmi_file_type_ext context register.

```
atom$fm_item_set <JPG>
```

Sets the value of the current context register to JPG.

```
atom$fm_item_type <fmi_file_type_desc>
```

Specifies the fmi_file_type_desc context register.

```
atom$fm_item_set <JPG Files>
```

Sets the value of the current context register to JPG Files.

```
atom$fm_register <FMR_FILE_TYPE>
```

Registers a new file type and creates a file type ID.

```
atom$fm_item_get <fmi_file_type_id>
```

Retrieves the value of the fmi_file_type_id context register.

```
atom$var_number_set_from_atom <NUMB>
```

Sets the value of the NUMB variable register to the file type ID assigned to the JPG file type.

```
atom$fm_item_type <fmi_file_type_desc>
```

Specifies the fmi_file_type_desc context register.

```
atom$fm_item_set <Graphics Files>
```

Sets the value of the current context register to Graphics Files.

```
atom$fm_register <FMR_FILE_GROUP>
```

Registers a new file group and creates a file group ID.

```
atom$var_number_get <NUMA>
```

Gets the value of the NUMA variable register.

```
➪ atom$uni_use_last_atom_value <prot$file>
    <atom$fm_add_type_to_group>
```

Adds the registered file type retrieved with the previous atom to the current group.

```
atom$var_number_get <NUMB>
```

Gets the value of the NUMB variable register.

```
➪ atom$uni_use_last_atom_value <prot$file>
    <atom$fm_add_type_to_group>
```

Adds the registered file type retrieved with the previous atom to the current group.

```
atom$fm_end
```

Disposes of the File Manager Tool context.

atom$fm_append_data
27 ($1B)

Description

atom$fm_append_data appends data at the current position to the file in context. The fmi_filespec context register contains the name of the file in context. The file must be open.

If the value of the fmi_text_mode context register is 1, indicating a text file, the fmi_text_flags context register is honored using an interim data buffer to format the data prior to writing it to the disk. The formatting process uses the values of the fmi_text_width and fmi_text_indent context registers. For more information about the context registers, see **atom$fm_item_type**.

atom$fm_append_data sets the value of the fmi_error_code context register to reflect the result of the operation. For more information about the possible values for this register, see **atom$fm_handle_error**.

Syntax

```
atom$fm_append_data <data>
```

`<data>`	Specifies data.

Return Value

Unchanged.

Example

The following is an example of how to append data to the currently open file at the current position:

```
atom$fm_start
atom$fm_item_type <fmi_xor_mask>
atom$fm_item_set <FFx FFx>
atom$fm_item_type <fmi_and_mask>
atom$fm_item_set <0>
atom$fm_add_file_type_mask
atom$fm_dialog_put_file <Open a file>
atom$fm_create_file
atom$fm_open_file <4>
```

```
➪ atom$fm_append_data <Hello>
```

The example is explained as follows:

`atom$fm_start`

Creates a File Manager Tool context.

`atom$fm_item_type <fmi_xor_mask>`

Specifies the fmi_xor_mask context register.

`atom$fm_item_set <FFx FFx>`

Sets the value of the current context register to FFx FFx.

`atom$fm_item_type <fmi_and_mask>`

Specifies the fmi_and_mask context register.

`atom$fm_item_set <0>`

Sets the value of the current context register to 0.

`atom$fm_add_file_type_mask`

Adds all registered file types matching the mask criteria to the context file type list.

`atom$fm_dialog_put_file <Open a file>`

Displays a standard Windows file dialog box and retrieves a filename and full path of a file to be created.

`atom$fm_create_file`

Creates a file.

`atom$fm_open_file <4>`

Opens an existing file in text mode for write operations and positions the file pointer to the beginning of the file.

➪ `atom$fm_append_data <Hello>`

Appends data at the current position to the file in context.

atom$fm_broadcast
5 ($05)

Description

atom$fm_broadcast broadcasts an event to the tools registered for that event. If the tool returns a value, the value is stored in the fmi_broadcast_return_value context register. For more information about registering tools for an event, see **???**.

Syntax

```
atom$fm_broadcast <event>
```

`<event>`	Specifies an event. Values are as follows: fmb_general (0) — general broadcast. fmb_open (1) — file open request. fmb_save_as (3) — save current document request. fmb_save (4) — save current document with default filename request. fmb_inquire_new (5) — invitation to add file types to the file list for use in a file type selection dialog box. fmb_new (6) — create a new document request. fmb_error (7) — error notification. fmb_print (9) — print current document request. fmb_abort_print (10) — current print job aborted notification.

Return Value

Unchanged.

Example

The following is an example of how to broadcast an event to the tools registered for that event:

```
  atom$fm_start
  atom$fm_item_type <fmi_xor_mask>
  atom$fm_item_set <FFx FFx>
  atom$fm_item_type <fmi_and_mask>
  atom$fm_item_set <0>
  atom$fm_add_file_type_mask
  atom$fm_dialog_get_file <Open a file>
  atom$fm_handle_error <fhf_display_msg+terminate>
➪ atom$fm_broadcast <fmb_open>
  atom$fm_end
```

The example is explained as follows:

`atom$fm_start`

Creates a File Manager Tool context.

`atom$fm_item_type <fmi_xor_mask>`

Specifies the fmi_xor_mask context register.

`atom$fm_item_set <FFx FFx>`

Sets the value of the current context register to FFx FFx.

`atom$fm_item_type <fmi_and_mask>`

Specifies the fmi_and_mask context register.

`atom$fm_item_set <0>`

Sets the value of the current context register to 0.

`atom$fm_add_file_type_mask`

Adds all registered file types matching the mask criteria to the context file type list.

`atom$fm_dialog_get_file <Open a file>`

Displays a standard Windows file dialog box and retrieves a filename and full path of an existing file.

`atom$fm_handle_error <fhf_display_msg+terminate>`

Provides generic error handling, using the fhf_display_msg+terminate flag.

➪ `atom$fm_broadcast <fmb_open>`

Broadcasts the fmb_open event type to all tools registered for that event.

`atom$fm_end`

Disposes of the File Manager Tool context.

atom$fm_close_file
26 ($1A)

Description

atom$fm_close_file closes a file. The fmi_handle context register contains the file handle associated with the file to be closed.

atom$fm_close_file sets the value of the fmi_error_code context register to reflect the result of the operation. For more information about the possible values for this register, see **atom$fm_handle_error**.

Syntax

```
atom$fm_close_file
```

Return Value

Unchanged.

Example

The following is an example of how to close a file:

```
  atom$fm_start
  .
  .
  .
  atom$fm_open_file <fmo_read>
  .
  .
  .
➪ atom$fm_close_file
```

The example is explained as follows:

```
atom$fm_start
```

Creates a File Manager Tool context.

```
atom$fm_open_file <fmo_read>
```

Opens an existing file in fmo_read mode and positions the file pointer to the beginning of the file.

➪ `atom$fm_close_file`

Closes the file.

atom$fm_create_file
21 ($15)

Description

atom$fm_create_file creates a file. The fmi_filespec context register contains the name of the file to be created. If the file exists, the contents are discarded. The value of the fmi_file_size context register determines the initial size of the new file.

atom$fm_create_file sets the value of the fmi_error_code context register to reflect the result of the operation. For more information about the possible values for this register, see **atom$fm_handle_error**.

Syntax

```
atom$fm_create_file
```

Return Value

Unchanged.

Example

The following is an example of how to create a file:

```
  atom$fm_start
  atom$fm_item_type <fmi_filespec>
  atom$fm_item_set <c:\test.txt>
  atom$fm_item_type <fmi_file_size>
  atom$fm_item_set <40x 0x>
➪ atom$fm_create_file
```

The example is explained as follows:

```
atom$fm_start
```

Creates a File Manager Tool context.

```
atom$fm_item_type <fmi_filespec>
```

Specifies the fmi_filespec context register.

```
atom$fm_item_set <c:\test.txt>
```

Sets the value of the current context register to c:\test.txt.

```
atom$fm_item_type <fmi_file_size>
```

Specifies the fmi_file_size context register.

```
atom$fm_item_set <40x 0x>
```

Sets the value of the fmi_file_size context register to 16K.

➪
```
atom$fm_create_file
```

Creates a file.

atom$fm_delete_file
23 ($17)

Description

atom$fm_delete_file deletes a file. The fmi_filespec context register contains the name of the file to be deleted. The file must be closed before it is deleted.

atom$fm_delete_file sets the value of the fmi_error_code context register to reflect the result of the operation. For more information about the possible values for this register, see **atom$fm_handle_error**.

Syntax

```
atom$fm_delete_file
```

Return Value

Unchanged.

Example

The following is an example of how to delete a file:

```
  atom$fm_start
  .
  .
  .
  atom$fm_open_file <fmo_read>
  .
  .
  .
  atom$fm_close_file
➪ atom$fm_delete_file
```

The example is explained as follows:

```
atom$fm_start
```

Creates a File Manager Tool context.

```
atom$fm_open_file <fmo_read>
```

Opens an existing file in fmo_read mode and positions the file pointer to the beginning of the file.

`atom$fm_close_file`

Closes the file.

➪ `atom$fm_delete_file`

Deletes the file.

atom$fm_delete_file_type
8 ($08)

Description

atom$fm_delete_file_type deletes a registered file type from the context file type list. For more information about registering file types, see **atom$fm_register**.

If no argument is specified, the value of the fmi_file_type_id context register is used.

Syntax

```
atom$fm_delete_file_type [<file_type_id>]
```

`[<file_type_id>]`	Specifies a registered file type.

Return Value

Unchanged.

Example

The following is an example of how to delete a registered file type from the context file type list:

```
atom$fm_start
atom$fm_item_type <fmi_file_type_ext>
atom$fm_item_set <ART>
atom$fm_item_type <fmi_file_type_desc>
atom$fm_item_set <ART Files>
atom$fm_register <FMR_FILE_TYPE>
atom$fm_item_get <fmi_file_type_id>
atom$var_number_set_from_atom <NUMA>
.
.
.
atom$fm_add_file_type
atom$var_number_get <NUMA>
➪ atom$uni_use_last_atom_value <prot$file>
  <atom$fm_delete_file_type>
```

The example is explained as follows:

`atom$fm_start`

Creates a File Manager Tool context.

`atom$fm_item_type <fmi_file_type_ext>`

Specifies the fmi_file_type_ext context register.

`atom$fm_item_set <ART>`

Sets the value of the current context register to ART.

`atom$fm_item_type <fmi_file_type_desc>`

Specifies the fmi_file_type_desc context register.

`atom$fm_item_set <ART Files>`

Sets the value of the current context register to ART Files.

`atom$fm_register <FMR_FILE_TYPE>`

Registers a new file type and creates a file type ID.

`atom$fm_item_get <fmi_file_type_id>`

Retrieves the value of the fmi_file_type_id context register.

`atom$var_number_set_from_atom <NUMA>`

Sets the value of the NUMA variable register to the file type ID assigned to the ART file type.

`atom$fm_add_file_type`

Adds the registered file type ID stored in the fmi_file_type_id context register to the context file type list.

`atom$var_number_get <NUMA>`

Gets the value of the NUMA variable register.

```
➪ atom$uni_use_last_atom_value <prot$file>
    <atom$fm_delete_file_type>
```

Deletes the registered file type retrieved with the previous atom from the context file type list.

atom$fm_delete_file_type_mask
10 ($0A)

Description

atom$fm_delete_file_type_mask deletes all registered file types matching a mask criteria from the context file type list. This atom is used with **atom$fm_dialog_get_file** and **atom$fm_dialog_put_file**. For more information about registering file types, see **atom$fm_register**.

The mask criteria requires that the type flag value of each registered file type be exclusive-*or*ed with the value of the fmi_xor_mask context register and *and*ed with the value of the fmi_and_mask context register. If the result is non-zero, the file type is deleted from the context file list. To delete all registered file types from the file list, set the value of the fmi_xor_mask register to <FFx FFx> and the value of the fmi_and_mask register to <0>. For more information about setting context registers, see **atom$fm_item_type** and **atom$fm_item_set**.

Syntax

```
atom$fm_delete_file_type_mask
```

Return Value

Unchanged.

Example

The following is an example of how to delete all registered file types matching a mask criteria from the context file type list:

```
  atom$fm_start
  atom$fm_item_type <fmi_xor_mask>
  atom$fm_item_set <FFx FFx>
  atom$fm_item_type <fmi_and_mask>
  atom$fm_item_set <0>
➪ atom$fm_delete_file_type_mask
```

The example is explained as follows:

```
atom$fm_start
```

Creates a File Manager Tool context.

`atom$fm_item_type <fmi_xor_mask>`

Specifies the fmi_xor_mask context register.

`atom$fm_item_set <FFx FFx>`

Sets the value of the current context register to FFx FFx.

`atom$fm_item_type <fmi_and_mask>`

Specifies the fmi_and_mask context register.

`atom$fm_item_set <0>`

Sets the value of the current context register to 0.

➪ `atom$fm_delete_file_type_mask`

Deletes all registered file types matching the mask criteria from the context file type list.

atom$fm_delete_type_from_group
15 ($0F)

Description

atom$fm_delete_type_to_group deletes a registered file type from the current group. The fmi_file_group_id context register contains the ID of the current group.

If no argument is specified, the value of the fmi_file_type_id context register is used.

Syntax

```
atom$fm_delete_type_from_group [<file_type_id>]
```

`[<file_type_id>]` Specifies a registered file type or group identifier.

Return Value

Unchanged.

Example

The following is an example of how to delete a file type from a group:

```
atom$fm_start
.
.
.
atom$var_number_get <NUMA>
➪ atom$uni_use_last_atom_value <prot$file>
  <atom$fm_delete_type_from_group>
```

The example is explained as follows:

```
atom$fm_start
```

Creates a File Manager Tool context.

```
atom$var_number_get <NUMA>
```

Gets the value of the NUMA variable register, which is a previously stored file type ID.

```
➪ atom$uni_use_last_atom_value <prot$file>
    <atom$fm_delete_type_from_group>
```

Deletes the registered file type retrieved with the previous atom from the current group.

atom$fm_dialog_get_file
17 ($11)

Description

atom$fm_dialog_get_file displays a standard Windows file dialog box and retrieves a filename and full path of an existing file from a member.

If the filename is valid, the following context registers are updated: fmi_file_type_id, fmi_file_type_ext, fmi_file_type_flags, fmi_filename, fmi_path, and fmi_filespec.

Syntax

```
atom$fm_dialog_get_file <title>
```

`<title>`	Specifies a title for the dialog box.

Return Value

Unchanged.

Example

The following is an example of how to display a standard Windows file dialog box and retrieve a filename and full path name of an existing file from a member:

```
  atom$fm_start
  atom$fm_item_type <fmi_xor_mask>
  atom$fm_item_set <FFx FFx>
  atom$fm_item_type <fmi_and_mask>
  atom$fm_item_set <0>
  atom$fm_add_file_type_mask
➪ atom$fm_dialog_get_file <Open a file>
```

The example is explained as follows:

```
atom$fm_start
```

Creates a File Manager Tool context.

```
atom$fm_item_type <fmi_xor_mask>
```

Specifies the fmi_xor_mask context register.

```
atom$fm_item_set <FFx FFx>
```

Sets the value of the current context register to FFx FFx.

```
atom$fm_item_type <fmi_and_mask>
```

Specifies the fmi_and_mask context register.

```
atom$fm_item_set <0>
```

Sets the value of the current context register to 0.

```
atom$fm_add_file_type_mask
```

Adds all registered file types matching the mask criteria to the context file type list.

```
➪ atom$fm_dialog_get_file <Open a file>
```

Displays a standard Windows file dialog box and retrieves a filename and full path of an existing file.

atom$fm_dialog_print
20 ($14)

Description

atom$fm_dialog_print displays a dialog box for printing.

atom$fm_dialog_print sets the value of the fmi_error_code context register to reflect the result of the operation. For more information about the possible values for this register, see **atom$fm_handle_error**.

In addition, the following context registers are updated: fmi_print_copies and fmi_hdc.

Syntax

```
atom$fm_dialog_print <dialog_code>
```

`<dialog_code>`	Specifies a dialog box. Values are as follows: fmp_print (0) — The standard Windows print dialog box. fmp_setup (1) — The standard Windows print setup dialog box. fmp_printing (2) — The online service print job status dialog box. fmp_done (3) — The online service print job completed dialog box.

Return Value

Unchanged.

Example

The following is an example of how to display a dialog box for printing:

```
atom$fm_start
```

```
  .
  .
  .
  atom$fm_item_type <fmi_print_min_page>
  atom$fm_item_set <1>
  atom$fm_item_type <fmi_print_max_page>
  atom$fm_item_set <2>
  atom$fm_item_type <fmi_print_job_name>
  atom$fm_item_set <Document1>
➪ atom$fm_dialog_print <fmp_print>
  atom$fm_broadcast <fmb_print>
```

The example is explained as follows:

`atom$fm_start`

Creates a File Manager Tool context.

`atom$fm_item_type <fmi_print_min_page>`

Specifies the fmi_print_min_page context register.

`atom$fm_item_set <1>`

Sets the value of the current context register to 1.

`atom$fm_item_type <fmi_print_max_page>`

Specifies the fmi_print_max_page context register.

`atom$fm_item_set <2>`

Sets the value of the current context register to 2.

`atom$fm_item_type <fmi_print_job_name>`

Specifies the fmi_print_job_name context register.

`atom$fm_item_set <Document1>`

Sets the value of the current context register to Document1.

➪ `atom$fm_dialog_print <fmp_print>`

Displays the standard Windows print dialog box.

`atom$fm_broadcast <fmb_print>`

Broadcasts the fmb_print event type to all tools.

atom$fm_dialog_put_file
18 ($12)

Description

atom$fm_dialog_put_file displays a standard Windows file dialog box and retrieves a filename and full path of a file to be created from a member.

If the filename is valid, the following context registers are updated: fmi_file_type_id, fmi_file_type_ext, fmi_file_type_flags, fmi_filename, fmi_path, and fmi_filespec.

Syntax

```
atom$fm_dialog_put_file <title>
```

`<title>`	Specifies a title for the dialog box.

Return Value

Unchanged.

Example

The following is an example of how to display a standard Windows file dialog box and retrieve a filename and full path name of a file to be created from a member:

```
  atom$fm_start
  atom$fm_item_type <fmi_xor_mask>
  atom$fm_item_set <FFx FFx>
  atom$fm_item_type <fmi_and_mask>
  atom$fm_item_set <0>
  atom$fm_add_file_type_mask
➪ atom$fm_dialog_put_file <Open a file>
```

The example is explained as follows:

```
atom$fm_start
```

Creates a File Manager Tool context.

```
atom$fm_item_type <fmi_xor_mask>
```

Specifies the fmi_xor_mask context register.

```
atom$fm_item_set <FFx FFx>
```

Sets the value of the current context register to FFx FFx.

```
atom$fm_item_type <fmi_and_mask>
```

Specifies the fmi_and_mask context register.

```
atom$fm_item_set <0>
```

Sets the value of the current context register to 0.

```
atom$fm_add_file_type_mask
```

Adds all registered file types matching the mask criteria to the context file type list.

```
➪ atom$fm_dialog_put_file <Open a file>
```

Displays a standard Windows file dialog box and retrieves a filename and full path of a file to be created.

atom$fm_dump
31 ($1F)

Description

atom$fm_dump displays the values of the File Manager Tool context registers as debug messages.

This atom is designed for development use only.

Syntax

```
atom$fm_dump
```

Return Value

Unchanged.

Example

The following is an example of how to display the values of the File Manager Tool context registers as debug messages:

```
atom$fm_dump
```

atom$fm_end
1 ($01)

Description

atom$fm_end disposes of the File Manager Tool context for the current atom stream.

Syntax

```
atom$fm_end
```

Return Value

Unchanged.

Example

The following is an example of how to dispose of the File Manager Tool context for the current atom stream:

```
   atom$fm_start
   .
   .
   .
➪ atom$fm_end
```

The example is explained as follows:

```
atom$fm_start
```

Creates a File Manager Tool context.

```
➪ atom$fm_end
```

Disposes of the File Manager Tool context.

atom$fm_find_file_type
16 ($10)

Description

atom$fm_find_file_type searches registered file types for one matching an extension and sets the fmi_file_type_id context register to the ID of the file type found.

If no argument is specified, the value of the fmi_file_type_ext context register is used.

Syntax

```
atom$fm_find_file_type [<file_ext>]
```

`[<file_ext>]`	Specifies a file extension.

Return Value

Unchanged.

Example

The following is an example of how to set the fmi_file_type_id context register:

```
atom$fm_start
.
.
.
➪ atom$fm_find_file_type <GIF>
atom$fm_item_get <fmi_file_type_id>
atom$uni_use_last_atom_value <prot$file>
  <atom$fm_add_file_type>
```

The example is explained as follows:

```
atom$fm_start
```

Creates a File Manager Tool context.

```
➪ atom$fm_find_file_type <GIF>
```

Searches registered file types for one matching GIF and sets the fmi_file_type_id context register to the ID of the file type found.

```
  atom$fm_item_get <fmi_file_type_id>
```

Retrieves the value of the fmi_file_type_id context register.

```
  atom$uni_use_last_atom_value <prot$file>
    <atom$fm_add_file_type>
```

Adds the registered file type ID retrieved with the previous atom to the context file type list.

atom$fm_flush_file
39 ($27)

Description

atom$fm_flush_file takes any data remaining in the data buffer and appends it at the current position to the file in context. The fmi_filespec context register contains the name of the file in context. The file must be open. For more information about the data buffer, see **atom$fm_append_data**.

If the value of the fmi_text_mode context register is 1, indicating a text file, the fmi_text_flags context register is honored, formatting the data in the data buffer prior to writing it to the disk. The formatting process uses the values of the fmi_text_width and fmi_text_indent context registers. For more information about the context registers, see **atom$fm_item_type**.

atom$fm_flush_file sets the value of the fmi_error_code context register to reflect the result of the operation. For more information about the possible values for this register, see **atom$fm_handle_error**.

Syntax

```
atom$fm_flush_file
```

Return Value

Unchanged.

Example

The following is an example of how to take any data remaining in the data buffer and append it at the current position to the file in context:

```
  atom$fm_start
  .
  .
  .
  atom$fm_item_type <fmi_text_width>
  atom$fm_item_set <5>
  atom$fm_open_file <fmo_text_write>
  atom$fm_append_data <THEEND>
➪ atom$fm_flush_file
```

```
atom$fm_close_file
```

The example is explained as follows:

```
atom$fm_start
```

Creates a File Manager Tool context.

```
atom$fm_item_type <fmi_text_width>
```

Specifies the fmi_text_width context register.

```
atom$fm_item_set <5>
```

Sets the value of the current context register to 5.

```
atom$fm_open_file <fmo_text_write>
```

Opens an existing file in fmo_text_write mode and positions the file pointer to the beginning of the file.

```
atom$fm_append_data <THEEND>
```

Appends data at the current position to the file in context.

```
➪ atom$fm_flush_file
```

Takes any data remaining in the data buffer and appends it at the current position to the file in context.

```
atom$fm_close_file
```

Closes the file.

The result of this example is that the data THEEND is appended in the following format to the file in context:

THEEN
D

Due to the fact that the file was opened in fmo_text_write mode, **atom$fm_append_data** appends the data to the file in blocks equal in size to the value of the fmi_text_width register (5). Any data in an odd-sized block at the end is placed in the data buffer. **atom$fm_flush_file** writes the contents of the buffer to the file

atom$fm_get_first_file
32 ($20)

Description

atom$fm_get_first_file locates the first file in the current path that matches the specified file name mask.

Syntax

```
atom$fm_get_first_file <file_mask>
```

`<file_mask>`	Specifies a file name mask. Wildcard characters are allowed.

Return Value

The name of the file found or NULL if no file is found.

Example

The following is an example of how to locate the first file in the current path that matches the specified file name mask:

```
  atom$fm_start
  .
  .
  .
  atom$fm_item_type <fmi_path>
  atom$fm_item_set <c:\aol\download>
➪ atom$fm_get_first_file <texas?.*>
```

The example is explained as follows:

```
atom$fm_start
```

Creates a File Manager Tool context.

```
atom$fm_item_type <fmi_path>
```

Specifies the fmi_path context register.

```
atom$fm_item_set <c:\aol\download>
```

Sets the value of the current context register to c:\aol\download.

```
➪ atom$fm_get_first_file <texas?.*>
```

Locates the first file in the current path that matches the texas?.* mask.

atom$fm_get_next_file
33 ($21)

Description

atom$fm_get_next_file locates the next file in the current path that matches the file name mask specified in the previous **atom$fm_get_first_file** atom.

Syntax

```
atom$fm_get_next_file
```

Return Value

The name of the file found or NULL if no file is found.

Example

The following is an example of how to locate the next file in the current path that matches the file name mask specified in the previous **atom$fm_get_first_file** atom:

```
  atom$fm_start
  .
  .
  .
  atom$fm_item_type <fmi_path>
  atom$fm_item_set <c:\aol\download>
  atom$fm_get_first_file <texas?.*>
➪ atom$fm_get_next_file
```

The example is explained as follows:

```
atom$fm_start
```

Creates a File Manager Tool context.

```
atom$fm_item_type <fmi_path>
```

Specifies the fmi_path context register.

```
atom$fm_item_set <c:\aol\download>
```

Sets the value of the current context register to c:\aol\download.

```
atom$fm_get_first_file <texas?.*>
```

Locates the first file in the current path that matches the texas?.* mask.

```
➪ atom$fm_get_next_file
```

Locates the next file in the current path that matches the previously specified mask.

atom$fm_handle_error
6 ($06)

Description

atom$fm_handle_error provides generic error handling. If the value of the fmi_error_code context register is fme_ok, no action is performed.

The value of the fmi_error_code context register is set by a number of File Manager Tool atoms. Values are as follows:

fme_ok (0) — no error.

fme_cancel (1) — dialog canceled by member.

fme_internal (2) — internal error.

fme_invalid_filename (3) — current filename is invalid.

fme_insufficient_space (4) — insufficient disk space for specified file.

fme_file_not_found (5) — fmi_filespec context register does not contain a valid value.

fme_path_not_found (6) — fmi_path context register does not contain a valid value.

fme_file_exists (7) — file already exists

fme_read (8) — general read error.

fme_write (9) — general write error.

fme_access (10) — access or sharing violation.

fme_no_default_printer (11) — no default printer defined.

fme_no_printers (12) — no printers defined.

fme_no_select (13) — current document cannot be printed or saved.

	fme_skip (14) — the skip push button was selected in the dialog box.

Syntax

```
atom$fm_handle_error <flag>
```

`<flag>`	Specifies the type of error handling. Values are as follows:
	fhf_display_msg (1) — Displays an generic error message.
	fhf_terminate (2) — Terminates the atom stream.
	fhf_display_msg+terminate (3) — Displays an generic error message and terminates the atom stream.
	fhf_broadcast (4) — Broadcasts an event to all tools.
	fhf_display_msg+broadcast (5) — Displays an generic error message and broadcasts an event to all tools.
	fhf_terminate+broadcast (6) — Terminates the atom stream and broadcasts an event to all tools.
	fhf_display_msg+terminate+broadcast (7) — Displays an generic error message, terminates the atom stream and broadcasts an event to all tools.

Return Value

Unchanged.

Example

The following is an example of how to provide generic error handling:

```
atom$fm_open_file <fmo_read>
```

```
➪ atom$fm_handle_error <fhf_display_msg+broadcast>
```

The example is explained as follows:

```
atom$fm_open_file <fmo_read>
```

Opens an existing file in fmo_read mode and positions the file pointer to the beginning of the file.

```
➪ atom$fm_handle_error <fhf_display_msg+broadcast>
```

???.

atom$fm_ini_read_data
34 ($22)

Description

atom$fm_ini_read_data reads data from an .INI file. The INI file to read from is specified in the fmi_ini_file context register. Similarly, the group and the key are specified in the fmi_ini_group and fmi_ini_key context registers respectively.

Syntax

```
atom$fm_ini_read_data
```

Return Value

Depends on the data type selected; the return value is numeric for word and boolean data types, and is a string data type otherwise.

Example

The following is an example of how to read data from an .INI file:

```
  atom$fm_item_type <fmi_ini_file>
  atom$fm_item_set <aol.ini>
  atom$fm_item_type <fmi_ini_group>
  atom$fm_item_set <preferences>
  atom$fm_item_type <fmi_ini_key>
  atom$fm_item_set <mode>
  atom$fm_item_type <fmi_ini_data_type>
  atom$fm_item_set <FM_INI_BOOL>
➪ atom$fm_ini_read_data
  .
  .
  .
  atom$if_last_return_true_then <1>
  .
  .
  .
  atom$uni_sync_skip <1>
```

The example is explained as follows:

```
atom$fm_item_type <fmi_ini_file>
```

Specifies the fmi_ini_file context register.

```
atom$fm_item_set <aol.ini>
```

Sets the value of the current context register to aol.ini.

```
atom$fm_item_type <fmi_ini_group>
```

Specifies the fmi_ini_group context register.

```
atom$fm_item_set <Preferences>
```

Sets the value of the current context register to Preferences.

```
atom$fm_item_type <fmi_ini_key>
```

Specifies the fmi_ini_key context register.

```
atom$fm_item_set <mode>
```

Sets the value of the current context register to mode.

```
atom$fm_item_type <fmi_ini_data_type>
```

Specifies the fmi_ini_data_type context register.

```
atom$fm_item_set <FM_INI_BOOL>
```

Sets the value of the current context register to FM_INI_BOOL.

```
➪ atom$fm_ini_read_data
```

Reads data from the current .INI file.

```
atom$if_last_return_true_then <1>
```

If the last return value is true, **atom$if_last_return_true_then** evaluates to true, and the atoms that fall between it and **atom$uni_sync_skip** are executed.

```
atom$uni_sync_skip <1>
```

Signifies the end of the true atoms. If **atom$if_last_return_true_then** evaluates to true, the true atoms are executed and stream execution continues immediately following this atom.

atom$fm_ini_write_data
35 ($23)

Description

atom$fm_ini_write_data writes data to an .INI file. The INI file to write to is specified in the fmi_ini_file context register. Similarly, the group and the key are specified in the fmi_ini_group and fmi_ini_key context registers respectively.

Syntax

```
atom$fm_ini_write_data <data>
```

`<data>`	Specifies data.

Return Value

Unchanged.

Example

The following is an example of how to write data to an .INI file:

```
atom$fm_item_type <fmi_ini_file>
atom$fm_item_set <aol.ini>
atom$fm_item_type <fmi_ini_group>
atom$fm_item_set <preferences>
atom$fm_item_type <fmi_ini_key>
atom$fm_item_set <name>
atom$fm_item_type <fmi_ini_data_type>
atom$fm_item_set <FM_INI_STRING>
atom$fm_item_type <fmi_ini_string_length>
atom$fm_item_set <5>
atom$man_set_context_relative <1>
atom$de_start_extraction <0>
atom$de_get_data_pointer
atom$uni_save_result
atom$de_end_extraction
atom$man_end_context
atom$uni_get_result
```

```
➪ atom$uni_use_last_atom_string <prot$file>
    <atom$fm_ini_write_data>
```

The example is explained as follows:

`atom$fm_item_type <fmi_ini_file>`

Specifies the fmi_ini_file context register.

`atom$fm_item_set <aol.ini>`

Sets the value of the current context register to aol.ini.

`atom$fm_item_type <fmi_ini_group>`

Specifies the fmi_ini_group context register.

`atom$fm_item_set <preferences>`

Sets the value of the current context register to Preferences.

`atom$fm_item_type <fmi_ini_key>`

Specifies the fmi_ini_key context register.

`atom$fm_item_set <mode>`

Sets the value of the current context register to mode.

`atom$fm_item_type <fmi_ini_data_type>`

Specifies the fmi_ini_data_type context register.

`atom$fm_item_set <FM_INI_BOOL>`

Sets the value of the current context register to mode.

`atom$fm_ini_read_data`

Reads data from the current .INI file.

`atom$man_set_context_relative <1>`

Sets the context to the object with a relative ID of 1.

`atom$de_start_extraction <0>`

Initializes the Data Manager tool for local operations.

`atom$de_get_data_pointer`

Retrieves data and returns a pointer to the data.

`atom$uni_save_result`

Saves the result of the previous atom.

`atom$de_end_extraction`

Ends the data extraction sequence.

`atom$man_end_context`

Returns the context to the object previously in context.

`atom$uni_get_result`

Gets the result of the previous **atom$uni_save_result**.

```
➪ atom$uni_use_last_atom_string <prot$file>
    <atom$fm_ini_write_data>
```

Writes the data retrieved with the previous atom to the current .INI file.

atom$fm_item_get
4 ($04)

Description

atom$fm_item_get retrieves the value of a File Manager context register. For a list of File Manager context registers, see **atom$fm_item_type**.

If no argument is specified, the value of the current context register is used.

Syntax

```
atom$fm_item_get [<fm_register>]
```

`[<fm_register>]`	Specifies a context register.

Return Value

The current value of the specified register.

Example

The following is an example of how to retrieve the value of a File Manager context register:

```
➪ atom$fm_item_get <fmi_date>
```

The example is explained as follows:

```
➪ atom$fm_item_get <fmi_date>
```

Retrieves the value of fmi_date context register.

atom$fm_item_set
3 ($03)

Description

atom$fm_item_set sets the value of the current File Manager Tool context register. This atom is used with **atom$fm_item_type**.

When used with **atom$uni_use_last_atom_value**, this atom can also set a register to an action variable value, the text of a display object, and so on. To copy the value from one register to another, for example, use **atom$fm_item_get**, then **atom$uni_use_last_atom** with **atom$fm_item_set** as the argument.

Syntax

```
atom$fm_item_set <value>
```

`<value>`	Specifies a value.

Return Value

Unchanged.

Example

The following is an example of how to set the value of the fmi_persistent_path context register:

```
   atom$fm_item_type <fmi_persistent_path>
➪ atom$fm_item_set <1-0-1234>
```

The example is explained as follows:

```
atom$fm_item_type <fmi_persistent_path>
```

Specifies the fmi_persistent_path context register.

```
➪ atom$fm_item_set <1-0-1234>
```

Sets the value of the fmi_persistent_path register to 1-0-1234.

atom$fm_item_type

2 ($02)

Description

atom$fm_item_type specifies a File Manager Tool context register. This atom is used with **atom$fm_item_set**.

Syntax

```
atom$fm_item_type <fm_register>
```

`<fm_register>`	Specifies a context register. Values are as follows:
	fmi_file_type_id (1) — current file type ID.
	fmi_file_group_id (2) — current file group ID.
	fmi_file_type_ext (3) — current file type extension.
	fmi_file_type_desc (4) — current file type description.
	fmi_file_type_flags (5) — current file type flags.
	fmi_filename (6) — current file name.
	fmi_path (7) — current file directory path.
	fmi_filespec (8) — current file fully qualified path and name.
	fmi_handle (9) — current file handle (0 if the file is not open).
	fmi_error_code (10) — error code returned by the last file operation. For a list of possible values, see **atom$fm_handle_error**.
	fmi_custom_data (11) — 32-bit numeric value passed with all broadcast messages.
	fmi_text_width (12) — line width of files opened in one of the text modes.

fmi_text_indent (13) — number of characters to indent each line for files opened in one of the text modes.

fmi_text_flags (15) — processing instructions for data in files opened in text mode. Values are as follows:

- 0 — no action.
- 4 — translate CR to CRLF.
- 8 — translate 7F to CR.
- 12 — translate CR to CRLF and 7F to CR.
- 16 — translate ESC sequences.
- 20 — translate CR to CRLF and translate ESC sequences.
- 24 — translate 7F to CR and translate ESC sequences.
- 28 — translate CR to CRLF, 7F to CR and translate ESC sequences.

fmi_dialog_flags (16) — processing instructions for controlling the appearance of the dialog boxes displayed by the File Manager tool. Values are as follows:

- 0 — do not display the Skip push button.
- 1 — display the Skip push button.

fmi_and_mask (17) — 32-bit AND mask.

fmi_xor_mask (18) — 32-bit exclusive OR mask.

fmi_file_size (19) — initial file size in bytes.

fmi_broadcast_return_value (21) — value returned by the last **atom$fm_broadcast**. The tool that handled the broadcast message generates the value.

fmi_file_type_list_size (22) — number of files in the current context file list.

fmi_print_from_page (23) — initial page number to print.

fmi_print_to_page (24) — final page number to print.

fmi_print_min_page (25) — minimum page number that can be printed.

fmi_print_max_page (26) — maximum page number that can be printed.

fmi_print_copies (27) — number of copies to print

fmi_hdc (28) — device context handle in which printing occurs. Reserved for internal File Manager tool use.

fmi_print_job_name (29) — title of the print job.

fmi_date (30) — current date in the format "Sunday, January 1, 1995." This register is read-only.

fmi_time (31) — current time in the format "10:30 p.m." This register is read-only.

fmi_text_mode (32) — file mode. This register is automatically set by **atom$fm_open_file**.

- 0 — binary mode.
- 1 — text mode.

fmi_ini_string_length (33) — maximum length of a string that can be written using **atom$fm_ini_write_data**.

fmi_ini_file (34) — full path name of an initialization (.INI) file.

fmi_ini_group (35) — internal group name in an initialization (.INI) file.

fmi_ini_key (36) — internal key name in an initialization (.INI) file.

fmi_ini_data_type (37) — type of data in an initialization (.INI) file.

fmi_thumbnail (38) — image handle that represents the thumbnail image to be displayed.

fmi_thumbnail_size (39) — size in bytes of the thumbnail image.

fmi_persistent_path (40) — global ID of the database record containing the path associated with the next **atom$fm_dialog_get_file** or **atom$fm_dialog_put_file**. For more information about global IDs, see Chapter 11, "The Database Manager Tool."

Return Value

Unchanged.

Example

The following is an example of how to specify a File Manager Tool context register:

```
➪ atom$fm_item_type <fmi_persistent_path>
  atom$fm_item_set <1-0-1234>
```

The example is explained as follows:

```
➪ atom$fm_item_type <fmi_persistent_path>
```

Specifies the fmi_persistent_path context register.

```
atom$fm_item_set <1-0-1234>
```

Sets the value of the fmi_persistent_path register to 1-0-1234.

atom$fm_open_file
22 ($16)

Description

atom$fm_open_file opens an existing file and positions the file pointer to the beginning of the file.

atom$fm_open_file sets the value of the fmi_text_mode context register to reflect the mode, either binary or text, with which the file was opened. For more information about the possible values for this register, see **atom$fm_item_type**.

atom$fm_open_file also sets the value of the fmi_error_code context register to reflect the result of the operation. For more information about the possible values for this register, see **atom$fm_handle_error**.

Syntax

```
atom$fm_open_file <mode>
```

`<mode>`	Specifies the mode. Values are as follows: 0 — binary mode for read operations only. 1 — binary mode for write operations only. 2 — binary mode for read and write operations. 3 — text mode for read operations only. 4 — text mode for write operations only. 5 — text mode for read and write operations.

Return Value

Unchanged.

Example

The following is an example of how to open an existing file and position the file pointer to the beginning of the file:

```
  atom$fm_item_type <fmi_filespec>
  atom$fm_item_set <c:\test.txt>
➪ atom$fm_open_file <5>
```

The example is explained as follows:

```
atom$fm_item_type <fmi_filespec>
```

Specifies the fmi_filespec context register.

```
atom$fm_item_set <c:\test.txt>
```

Sets the value of the fmi_filespec register to c:\test.txt.

```
➪ atom$fm_open_file <5>
```

Opens an existing file in text mode for read and write operations and positions the file pointer to the beginning of the file.

atom$fm_position_eof
29 ($1D)

Description

atom$fm_position_eof sets the file pointer to the end of the file. The file must be open.

atom$fm_position_eof sets the value of the fmi_error_code context register to reflect the result of the operation. For more information about the possible values for this register, see **atom$fm_handle_error**.

Syntax

```
atom$fm_position_eof
```

Return Value

Unchanged.

Example

The following is an example of how to set the file pointer to the end of the file:

```
  atom$fm_item_type <fmi_filespec>
  atom$fm_item_set <c:\test.txt>
  atom$fm_open_file <5>
➪ atom$fm_position_eof
  atom$fm_append_data <Hello>
```

The example is explained as follows:

```
atom$fm_item_type <fmi_filespec>
```

Specifies the fmi_filespec context register.

```
atom$fm_item_set <c:\test.txt>
```

Sets the value of the fmi_filespec register to c:\test.txt.

```
atom$fm_open_file <5>
```

Opens an existing file in text mode for read and write operations and positions the file pointer to the beginning of the file.

➪ `atom$fm_position_eof`

Sets the file pointer to the end of the file.

`atom$fm_append_data <Hello>`

Appends data at the current position to the file in context.

atom$fm_position_file
28 ($1C)

Description

atom$fm_position_file sets the file pointer to a specific offset from the beginning of the file. The file must be open.

atom$fm_position_file sets the value of the fmi_error_code context register to reflect the result of the operation. For more information about the possible values for this register, see **atom$fm_handle_error**.

Syntax

```
atom$fm_position_file <offset>
```

`<offset>`	Specifies the offset in bytes from the beginning of the file.

Return Value

Unchanged.

Example

The following is an example of how to set the file pointer to a specific offset from the beginning of the file:

```
  atom$fm_item_type <fmi_filespec>
  atom$fm_item_set <c:\test.txt>
  atom$fm_open_file <5>
➪ atom$fm_position_file <27>
  atom$fm_append_data <Hello>
```

The example is explained as follows:

```
atom$fm_item_type <fmi_filespec>
```

Specifies the fmi_filespec context register.

```
atom$fm_item_set <c:\test.txt>
```

Sets the value of the fmi_filespec register to c:\test.txt.

```
  atom$fm_open_file <5>
```

Opens an existing file in text mode for read and write operations and positions the file pointer to the beginning of the file.

```
➪ atom$fm_position_file <27>
```

Sets the file pointer to an offset of 27 bytes from the beginning of the file.

```
  atom$fm_append_data <Hello>
```

Appends data at the current position to the file in context.

atom$fm_read_file
37 ($25)

Description

atom$fm_read_file reads data from a file. The file must be open.

If no argument is specified, the contents of the entire file is read.

atom$fm_read_file sets the value of the fmi_error_code context register to reflect the result of the operation. For more information about the possible values for this register, see **atom$fm_handle_error**.

Syntax

```
atom$fm_read_file [<bytes>]
```

`[<bytes>]`	Specifies the number of bytes to read

Return Value

0 (zero) if no error; otherwise, non-zero.

Example

The following is an example of how to read data from a file:

```
  atom$fm_item_type <fmi_filespec>
  atom$fm_item_set <c:\test.txt>
  atom$fm_open_file <3>
  atom$fm_position_file <27>
➪ atom$fm_read_file <10>
```

The example is explained as follows:

```
atom$fm_item_type <fmi_filespec>
```

Specifies the fmi_filespec context register.

```
atom$fm_item_set <c:\test.txt>
```

Sets the value of the fmi_filespec register to c:\test.txt.

```
atom$fm_open_file <3>
```

Opens an existing file in text mode for read operations and positions the file pointer to the beginning of the file.

```
atom$fm_position_file <27>
```

Sets the file pointer to an offset of 27 bytes from the beginning of the file.

```
➪ atom$fm_read_file <10>
```

Reads 10 bytes of data from the file.

atom$fm_register
11 ($0B)

Description

atom$fm_register registers a new file type or group. This atom sets the value of fmi_file_type_id or fm_file_group_id to the ID of the new file type or group.

If you are registering a file type, you must set the fmi_file_type_ext and the fmi_file_type_desc context registers before using **atom$fm_register**.

Syntax

```
atom$fm_register (<FMR_FILE_TYPE> | <FMR_FILE_GROUP>)
```

`<FMR_FILE_TYPE>`	Specifies registration of a new file type.
`<FMR_FILE_GROUP>`	Specifies registration of a new group.

Return Value

Unchanged.

Example

The following is an example of how to register a new file type:

```
  atom$fm_start
  atom$fm_item_type <fmi_file_type_ext>
  atom$fm_item_set <GIF>
  atom$fm_item_type <fmi_file_type_desc>
  atom$fm_item_set <GIF Files>
➪ atom$fm_register <FMR_FILE_TYPE>
  .
  .
  .
  atom$fm_end
```

The example is explained as follows:

```
atom$fm_start
```

Creates a File Manager Tool context.

```
atom$fm_item_type <fmi_file_type_ext>
```

Specifies the fmi_file_type_ext context register.

```
atom$fm_item_set <GIF>
```

Sets the value of the current context register to GIF.

```
atom$fm_item_type <fmi_file_type_desc>
```

Specifies the fmi_file_type_desc context register.

```
atom$fm_item_set <GIF Files>
```

Sets the value of the current context register to GIF Files.

```
➪ atom$fm_register <FMR_FILE_TYPE>
```

Registers a new file type and creates a file type ID.

```
atom$fm_end
```

Disposes of the File Manager Tool context.

atom$fm_rename_file

24 ($18)

Description

atom$fm_rename_file renames a file. The path name is not changed.

atom$fm_rename_file sets the value of the fmi_error_code context register to reflect the result of the operation. For more information about the possible values for this register, see **atom$fm_handle_error**.

Syntax

```
atom$fm_rename_file <name>
```

`<name>`	Specifies a filename.

Return Value

Unchanged.

Example

The following is an example of how to rename a file:

```
   atom$fm_item_type <fmi_filespec>
   atom$fm_item_set <c:\download\badname.jpg>
➪ atom$fm_rename_file <goodname.jpg>
```

The example is explained as follows:

```
atom$fm_item_type <fmi_filespec>
```

Specifies the fmi_filespec context register.

```
atom$fm_item_set <c:\download\badname.jpg>
```

Sets the value of the fmi_filespec context register to c:\download\badname.jpg.

```
➪ atom$fm_rename_file <goodname.jpg>
```

Renames the file to goodname.jpg.

atom$fm_set_relative_path
36 ($24)

Description

atom$fm_set_relative_path sets the path of the current context to the specified path, relative to the path of the online service application directory.

Syntax

```
atom$fm_set_relative_path <path>
```

`<path>`	Specifies the path.

Return Value

Preserves lValue.

Example

The following is an example of how to set the path of the current context to the specified path:

```
➪ atom$fm_set_relative_path <download>
```

The example is explained as follows:

```
➪ atom$fm_set_relative_path <download>
```

Sets the path of the current context to dowload.

atom$fm_start
0 ($00)

Description

atom$fm_start creates a File Manager Tool context for the current atom stream.

Syntax

```
atom$fm_start
```

Return Value

Unchanged.

Example

The following is an example of how to create a File Manager Tool context for the current atom stream:

```
➪ atom$fm_start
  .
  .
  .
  atom$fm_end
```

The example is explained as follows:

```
➪ atom$fm_start
```

Creates a File Manager Tool context.

```
atom$fm_end
```

Disposes of the File Manager Tool context.

atom$fm_unregister
12 ($0C)

Description

atom$fm_unregister unregisters a file type or group.

If no argument is specified, the value of the fmi_file_type_id context register is used.

Syntax

```
atom$fm_unregister [<file_type_id>]
```

`[<file_type_id>]`	Specifies a file type ID to be unregistered.

Return Value

Unchanged.

Example

The following is an example of how to unregister a file type or group:

```
  atom$fm_find_file_type <GIF>
➪ atom$fm_unregister
```

The example is explained as follows:

```
atom$fm_find_file_type <GIF>
```

Searches registered file types for one matching GIF and sets the fmi_file_type_id context register to the ID of the file type found.

```
➪ atom$fm_unregister
```

Unregisters the GIF file type.

atom$fm_unregister_list
38 ($26)

Description

atom$fm_unregister_list unregisters every file type in the current context.

Syntax

```
atom$fm_unregister_list
```

Return Value

Unchanged.

Example

The following is an example of how to unregister every file type in the current context:

```
➪ atom$fm_unregister_list
```

The example is explained as follows:

```
➪ atom$fm_unregister_list
```

Unregistera every file type in the current context.