Manual: FDO91 Manual
Chapter 7: Universal Protocol defines the universal protocol atoms and provides information for controlling atom streams.
Last updated: July 1997

CHAPTER 7
Universal (UNI) Protocol

The Universal (UNI) protocol (protocol ID 0) consists of atoms that provide basic utility and control functions common to all platforms that use the FDO91 language. This functionality includes stream execution control and atom management, such as starting, ending, and controlling atom streams and managing large atoms. The protocol also supports some data manipulation operations and client session control.

Universal Protocol Atoms

The Universal (UNI) protocol functions and their associated atoms follow:

Function	**Atoms**
Stream execution control	atom$uni_abort_stream atom$uni_change_stream_id atom$uni_data atom$uni_end_loop atom$uni_end_stream atom$uni_force_processing atom$uni_get_current_stream_id atom$uni_get_first_stream atom$uni_get_next_stream atom$uni_get_stream_window

	atom$uni_hold atom$uni_invoke_local atom$uni_invoke_local_later atom$uni_invoke_local_preserve atom$uni_invoke_no_context atom$uni_set_data_length atom$uni_start_loop atom$uni_start_stream atom$uni_start_stream_wait_on atom$uni_transaction_id atom$uni_sync_skip atom$uni_wait_off_end_stream
Data manipulation operations	atom$uni_convert_last_atom_data atom$uni_convert_last_atom_string atom$uni_data atom$uni_get_result atom$uni_save_result atom$uni_use_last_atom_data atom$uni_use_last_atom_string atom$uni_use_last_atom_value
Large atom management	atom$uni_end_large_atom atom$uni_large_atom_segment atom$uni_start_large_atom

Client session control	atom$uni_cancel_action atom$uni_set_watchdog_interval atom$uni_start_stream_wait_on atom$uni_wait_off atom$uni_wait_off_end_stream atom$uni_wait_on
Language data type for display	atom$uni_end_typed_data atom$uni_next_atom_typed atom$uni_start_typed_data
Atom stream debugging	atom$uni_break atom$uni_diagnostic_msg atom$uni_single_step atom$uni_void

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

atom$uni_abort_stream
3 ($03)

Description

atom$uni_abort_stream causes the current atom stream to be terminated. Note that if this atom is executed in a host-to-client stream, the stream is terminated for the current packet only. Any atoms stored in the buffer are ignored without further processing.

Syntax

```
atom$uni_abort_stream
```

Return Value

None.

Example

The following example terminates the current stream if the last return value is 0:

```
atom$uni_start_stream
.
.
.
atom$if_last_return_false <5>
atom$uni_abort_stream
atom$uni_sync_skip <5>
.
.
.
atom$_uni_end_stream
```

The example is explained as follows:

```
atom$if_last_return_false <5>
```

This example tests the last return value for a true or false state.

If the result is true, which means the last return value is false, the atoms that appear between **atom$if_last_return_false <5>** and **atom$uni_sync_skip <5>** are executed, followed by the atoms that appear after **atom$uni_sync_skip <5>**.

If the result is false, which means the last return value is true, the atoms that appear between **atom$if_last_return_false <5>** and **atom$uni_sync_skip <5>** are skipped, and the atoms that follow **atom$uni_sync_skip <5>** are executed.

➪ `atom$uni_abort_stream`

This example aborts the atom stream if **atom$if_last_return_false <5>** evaluates to true.

`atom$uni_sync_skip <5>`

This atom marks the end of the true atoms, which is where execution skips to in the atom stream if **atom$if_last_return_false <5>** evaluates to false.

atom$uni_break
43 ($2B)

Description

atom$uni_break breaks the processing of the current stream and invokes a programmer's Windows debugger.

Syntax

```
atom$uni_break
```

Return Value

None.

Example

The following example breaks the execution of the current atom stream and invokes a programmer's Windows debugger when the numeric value of register A is 0:

```
atom$uni_start_stream
.
.
.
atom$if_numa_false_then <5>
➪ atom$uni_break
atom$uni_sync_skip <5>
.
.
.
atom$_uni_end_stream
```


atom$uni_cancel_action
49 ($31)

Description

atom$uni_cancel_action executes the cancel action associated with the top-most form.

Syntax

```
atom$uni_cancel_action
```

Return Value

None.

Example

The following example executes the cancel action associated with the top-most form when a member pressed the ESC key:

```
atom$uni_start_stream
atom$uni_cancel_action
.
.
.
atom$uni_end_stream
```

atom$uni_change_stream_id
35 ($23)

Description

atom$uni_change_stream_id causes the flow of execution to change from the current atom stream to the specified atom stream. If the terminate flag is set to a non-zero value, the current stream is terminated and the specified stream is activated. (If the specified stream ID is not valid, the current stream is terminated.) If the terminate flag is set to 0, the specified stream is activated, and the current stream is put on hold until the specified stream completes processing.

Syntax

```
atom$uni_change_stream_id <flag[, stream_ID]>
```

`<flag>`	Specifies a 1-byte termination flag. Values are 0 or greater.
`[<stream_ID>]`	Specifies an optional stream ID. Values are 1 or greater.

Return Value

None.

Examples

The following example terminates the current stream (ID 24) and activates stream ID 23 when register A is greater than 99:


```
atom$uni_start_stream
.(This is stream 23)
.
.
atom$var_number_set <B, 99>
      atom$uni_start_stream
      . (This is stream 24)
      .
      .
      atom$if_numa_gt_numb_then <5>
➪     atom$uni_change_stream_id <1, 23>
      atom$uni_sync_skip <5>
      .
      .
      .
      atom$uni_end_stream
. (This is stream 23)
.
.
atom$uni_end_stream
```

The following example puts the current stream (ID 24) on hold and activates stream ID 23 when register A is greater than 99:

```
atom$uni_start_stream
.(This is stream 23)
.
.
atom$var_number_set <B, 99>
      atom$uni_start_stream
      . (This is stream 24)
      .
      .
      atom$if_numa_gt_numb_then <5>
➪     atom$uni_change_stream_id <0, 23>
      atom$uni_sync_skip <5>        .
      .
      .
      atom$uni_end_stream
. (This is stream 23)
.
.
atom$uni_end_stream
```

atom$uni_convert_last_atom_data
45 ($2D)

Description

atom$uni_convert_last_atom_data converts the last return value to characters and returns a pointer to the character string. This string is valid until another string is converted.

Syntax

```
atom$uni_convert_last_atom_data
```

Return Value

Pointer to the character string.

Example

The following example converts the last return value to characters. For this example, the last return value is decimal 617:

```
atom$uni_start_stream
.
.
.
atom$man_set_context_relative <56>
atom$var_lookup_by_id <A, 23>
atom$var_number_get <A>
atom$uni_save_result
.
.
.
atom$man_change_context_relative <11>
atom$uni_get_result
➪ atom$uni_convert_last_atom_data
atom$uni_use_last_atom_string <atom$man_replace_data>
atom$man_end_context
.
.
atom$man_update_display
atom$uni_end_stream
```

The result of this example is that a pointer to the string "617" is returned and placed in object 11.

atom$uni_convert_last_atom_string
42 ($2A)

Description

atom$uni_convert_last_atom_string converts the last string return value to a numeric value and returns the result. (Note that valid string values are null-terminated.)

Syntax

```
atom$uni_convert_last_atom_string
```

Return Value

Numeric value of the string.

Example

The following example converts the last string return value to a numeric value. For this example, the last return value points to the string "123:"

```
atom$uni_start_stream
.
.
.
atom$man_set_context_relative <56>
atom$var_lookup_by_id <A, 23>
atom$var_string_get <A>
atom$uni_save_result
.
.
.
atom$man_change_context_relative <11>
atom$uni_get_result
➪ atom$uni_convert_last_atom_string
atom$uni_use_last_atom_value <atom$var_number_set, A>
atom$var_number_save <A, 10>
atom$man_end_context
.
.
atom$man_update_display
atom$uni_end_stream
```

The result of this example is that the return value is 123 in decimal, and it is saved in register 10 for object 11.

atom$uni_data
14 ($0E)

Description

atom$uni_data delivers raw data in a stream to the host. The preceding atoms set the machine state and context for the type of data expected.

Syntax

```
atom$uni_data <raw_data>
```

`<raw_data>` Specifies the raw data to be sent.

Return Value

Pointer to the raw data specified.

Example

The following example sends raw data to the host:

```
  atom$reg_data_type <reg_data_city>
➪ atom$uni_data <56x 69x 65x 6Ex 6Ex 61x>
```

The example is explained as follows:

```
atom$reg_data_type <reg_data_city>
```

Tells a host register process during new member registration to expect data describing the member's city.

```
➪ atom$uni_data <56x 69x 65x 6Ex 6Ex 61x>
```

Transmits the data (56x 69x 65x 6Ex 6Ex 61x) that indicates the new member's city (Vienna) to the host.

atom$uni_diagnostic_msg
36 ($24)

Description

atom$uni_diagnostic_msg lets you put status notification messages in streams. This atom is used for debugging purposes.

Syntax

```
atom$uni_diagnostic_msg <flag[, string]>
```

`<flag>`	Specifies a 1-byte debug flag, which is currently not used. The default value is 0.
`[<string>]`	Specifies an optional message string.

Return Value

None.

Example

The following example displays a status notification message for testing purposes:

```
atom$uni_start_stream
.
.
.
➪ atom$uni_diagnostic_msg <0, "Create toolbar">
.
.
.
atom$uni_end_stream
```

When this atom command is executed, the message "Create toolbar" appears on screen. You can insert this command into an atom stream prior to atoms that create and display a toolbar. If errors are occurring and you think the toolbar is causing them, this command lets you easily pinpoint the problem.

atom$uni_end_large_atom
6 ($06)

Description

atom$uni_end_large_atom completes the large atom being collected (started with **atom$uni_start_large_atom**). For efficiency purposes, the **Atomizer** tool mathematically determines whether or not to construct a large atom.

Syntax

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

`[<data>]` Specifies optional unlimited data.

Return Value

Result of the atom processed.

Examples

The following example assembles large amounts of text into a large atom via the **Atomizer** tool:

```
atom$uni_start_large_atom <atom$man_text, 32>
atom$uni_large_atom_segment <Text goes here...>
atom$uni_end_large_atom <Rest of text goes here...>
```

The example is explained as follows:

```
atom$uni_start_large_atom <atom$man_text, 32>
```

Starts a large atom that assembles data into 0 or more segments.

`<atom$man_text>` Indicates that the large atom is an **atom$man_text** atom.

`<32>` Indicates that the length of the large atom is 32 bytes.

```
atom$uni_large_atom_segment <Text goes here...>
```

Assembles the data being sent into as many segments as necessary.

➪ `atom$uni_end_large_atom` <*Rest of text goes here...*>

Sends the rest of the text, if more exists, and ends the large atom.

atom$uni_end_loop
9 ($09)

Description

atom$uni_end_loop designates the end of a series of atoms that can be repeated. This atom is used in conjunction with the **atom$uni_start_loop** command.

Syntax

```
atom$uni_end_loop
```

Return Value

None.

Example

The following example illustrates the use of a stream execution loop, which will append "Hello there" 10 times to the object in context:

```
atom$var_number_set <A, 10>
atom$var_number_set <B, 0>
atom$uni_start_loop
atom$man_append_data <"Hello there">
atom$var_number_decrement <A>
atom$if_numa_neq_numb_then <35>
➪ atom$uni_end_loop
atom$uni_sync_skip <35>
```

The example is explained as follows:

```
atom$var_number_set <A, 10>
```

Sets the numeric area of register A to 10.

```
atom$var_number_set <B, 0>
```

Sets the numeric area of register B to 0.

```
atom$uni_start_loop
```

Defines the beginning of a loop that will repeatedly execute atoms until a specified termination point is reached. In this case, the loop will continue until the numeric value of register A reaches 0.


```
atom$man_append_data <"Hello there">
```

Appends the text "Hello there" to the current object.

```
atom$var_number_decrement <A>
```

Decrements the numeric value of register A by 1, which results in a value of 9.

```
atom$if_numa_neq_numb_then <35>
```

Compares the numeric content of register A with the numeric content of register B to see if they are equal.

After the first execution of the loop, the register A numeric content is 9 and register B numeric content is 0 so this command evaluates to false and execution returns to the beginning of the loop.

The loop executes until the register A numeric content is decremented to 0. At that point, this command evaluates to true, and **atom$uni_end_loop** is executed, followed by **atom$uni_sync_skip <35>**.

```
atom$uni_end_loop
```

Defines the end of the loop.

```
atom$uni_sync_skip <35>
```

Processing continues here after **atom$if_numa_neq_numb_then <35>** evaluates to true and the loop is terminated.

atom$uni_end_stream
2 ($02)

Description

atom$uni_end_stream designates the end of the current atom stream.

Syntax

```
atom$uni_end_stream
```

Return Value

None.

Example

The following example terminates the processing of an atom stream:

```
atom$uni_start_stream
.
.
.
atom$uni_end_stream
```


atom$uni_end_typed_data
24 ($18)

atom$uni_end_type_data is described in Appendix B, "Internationalization (i18n) Atoms."

atom$uni_force_processing
32 ($20)

Description

atom$uni_force_processing signals the client computer to change the normal atom processing mode that processes each atom in the stream as it is received to a mode that collects and then processes the buffered chunks of atoms from a stream received.

Note that whenever this atom arrives, any buffered atoms are processed before the processing mode is changed. This lets this atom precisely define the processing chunks where required.

Syntax

```
atom$uni_force_processing <word>
```

`<word>`	Specifies the processing mode of the atom stream. Values are as follows:	
	EVERY	Processes all atoms as they arrive (as normal).
	AT_END	Processes no atoms until the end of the stream.

Return Value

None.

Example

The following example changes the execution mode to defer atom processing until the end of the stream is reached:

```
atom$uni_start_stream
.
.
.
atom$uni_force_processing <AT_END>
atom$uni_end_stream
atom$uni_start_stream
.
.
.
atom$uni_end_stream
```


atom$uni_get_current_stream_id
50 ($32)

Description

atom$uni_get_current_stream_id returns the ID of the current stream.

Syntax

```
atom$uni_get_current_stream_id
```

Return Value

ID of the current stream.

Example

The following example gets the ID of the current stream:

```
atom$uni_start_stream
.
.
.
      atom$uni_start_stream
      .
      .
      .
➪     atom$uni_get_current_stream_id
      .
      .
      .
      atom$uni_end_stream
.
.
.
atom$uni_end_stream
```

atom$uni_get_first_stream
46 ($2E)

Description

atom$uni_get_first_stream returns the ID of the first stream in the list of open streams.

Syntax

```
atom$uni_get_first_stream
```

Return Value

ID of first open stream.

Example

The following example gets the ID of the first stream that was opened:

```
atom$uni_start_stream
.
.
.
      atom$uni_start_stream
      .
      .
      .
            atom$uni_start_stream
            .
            .
            .
      ➪     atom$uni_get_first_stream
            .
            .
            .
            atom$uni_end_stream
      .
      .
      .
      atom$uni_end_stream
.
.
.
atom$uni_end_stream
```


atom$uni_get_next_stream
47 ($2F)

Description

atom$uni_get_next_stream returns the ID of the next stream in the open stream list.

Syntax

```
atom$uni_get_next_steam
```

Return Value

ID of next open stream.

Example

The following example gets the ID of the next stream:

```
atom$uni_start_stream
.
.
.
      atom$uni_start_stream
      .
      .
      .
            atom$uni_start_stream
            .
            .
            .
            atom$uni_get_first_stream
      ➪     atom$uni_get_next_stream
            .
            .
            .
            atom$uni_end_stream
      .
      .
      .
      atom$uni_end_stream
.
.
.
atom$uni_end_stream
```

The example is explained as follows:

`atom$uni_get_first_stream`

Gets the ID of the first open stream.

➪ `atom$uni_get_next_stream`

Gets the ID of the next open stream.

atom$uni_get_result
21 ($15)

Description

atom$uni_get_result gets the result of the previous **atom$uni_save_result** atom and then uses it for the next atom.

Syntax

```
atom$uni_get_result
```

Return Value

Value of the last saved atom result.

Example

The following example executes **atom$uni_get_result** that gets the numeric value that was saved with **atom$uni_save_result** in a previous operation. For this example, the last save result is decimal 617.

```
atom$uni_start_stream
.
.
.
atom$man_set_context_relative <56>
atom$var_lookup_by_id <A, 23>
atom$var_number_get <A>
atom$uni_save_result
.
.
.
atom$man_change_context_relative <11>
➪ atom$uni_get_result
atom$uni_convert_last_atom_data
atom$uni_use_last_atom_string <atom$man_replace_data>
atom$man_end_context
.
.
atom$man_update_display
atom$uni_end_stream
```

The result of this example is that a pointer to the string "617" is returned and placed in object 11.

atom$uni_get_stream_window
48 ($30)

Description

atom$uni_get_stream_window returns the global ID of the window associated with the specified stream or that of the current stream if no stream ID is specified.

Syntax

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

`[<dword>]` Specifies an optional stream ID.

Return Value

Global ID of the window.

Example

The following example gets the global ID of a window that is associated with the current stream:

```
atom$uni_start_stream
.
.
.
      atom$uni_start_stream
      .
      .
      .
➪     atom$uni_get_stream_window
      .
      .
      .
      atom$uni_end_stream
.
.
.
atom$uni_end_stream
```


atom$uni_hold
37 ($25)

Description

atom$uni_hold causes the specified atom stream to be placed on hold. If no stream ID is specified, the current stream is held.

Syntax

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

`[<dword>]` Specifies an optional stream ID.

Return Value

None.

Examples

The following example places the current stream on hold:

```
atom$uni_start_stream
.
.
.
      atom$uni_start_stream
      .
      .
      .
➪     atom$uni_hold
      .(This stream on hold)
      .
      .
      atom$uni_end_stream
.
.
.
atom$uni_end_stream
```

The following example places a stream with an ID of 23 on hold:

```
atom$uni_start_stream
.(This is stream 23)
.
.
      atom$uni_start_stream
      .
      .
      .
➪     atom$uni_hold <23>
      .
      .
      .
      atom$uni_end_stream
. (This stream on hold)
.
.
atom$uni_end_stream
```


atom$uni_invoke_local
20 ($14)

Description

atom$uni_invoke_local invokes the atom stream contained in the specified database record. This may be a complete atom stream or a fragment. The context of the current object is assumed when this record is invoked. The context for the next atom processed is based on the final context of the database record processed.

Syntax

```
atom$uni_invoke_local <dword>
```

`<dword>` Specifies a database record.

Return Value

Database record from which the atom stream was invoked.

Example

The following example invokes a stream retrieved from a database record:

```
atom$uni_start_stream
atom$man_set_context_relative <56>
.
.
.
atom$uni_invoke_local <3005>
      .
      .
      .
      atom$uni_end_stream
.
.
.
atom$uni_end_stream
```

The result of this example is that database record 3005 is looked up and given to the **Atomizer** tool for processing.

atom$uni_invoke_local_later
41 ($29)

Description

atom$uni_invoke_local_later invokes the atom stream contained in the given database record when the next client computer idle period begins. This may be a complete atom stream or a fragment. The context of the current object is assumed when this record is invoked.

Syntax

```
atom$uni_invoke_local_later <dword>
```

`<dword>` Specifies a database record.

Return Value

Database record from which the atom stream was invoked.

Example

The following example retrieves a stream from a database record and invokes it when the client computer is idle:

```
atom$uni_start_stream
atom$man_set_context_relative <56>
.
.
.
➪ atom$uni_invoke_local_later <1206>
.
.
.
atom$man_end_context
atom$uni_end_stream
atom$uni_start_stream

. (This is stream 1206)
.
.
atom$uni_end_stream
```

The result of this example is that database record 1206 is looked up and given to the **Atomizer** tool for processing when the client application is idle.

atom$uni_invoke_local_preserve
40 ($28)

Description

atom$uni_invoke_local_preserve preserves the context of the current object when the atom stream contained in the given database record is invoked. This may be a complete atom stream or a fragment. After the database record is processed, the next atom processed will have context restored to its normal state (the current object).

Syntax

```
atom$uni_invoke_local_preserve <dword>
```

`<dword>` Specifies a database record.

Return Value

Database record from which the atom stream was invoked.

Example

The following example retrieves a stream from the database and invokes it while preserving the context of the current object:

```
atom$uni_start_stream
atom$man_set_context_relative <56>
.
.
.
➪ atom$uni_invoke_local_preserve <3141>
        .
        .
        .
        atom$uni_end_stream
.(This is object context 56)
.
.
atom$uni_end_stream
```

The result of this example is that database record 3141 is looked up and given to the **Atomizer** tool for processing.

atom$uni_invoke_no_context
19 ($13)

Description

atom$uni_invoke_no_context invokes the atom stream contained in the given database record. This may be a complete atom stream or a fragment. The context of the current object is ignored when this record is invoked. The context for the next atom processed is based on the final context of the database record processed.

Syntax

```
atom$uni_invoke_no_context <dword>
```

`<dword>` Specifies a database record.

Return Value

Database record from which the atom stream was invoked.

Example

The following example retrieves a stream from the database and invokes it while ignoring the context of the current object:

```
atom$uni_start_stream
atom$man_set_context_relative <56>
.
.
.
➪ atom$uni_invoke_no_context <721>
        .
        .
        .
        atom$uni_end_stream
.
.
.
atom$uni_end_stream
```

The result of this example is that database record 721 is looked up and given to the **Atomizer** tool for processing.

atom$uni_large_atom_segment
5 ($05)

Description

atom$uni_large_atom_segment adds data to the large atom being processed. Zero or more large atom segments can follow **atom$uni_start_large_atom**. For efficiency purposes, the **Atomizer** tool mathematically determines whether or not to construct a large atom.

Syntax

```
atom$uni_large_atom_segment <data>
```

`<data>`	Specifies a large segment of data.

Return Value

None

Example

The following example splits large amounts of data into several atom segments:

```
atom$uni_start_large_atom <atom$man_append_data, 32>
atom$uni_large_atom_segment <Data goes here...>
.
.
(More atom$uni_large_atom_segment atoms, if necessary)
.
.
atom$uni_end_large_atom
```

The example is explained as follows:

```
atom$uni_start_large_atom <atom$man_append_data, 32>
```

Starts a large atom that assembles data into 0 or more segments.

`<atom$man_append_data>`	Indicates that the large atom is an **atom$man_append_data** atom.
`<32>`	Indicates that the length of the large atom is 32 bytes.

➪ `atom$uni_large_atom_segment` <*Data goes here...*>

Assembles the data being sent into as many segments as necessary.

`atom$uni_end_large_atom`

Ends the large atom.

atom$uni_next_atom_typed
22 ($16)

atom$uni_next_atom_typed is described in Appendix B, "Internationalization (i18n) Atoms."

atom$uni_save_result
12 ($0C)

Description

atom$uni_save_result saves the result of the last atom executed. Only one atom result can be saved at a time.

Syntax

```
atom$uni_save_result
```

Return Value

None.

Example

The following example saves the result of the last atom value for later use:

```
atom$uni_start_stream
.
.
.
atom$man_set_context_relative <56>
atom$var_lookup_by_id <A, 23>
atom$var_number_get <A>
➪ atom$uni_save_result
.
.
.
atom$man_change_context_relative <11>
atom$uni_get_result
atom$uni_convert_last_atom_data
atom$uni_use_last_atom_string <atom$man_replace_data>
atom$man_end_context
.
.
atom$man_update_display
atom$uni_end_stream
```


atom$uni_set_data_length
51 ($33)

Description

atom$uni_set_data_length sets the default data length for the current stream.

Syntax

```
atom$uni_set_data_length <data_length>
```

`<data_length>` Specifies the default data length in bytes.

Return Value

None.

Example

The following example sets the default data length for the current stream to 4 bytes:

```
atom$uni_start_stream
atom$uni_set_data_length <4>
.
.
.
atom$uni_end_stream
```

atom$uni_set_watchdog_interval
53 ($35)

Description

atom$uni_set_watchdog_interval sets the amount of time that elapses before the client's watchdog timer expires. The watchdog timer is used to detect when the host has failed to respond to a request.

When the watchdog interval has expired, an abort action is triggered. The standard abort action is to turn off the wait cursor and display a dialog stating **The host has failed to respond. Please continue**. An abort action is defined using **atom$act_set_criterion <17>**, which specifies a timeout action.

Syntax

```
atom$uni_set_watchdog_interval <dword>
```

`<dword>`	Specifies the watchdog interval in a client-dependent unit of time. Values are 1 tick or greater. (For example, the unit of time for the Windows client application is ticks, where 1 tick = 1/60th of a second.)

Return Value

None.

Example

The following example sets the watchdog interval to 45 seconds:

```
atom$uni_start_stream
.
.
.
➪ atom$uni_set_watchdog_interval <2700>
.
.
.
atom$uni_end_stream
```

The result of this example is that the watchdog interval is set to 45 seconds (2700 divided by 60 = 45). This means that the client will wait for 45 seconds for the host to respond before giving up and performing a specified abort action.

atom$uni_single_step
44 ($2C)

Description

atom$uni_single_step invokes or terminates single step execution of atoms in the stream. For each atom, the Single Stepper Dialog box appears and lets you modify any Action register value and the atom's data.

Syntax

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

`[<boolean>]` Invokes or terminates the single step atom execution mode. Values are as follows:

- 0 Invokes the single step execution mode. (Default)
- 1 Terminates the single step execution mode.

Return Value

None.

Examples

The following example invokes the single step atom execution mode:

```
atom$uni_single_step
```

The following example terminates the single step atom execution mode:

```
atom$uni_single_step <1>
```

atom$uni_start_large_atom
4 ($04)

Description

atom$uni_start_large_atom marks the beginning of a large atom. A large atom is an atom that does not fit in a data packet. The size limit for a large atom is 64K bytes. Each time a large atom is sent, it needs to be broken into a stream of atoms that begins with **atom$uni_start_large_atom** and ends with **atom$uni_end_large_atom**, with 0 or more **atom$uni_large_atom_segment** atoms in between.

For efficiency purposes, the **Atomizer** tool mathematically determines whether or not to construct a large atom.

Syntax for VP Designer

```
atom$uni_start_large_atom <atom, atom_length>
```

Syntax for form_edit

```
atom$uni_start_large_atom <proto> <atom> <atom_length>
```

`<proto>` This argument is not used for **VP Designer**.

For **form_edit**, this argument specifies a protocol name to which the large atom belongs. The possible protocol names are prefixed **prot$** and are defined on Stratus in **arclient>atoms_dir>protocols.file**. Some typical values are as follows:

prot$uni — Universal Protocol

prot$display — Display Manager Protocol

prot$action — Action Protocol

prot$extract — Data Manager Protocol

prot$buffer — Buffer Protocol

prot$database — Database Manager Protocol

prot$xfer — File Transfer Protocol

prot$file — File Manager Protocol

prot$list — List Manager Protocol

prot$code — Code Manager Protocol

prot$chat — Chat Protocol

prot$var — Variable Protocol

prot$async — Async Protocol

prot$shorthand — Shorthand Protocol

prot$if — IF Protocol

prot$mat — Display Attributes Protocol

prot$mip — Message Interchange Protocol

prot$reg — Registration Protocol

prot$mmi — Multimedia Interface Protocol

prot$imgxfer — Image Transfer Protocol

prot$image — Image Manager Protocol

prot$chart — Chart Protocol

prot$morg — Multimedia Organizer Protocol

prot$rich — Rich Protocol

prot$exapi — External API Protocol

prot$ccl — CCL Protocol

prot$p3 — P3 Protocol

prot$pakman — File Decompression Protocol

prot$address — Address Protocol

prot$mt — Debugging Protocol

`<atom>`	Specifies an atom used to interpret the atom stream once it has been completely received.
`<atom_length>`	Specifies an atom length that provides the receiver with an upper limit on the length of the large atom. Although this value can be used for memory allocation purposes, the true length of the large atom is measured.

Return Value

None.

Example

The following example assembles large amounts of text into a large atom:

```
atom$uni_start_large_atom <atom$man_text, 32>
atom$uni_large_atom_segment <Text goes here...>
atom$uni_end_large_atom
```

The example is explained as follows:

`atom$uni_start_large_atom <atom$man_text, 32>`

Starts a large atom that assembles data into 0 or more segments.

`<atom$man_text>`	Indicates that the large atom is **atom$man_text**.
`<32>`	Indicates that the length of the large atom is 32 bytes.

`atom$uni_large_atom_segment <Text goes here...>`

Assembles the data being sent into as many segments as necessary.

`atom$uni_end_large_atom`

Ends the large atom.

atom$uni_start_loop
8 ($08)

Description

atom$uni_start_loop designates the beginning of a series of atom executions that can be repeated. This atom is used in conjunction with **atom$uni_end_loop**. The only way to break out of the loop is with an atom from the Conditional protocol. For more information, see Chapter 5, "Conditional (IF) Protocol."

Syntax

```
atom$uni_start_loop
```

Return Value

None.

Example

The following example appends "Hello there" 10 times to the object in context using an atom processing loop:

```
  atom$var_number_set <A, 10>
  atom$var_number_set <B, 0>
➪ atom$uni_start_loop
  atom$man_append_data <"Hello there">
  atom$var_number_decrement <A>
  atom$if_numa_neq_numb_then <35>
  atom$uni_end_loop
  atom$uni_sync_skip <35>
```

The example is explained as follows:

```
atom$var_number_set <A, 10>
```

Sets the numeric area of register A to 10.

```
atom$var_number_set <B, 0>
```

Sets the numeric area of register B to 0.

➪ `atom$uni_start_loop`

Defines the beginning of a loop that will repeatedly execute atoms until a specified termination point is reached. In this case, the loop will continue until the numeric value of register A reaches 0.

`atom$man_append_data <"Hello there">`

Appends the text "Hello there" to the current object.

`atom$var_number_decrement <A>`

Decrements the numeric value of register A by 1, which gives it a value of 9.

`atom$if_numa_neq_numb_then <35>`

Compares the numeric content of register A to the numeric content of register B to see if they are equal.

After the first execution of the loop, the numeric value of register A is 9 and the numeric value of register B is 0 so this command evaluates to false and execution returns to the beginning of the loop.

The loop executes until the numeric value of register A is decremented to 0. At that point, this command evaluates to true, and **atom$uni_end_loop** is executed, followed by **atom$uni_sync_skip <35>**.

`atom$uni_end_loop`

Defines the end of the loop.

`atom$uni_sync_skip <35>`

Processing continues here after **atom$if_numa_neq_numb_then <35>** evaluates to true and the loop is terminated.

atom$uni_start_stream
1 ($01)

Description

atom$uni_start_stream appears at the beginning of every atom stream. An atom stream can have multiple or nested **atom$uni_start_stream** entries.

Syntax

```
atom$uni_start_stream
```

Return Value

None.

Example

The following example shows the initiation of two streams, one nested within the other:

```
➪ atom$uni_start_stream
  .
  .
  .
➪       atom$uni_start_stream
        .
        .
        .
        atom$uni_end_stream
  .
  .
  .
  atom$uni_end_stream
```

The example is explained as follows:

➪ `atom$uni_start_stream`

Signifies the beginning of an atom stream.

`atom$uni_end_stream`

Signifies the end of an atom stream.

atom$uni_start_stream_wait_on
17 ($11)

Description

atom$uni_start_stream_wait_on is identical to **atom$uni_start_stream** with the exception that it turns the wait cursor on. Atom streams may have multiple or nested **atom$uni_start_stream_wait_on** atoms.

Syntax

```
atom$uni_start_stream_wait_on
```

Return Value

None.

Example

The following example starts an atom stream and simultaneously turns on the wait cursor.

```
atom$uni_start_stream_wait_on
atom$sm_check_domain
atom$de_start_extraction
atom$de_validate <display | terminate>
.
.
.
atom$uni_end_stream
```


atom$uni_start_typed_data
23 ($17)

atom$uni_start_typed_data is described in Appendix B, "Internationalization (i18n) Atoms."

atom$uni_sync_skip
7 ($07)

Description

atom$uni_sync_skip serves as a label to mark the end of a conditional stream. This atom is used with atoms in the IF protocol. For more information, see Chapter 5, "Conditional (IF) Protocol."

Syntax

```
atom$uni_sync_skip <label>
```

`<label>`	Specifies a numeric value for a label that marks the end of the conditional stream.

Return Value

None.

Example

The following example illustrates the use of this atom to mark the end of conditional streams:

```
atom$if_online_then <1, 2>
.
.   (Atoms to execute if the atom evaluates to true...)
.
➪ atom$uni_sync_skip <1>
.
.   (Atoms to execute if the atom evaluates to false...)
.
➪ atom$uni_sync_skip <2>
.
.   (Stream execution continues here after execution of true or false atoms...)
.
```

The example is explained as follows:

```
atom$if_online_then <1, 2>
```

The result of this example evaluates to true if the member is online or false if the member is offline. If the member is online (true), the atoms that immediately follow are executed. If the member is offline (false), the atoms that fall between **atom$uni_sync_skip <1>** and **atom$uni_sync_skip <2>** are executed.

➪ `atom$uni_sync_skip <1>`

Marks the end of the true atom stream. When **atom$if_online_then <1, 2>** evaluates to true, the true atoms are executed, the false atoms are skipped, and stream execution continues after **atom$uni_sync_skip <2>**.

➪ `atom$uni_sync_skip <2>`

Marks the end of the false atom stream. When **atom$if_online_then <1, 2>** evaluates to false, the true atoms are skipped, the false atoms are executed, and stream execution continues immediately following this atom.

atom$uni_transaction_id

3C ($13)

Description

atom$uni_transaction_id associates an ID to the transaction in progress between the host and client application. Transaction IDs are generated by the client application. Because stream IDs are not always unique, transaction IDs may be required.

Syntax

```
atom$uni_transaction_id <dword>
```

`<dword>` Specifies the transaction ID. Values are 1 or greater.

Return Value

None.

Example

The following example assigns a transaction ID of 1 to a data on demand (DOD) activity:

```
  atom$uni_start_stream
  atom$dod_start
➪ atom$uni_transaction_id <1>
  atom$dod_hints <00x, 2ax, 00x, 2ax>
  atom$dod_end
  atom$uni_end_stream
```


atom$uni_use_last_atom_data
52 ($34)

Description

atom$uni_use_last_atom_data executes the specified atom using the data pointed to by the return value of the previously executed atom. This atom can append the data content of an input field to a database.

Syntax for VP Designer

```
atom$uni_use_last_atom_data <atom[, data]>
```

Syntax for form_edit

```
atom$uni_use_last_atom_data  <proto> <atom> [<data>]
```

`<proto>` This argument is not used for **VP Designer**.

For **form_edit**, this argument specifies the protocol of the atom to execute. The possible protocol names are prefixed **prot$** and are defined on Stratus in **arclient>atoms_dir>protocols.file**. Some typical values are as follows:

prot$uni — Universal Protocol

prot$display — Display Manager Protocol

prot$action — Action Protocol

prot$extract — Data Manager Protocol

prot$buffer — Buffer Protocol

prot$database — Database Manager Protocol

prot$xfer — File Transfer Protocol

prot$file — File Manager Protocol

prot$list — List Manager Protocol

prot$code — Code Manager Protocol

	prot$chat — Chat Protocol
	prot$var — Variable Protocol
	prot$async — Async Protocol
	prot$shorthand — Shorthand Protocol
	prot$if — IF Protocol
	prot$mat — Display Attributes Protocol
	prot$mip — Message Interchange Protocol
	prot$reg — Registration Protocol
	prot$mmi — Multimedia Interface Protocol
	prot$imgxfer — Image Transfer Protocol
	prot$image — Image Manager Protocol
	prot$chart — Chart Protocol
	prot$morg — Multimedia Organizer Protocol
	prot$rich — Rich Protocol
	prot$exapi — External API Protocol
	prot$ccl — CCL Protocol
	prot$p3 — P3 Protocol
	prot$pakman — File Decompression Protocol
	prot$address — Address Protocol
	prot$mt — Debugging Protocol
`<atom>`	Specifies the target atom to execute using a return value (raw data) from a preceding atom as its argument.
`[<data>]`	Specifies optional data.

Return Value

Return value resulting from the execution of the specified atom.

Example

The following example appends data base record 32-12-345 with the data saved from a previous operation provided that the data is not 0:

```
atom$uni_start_stream
.
.
.
atom$man_set_context_relative <56>
atom$var_lookup_by_id <A, 123>
atom$var_data_get <A>
.
.
.
atom$if_data_neq_zero_then <10>
atom$idb_set_context <32-12-345>
atom$uni_use_last_atom_data <atom$idb_append_data>
atom$idb_end_context
atom$man_end_context
atom$uni_sync_skip <10>
.
.
.
atom$uni_end_stream
```

atom$uni_use_last_atom_string
10 ($0A)

Description

atom$uni_use_last_atom_string executes the specified atom using the string value returned by the previously executed atom.

Syntax for VP Designer

```
atom$uni_use_last_atom_string  <atom[, data]>
```

Syntax for form_edit

```
atom$uni_use_last_atom_string  <proto> <atom> [<data>]
```

`<proto>` This argument is not used for **VP Designer**.

For **form_edit**, this argument specifies a protocol name. The possible protocol names are prefixed **prot$** and are defined on Stratus in **arclient>atoms_dir>protocols.file**. Some typical values are as follows:

prot$uni — Universal Protocol

prot$display — Display Manager Protocol

prot$action — Action Protocol

prot$extract — Data Manager Protocol

prot$buffer — Buffer Protocol

prot$database — Database Manager Protocol

prot$xfer — File Transfer Protocol

prot$file — File Manager Protocol

prot$list — List Manager Protocol

prot$code — Code Manager Protocol

	prot$chat — Chat Protocol
	prot$var — Variable Protocol
	prot$async — Async Protocol
	prot$shorthand — Shorthand Protocol
	prot$if — IF Protocol
	prot$mat — Display Attributes Protocol
	prot$mip — Message Interchange Protocol
	prot$reg — Registration Protocol
	prot$mmi — Multimedia Interface Protocol
	prot$imgxfer — Image Transfer Protocol
	prot$image — Image Manager Protocol
	prot$chart — Chart Protocol
	prot$morg — Multimedia Organizer Protocol
	prot$rich — Rich Protocol
	prot$exapi — External API Protocol
	prot$ccl — CCL Protocol
	prot$p3 — P3 Protocol
	prot$pakman — File Decompression Protocol
	prot$address — Address Protocol
	prot$mt — Debugging Protocol
`<atom>`	Specifies an atom to execute using a return value (string) from a preceding atom as its argument.
`[<data>]`	Specifies optional data.

Return Value

Result of the processed atom.

Example

The following example executes the **atom$man_replace_data** atom that replaces a string to object 11, which resulted from the data conversion by the previously executed atom:

```
atom$uni_start_stream
.
.
.
atom$man_set_context_relative <56>
atom$var_lookup_by_id <A, 23>
atom$var_number_get <A>
atom$uni_save_result
.
.
.
atom$man_change_context_relative <11>
atom$uni_get_result
atom$uni_convert_last_atom_data
atom$uni_use_last_atom_string <atom$man_replace_data>
atom$man_end_context
.
.
atom$man_update_display
atom$uni_end_stream
```


atom$uni_use_last_atom_value
11 ($0B)

Description

atom$uni_use_last_atom_value executes the specified atom using the numeric value returned by the previously executed atom.

Syntax for VP Designer

```
atom$uni_use_last_atom_value <atom[, data>]
```

Syntax for form_edit

```
atom$uni_use_last_atom_value <proto> <atom> [<data>]
```

`<proto>` This argument is not used for **VP Designer**.

For **form_edit**, this argument specifies a protocol name. The possible protocol names are prefixed **prot$** and are defined on Stratus in **arclient>atoms_dir>protocols.file**. Some typical values are as follows:

prot$uni — Universal Protocol

prot$display — Display Manager Protocol

prot$action — Action Protocol

prot$extract — Data Manager Protocol

prot$buffer — Buffer Protocol

prot$database — Database Manager Protocol

prot$xfer — File Transfer Protocol

prot$file — File Manager Protocol

prot$list — List Manager Protocol

prot$code — Code Manager Protocol

prot$chat — Chat Protocol

prot$var — Variable Protocol

prot$async — Async Protocol

prot$shorthand — Shorthand Protocol

prot$if — IF Protocol

prot$mat — Display Attributes Protocol

prot$mip — Message Interchange Protocol

prot$reg — Registration Protocol

prot$mmi — Multimedia Interface Protocol

prot$imgxfer — Image Transfer Protocol

prot$image — Image Manager Protocol

prot$chart — Chart Protocol

prot$morg — Multimedia Organizer Protocol

prot$rich — Rich Protocol

prot$exapi — External API Protocol

prot$ccl — CCL Protocol

prot$p3 — P3 Protocol

prot$pakman — File Decompression Protocol

prot$address — Address Protocol

prot$mt — Debugging Protocol

`<atom>`	Specifies an atom to execute using a return value (numeric data) from a preceding atom as its argument.
`[<data>]`	Specifies optional data.

Return Value

Result of the processed atom.

Example

The following example executes the **atom$var_number_set** atom that sets and saves a number in register A for object 11. The value saved resulted from the string conversion from the execution of the last atom.

```
atom$uni_start_stream
.
.
.
atom$man_set_context_relative <56>
atom$var_lookup_by_id <A, 23>
atom$var_string_get <A>
atom$uni_save_result
.
.
.
atom$man_change_context_relative <11>
atom$uni_get_result
atom$uni_convert_last_atom_string
➪atom$uni_use_last_atom_value <atom$var_number_set, 00x>
atom$var_number_save <00x, 07x>
atom$man_end_context
.
.
atom$man_update_display
atom$uni_end_stream
```

atom$uni_void
0 ($0)

Description

atom$uni_void does nothing. It is similar to a "NOP" instruction in many assembly languages. It is useful for leaving space for runtime patches, or as a visual marker to help stream authors identify sections of streams. It is also useful to force **form_edit** to show comment fields or, more specifically, a stream of atoms that were commented out with semicolons that would otherwise have been rejected from the output by **form_edit**.

Syntax

```
atom$uni_void
```

Return Value

None.

Example

The following example provides a visual bookmark in the code for the programmer:

```
atom$uni_start_stream
.
.
.
➪ atom$uni_void
        ;atom$if_numa_false_then <5>
        ;atom$uni_break
        ;atom$uni_sync_skip <5>
.
.
.
atom$uni_end_stream
```


atom$uni_wait_off
16 ($10)

Description

atom$uni_wait_off turns off the wait state.

Syntax

```
atom$uni_wait_off
```

Return Value

None.

Example

The following example turns off the wait state:

```
atom$uni_start_stream
.
.
.
atom$man_update_display
atom$man_end_context
➪ atom$uni_wait_off
atom$uni_end_steam
```

atom$uni_wait_off_end_stream
18 ($12)

Description

atom$uni_wait_off_end_stream turns the wait cursor off and terminates the current atom stream. Any buffered atoms associated with this stream are processed before the stream terminates.

Syntax

```
atom$uni_wait_off_end_stream
```

Return Value

None.

Example

The following example turns the wait cursor off and terminates the current stream:

```
atom$uni_start_stream
.
.
.
atom$man_update_display
atom$man_end_context
➪ atom$uni_wait_off_end_stream
```


atom$uni_wait_on
15 ($0F)

Description

atom$uni_wait_on activates the wait state in a member session, which changes the cursor to an hourglass (wait cursor). For every **atom$uni_wait_on**, a corresponding **atom$uni_wait_off** must occur.

Syntax

```
atom$uni_wait_on
```

Return Value

None.

Example

The following example activates the wait state in a stream and turns on the wait cursor in a member session:

```
  atom$uni_start_stream
➪ atom$uni_wait_on
  atom$sm_check_domain
  atom$de_start_extraction
  atom$de_validate <display_msg | terminate>
  .
  .
  .
  atom$uni_wait_off
  atom$uni_end_stream
```