uc4400 (view source)

Defined in: uf.h

Overview

Select the listing device. The listing device can be set to a file, the listing window, or both file and listing window. The listing device can also be set to no output.

Return

0 = Success
2 = Listing device is unavailable
otherwise an error code

Environment

Internal and External

See Also

UF_UI_open_listing_window
UF_UI_close_listing_window
UF_UI_write_listing_window

Required License(s)

gateway

```
int uc4400
(
    int listing_device,
    char * file_spec,
    int ip3
)
```

int	**listing_device**	Input	The listing device to use: = 0 standard listing window = 4 output to a file = 8 output to both file and listing window = 9 No output is written
char *	**file_spec**	Input	This is only used for output to a file. This is the file specification for the output file, and must be less than MAX_FSPEC_NCHARS characters.
int	**ip3**	Input	This is no longer used.

---

uc4403 (view source)

Defined in: uf.h

Overview

uc4403 write character string to listing device set by call to uc4400
if you only need to write to the listing window
(as opposed to a listing device set by uc4400)
this function may be replaced with UF_UI_write_listing_window

Required License(s)

gateway

```
int uc4403
(
    const char * cp1
)
```

const char *	**cp1**	Input

---

uc4404 (view source)

Defined in: uf.h

Overview

uc4404 write character string and integer to listing device set by uc4400
if you only need to write the data to the listing window
(and not a particular device set by uc4400)
this function may be replaced by UF_UI_write_listing_window

Required License(s)

gateway

```
int uc4404
(
    const char * cp1,
    int ip2
)
```

const char *	**cp1**	Input
int	**ip2**	

---

uc4406 (view source)

Defined in: uf.h

Overview

uc4406 write character string and real to listing device set by uc4400
if you only need to write the data to the listing window
(and not a particular listing device set by uc4400)
this function may be replaced by UF_UI_write_listing_window

Required License(s)

gateway

```
int uc4406
(
    const char * cp1,
    double rp2
```

)

const char *	**cp1**	Input
double	**rp2**	

---

uc4409 (view source)

Defined in: uf.h

Overview
Query the current disposition of the listing device.

Environment
Internal and External

Returns
0 = Listing device is set to the listing window
4 = Listing device is set to a file
8 = Listing device is set to both file and the listing window
9 = Listing device is set to NULL
otherwise the return is an error code.

See Also
UF_UI_open_listing_window
UF_UI_close_listing_window
UF_UI_write_listing_window
uc4400

Required License(s)
gateway

```
int uc4409
(
    char file_name [ MAX_FSPEC_BUFSIZE ] ,
    int * ir3
)
```

char	**file_name [MAX_FSPEC_BUFSIZE]**	Output	For return code 4 or 8, this is the file specification of the listing device file.
int *	**ir3**	Output	Unused.

---

uf3192 (view source)

Defined in: uf.h

Overview
Query the current NX Open program environment for the current image.

Note: This value is updated during initialization so it is set for

each image and won't remember the information from previous images.

Environment
Internal and External

Returns
Void

Required License(s)
gateway

void uf3192
(
int * ir1
)

int *	**ir1**	Output	The environment that the NX Open program is running in. 0 is External Open, 1 is Internal Open.

---

uf4401 (view source)

Defined in: uf.h

Overview
uf4401 opens the listing device set by uc4400 for output
if you only need to open the listing window for output
(and not a particular listing device set by uc4400)
this function may be replaced with UF_UI_open_listing_window

Required License(s)
gateway

void uf4401
(
int * ir1
)

int *	**ir1**		

---

uf4402 (view source)

Defined in: uf.h

Overview
uf4402 closes listing device set by uc4400
if you only need to close the listing window
(and not a particular device set by uc4400)
this function may be replaced by UF_UI_close_listing_window

Required License(s)

gateway

```
void uf4402
(
    void
)
```

UF_add_callback_function (view source)

Defined in: uf.h

Overview

Registers the callback to be called whenever the reason specified occurs within NX.

Please note the following:

Once a routine is registered as a callback, the shared library it resides in should not be unloaded until it is deregistered.

The callback function must call UF_initialize and UF_terminate, if it makes any Open C function calls

The callback function receives three arguments:
argument 1 - The reason the callback has been called
argument 2 - A pointer to the part tag that caused the callback.
argument 3 - A user supplied pointer. Typically this will be a structure pointer of data that the callback function will need.

The callback will be called towards the end of the processing for that particular reason. For example, the callback registered against UF_create_part_reason will be called when a new part is created, after almost all the process of creating this new part is completed.

The only exception to the above is UF_close_part_reason, for which the callback will be called at the beginning of closing the part.

NOTE: For the new and open part reasons the callback is run at a time when the newly created or opened part has not been set as the work part. Also operations such as create part, open part, save part or change display part should not be called from callbacks fired by any of these same operations.

Environment

Internal and External

See Also

UF_remove_callback_function
UF_callback_reason_e
Refer to the example of setting up a callback.

History

V16 was changed to require the callback to call UF_initialize

Required License(s)

gateway

```
int UF_add_callback_function
(
    UF_callback_reason_e_t reason,
    UF_callback_fn_t fn,
    void * user_data,
    UF_registered_fn_p_t * function_id
)
```

UF_callback_reason_e_t	**reason**	Input	Reason to call callback
UF_callback_fn_t	**fn**	Input	Function to call. Note this function must call UF_initialize if it makes any Open C function calls. It should also call UF_terminate.
void *	**user_data**	Input	User data. This pointer will be passed to your callback function as the third argument. It will typically be a structure pointer with information that your callback function will require.
UF_registered_fn_p_t *	**function_id**	Output	Identifier to use to remove callback

---

UF_allocate_memory (view source)

Defined in: uf.h

Overview
Allocates a block of storage and returns a pointer to it. Use UF_free to free the allocated memory when no longer required.

Return
void - a pointer to the block of storage, aligned so that any data type can be stored in it.
when error occur or nbytes = 0, a NULL pointer is returned

Environment
Internal and External

History
This function was originally released in V15.0

Required License(s)
gateway

```
void * UF_allocate_memory
(
    unsigned int nbytes,
    int * error_code
)
```

unsigned int	**nbytes**	Input	The number of bytes to be allocated. If 0, NULL is returned

int *	**error_code**	Output	NX standard error code 0 success !=0 failure

---

UF_ask_application_module (view source)

Defined in: uf.h

Overview

Returns the current application module. The value that is returned maps to a string defined constant. You can find the string in the header file uf.h. The strings are of the form UF_APP_<module>. For example, UF_APP_DRAFTING represents the drafting module.

Note - if this function is called from an external program it will always return a FALSE and an app id of APP_NONE.

Environment

Internal and External

Required License(s)

gateway

```
int UF_ask_application_module
(
    int * module_id
)
```

int *	**module_id**	Output	The module identification value of the current application.

UF_ask_codeset (view source)

Defined in: uf.h

Overview

Returns the code set identifier supported by a machine. The code set identifier can be any one of the enumerated constants in UF_codeset_e.

Environment

Internal and External

See Also

UF_codeset_t

Required License(s)

gateway

```
int UF_ask_codeset
(
```

```
    UF_codeset_t * codeset
)
```

UF_codeset_t *	**codeset**	Output	Pointer to the codeset identifier.

UF_ask_grip_args (view source)

Defined in: uf_ugopenint.h

Overview

This routine is used to receive arguments from a grip program. The grip program sets up these arguments with the GRIP command, GRARGS. The GRIP program then invokes execution of the internal Open API program with the GRIP XSPAWN command. The Open API program, during execution, maintains complete control of the system including any interactive statements or error messages.

To receive the arguments from GRIP, define the structure, UF_args_s, that describes the arguments to share with the GRIP program. This definition must match the definition supplied to the GRIP GRARGS command so that the contents of the GRIP variables can be copied to the contents of the Open API variables.

Please note the following limitations:
. The GRARGS command is limited to 103 arguments.
. The size of character strings input to UF_ask_grip_args should be declared to the largest GRIP string (132 characters) plus a null character or 133 characters.
. Calls cannot be nested. For example, a sequence where GRIP XPAWNs an internal Open API program which uses UF_call_grip to execute GRIP, and anything similar, is not allowed.

Environment

Internal

See Also

See the example
which is called by this grip program.

Required License(s)

gateway

```
int UF_ask_grip_args
(
    int argument_count,
    UF_args_p_t gruf_arg_list
)
```

int	**argument_count**	Input	Count of arguments from GRIP executable
UF_args_p_t	**gruf_arg_list**	Input / Output	The data arguments from GRIP. An array of structures where each element in the array is a structure that contains an arguments type, size, and address. The actual values will be modified by this call.

UF_ask_syslog_filename (view source)

Defined in: uf.h

Overview
Get name of the current syslog

Environment
Internal and External

History
Released in NX2.0.5

Required License(s)
gateway

```
int UF_ask_syslog_filename
(
    char * * filename
)
```

char * *	**filename**	Output to UF_*free*	pointer to dynamically allocated path filename must be freed with UF_free()

UF_ask_system_info (view source)

Defined in: uf.h

Overview
Get the information about an Open C Program

This routine can be used to gather the system information of an Open C program, like the machine name, Operating System information etc., and is useful in benchmarking, where you want to record the information about the machine the benchmark was run on.

This routine outputs the data through the structure UF_system_info_t.

Once the data is no longer needed, the memory allocated to the fields of this structure should be freed by a corresponding call to the routine UF_free_system_info.

Environment
Internal and External

See Also
UF_free_system_info
UF_system_info_t

History
Released in V18.0, modified in NX 6.0 to return used bundles

Required License(s)
gateway

int UF_ask_system_info
(
UF_system_info_p_t info
)

UF_system_info_p_t	**info**	Output to UF_*free*	structure to hold the process information. The allocated fields should be freed by calling UF_free_system_info

---

UF_begin_timer (view source)

Defined in: uf.h

Overview
Start a timer.

This routine can be used to gather timing information about an Open C program, and is useful in benchmarking. This routine will start a timer. The timer will be stopped with a corresponding call to UF_end_timer.

More than one timer can be active at any time, this is useful when you have one overall timer, and then several sub-timers timing individual sub-operations.

Environment
Internal and External

See Also
UF_end_timer
UF_timer_values_t

History
Released in V18.0

Required License(s)
gateway

int UF_begin_timer
(
UF_timer_p_t timer
)

UF_timer_p_t	**timer**	Output	Timer id that is created

UF_call_grip (view source)

Defined in: uf.h

Overview

Calls and executes a GRIP program. Your GRIP program must be written, compiled, linked, and ready to run.

While a GRIP program executes, it can have access to an Open C API function arguments and their contents. Argument passing to the GRIP program occurs through UFARGS. After GRIP program termination, the Open C API program has access to any assignments made by the GRIP program. The GRIP program completes its execution and then UF_call_grip outputs a return value.

Note: If the path is omitted in the grip_executable string passed to UF_call_grip, NX will look in the current directory for the grip program. It will not look in the directory of the ufunc program or use any environment variables, including UGII_INITIAL_GRIP_DIR, to search for the GRIP program.

UF_translate_variable can be used to extract an environment variable and concatenate it with the program name.

The following GRIP commands are not supported in External mode when calling GRIP:
ALL OF GRIP NC
PLNOTE EDTXHT CHAIN
BLANK UNBLNK RVBLNK
If any of these commands are encountered, an error is written to the batch log file.

Environment

Internal and External

See Also

See the example which calls the grip program.

Required License(s)

gateway

```
int UF_call_grip
(
    char * grip_executable,
    int count,
    UF_args_p_t UFARGS
)
```

char *	**grip_executable**	Input	Name of GRIP program to execute (file name or full path name).
int	**count**	Input	Count of arguments to pass to GRIP executable
UF_args_p_t	**UFARGS**	Input	An array of structures where each element in the array is a structure that contains an arguments type, size, and address. Note that if an argument is type UF_TYPE_CHAR, the array must be initialized prior to the UF_call_grip call.

UF_decode_auth_file (view source)

Defined in: uf.h

Overview
Decodes an input authorization file using an integer key value that matches the original key for perturbation.

NOTE: See %UGII_BASE_DIR%\UGALLIANCE\alliance_readme.txt for more information

Environment
Internal

Required License(s)
gateway

```
int UF_decode_auth_file
(
    int key,
    char * input_file,
    void * * out_mem
)
```

int	**key**	Input	Integer valued key for decoding the input file
char *	**input_file**	Input	Input authorization file to decode Use the leaf name of a file located under the folder %UGALLIANCE%\auth_files. Entering the full path will return an error of -5 "Invalid file name".
void * *	**out_mem**	Output to UF_*free*	Pointer to decoded contents of the authorization file. Use UF_free to deallocate the memory.

UF_encode_auth_file (view source)

Defined in: uf.h

Overview
Encodes an input authorization file using an integer value for perturbation.

Environment
Internal

Required License(s)
gateway

```
int UF_encode_auth_file
(
    int key,
    char * input_file,
    char * output_file
)
```

int	**key**	Input	Value used to perturb the input file
char *	**input_file**	Input	Input authorization file name
char *	**output_file**	Input	Output encoded authorization file name

---

UF_end_timer (view source)

Defined in: uf.h

Overview

Stop the timer and get the timer values

This routine is used to gather timing information about an Open C program, and is useful in benchmarking. This routine will stop the timer created by a corresponding UF_begin_timer call, and output the timing information in the structure UF_timer_values_t.

Environment

Internal and External

See Also

UF_begin_timer
UF_timer_values_t

History

Released in V18.0

Required License(s)

gateway

```
int UF_end_timer
(
    UF_timer_t timer,
    UF_timer_values_p_t values
)
```

UF_timer_t	**timer**	Input	Timer to be stopped
UF_timer_values_p_t	**values**	Output	Timer values when it is stopped

---

UF_find_all_subdirectories (view source)

Defined in: uf.h

Overview

Find all subdirectories of a given name in the directories specified in the file custom_dirs.dat and ug_custom_dirs.dat. Since the same subdirectory may be in multiple directories, an array of path names will be returned. As an example searching for the subdirectory "startup"

will return almost all of the directories, since this is a standard subdirectory.

Environment
Internal and External

See Also
UF_find_file

History
Released in NX 1.0

Required License(s)
gateway

```
int UF_find_all_subdirectories
(
    const char * subdirectory,
    int * num_found,
    char * * * path
)
```

const char *	**subdirectory**	Input	The subdirectory to look for.
int *	**num_found**	Output	The number of subdirectories found.
char * * *	**path**	Output to UF_*free*	The list of subdirectories found. This must be freed by calling UF_free_string_array.

UF_find_file (view source)

Defined in: uf.h

Overview
Find a given subdirectory and file. This routine will traverse all the directories specified in the file custom_dirs.dat and ug_custom_dirs.dat looking for the file specified by the given subdirectory and filename. This routine will return the first file found.

Environment
Internal and External

See Also
UF_find_all_subdirectories

History
Released in NX 1.0

Required License(s)
gateway

```
int UF_find_file
(
    const char * subdirectory,
    const char * filename,
```

char * * path
)

const char *	**subdirectory**	Input	The subdirectory in which to look for the given file.
const char *	**filename**	Input	The filename to look for, including any extension.
char * *	**path**	Output to UF_*free*	The filename found. If no file is found, path will be NULL. The path must be freed by calling UF_free.

UF_free (view source)

Defined in: uf.h

Overview
Frees the memory allocated and returned by an Open C API routine.

Return
void

Environment
Internal and External

Required License(s)
gateway

void UF_free
(
void * data
)

void *	**data**	Input	Pointer to data item to free

UF_free_string_array (view source)

Defined in: uf.h

Overview
Frees the memory allocated for an array of pointers and frees the strings within the array. You must pass in the number of pointer in the array. Various Open C API routines malloc space for an array of pointers, and then allocate space for each of the pointers.

Return
void

Environment
Internal and External

Required License(s)

gateway

void UF_free_string_array
(
int n_strings,
char * * string_array
)

int	**n_strings**	Input	number of pointers
char * *	**string_array**	Input	n_strings pointer to the array of pointers to free

---

UF_free_system_info (view source)

Defined in: uf.h

Overview

Free the fields of the structure UF_system_info_t, containing the process information

This structure is filled earlier by a call to the routine UF_ask_system_info

Environment

Internal and External

See Also

UF_ask_system_info
UF_system_info_t

History

Released in V18.0

Required License(s)

gateway

int UF_free_system_info
(
UF_system_info_p_t info
)

UF_system_info_p_t	**info**	Input	structure whose fields are to be freed

---

UF_get_customer_default (view source)

Defined in: uf.h

Overview

This routine retrieves a customer default value from the database.
There are several different conditions that can arise from querying

the database:
1) The record was found and its units match. The function returns SUCCESS(0)
2) The record was not found. If units given was ENGLISH or METRIC nothing is returned and the function returns a FAILED(1). If the units given was NONE a second and/or third attempt is made with units ENGLISH THEN METRIC used instead. The results of these attempts can be any of the conditions outlined above.

If you specify either METRIC or ENGLISH for units, this function automatically appends _MU or _EU to the input name. For example, if you specify METRIC and input the name "UG_gridSpacingY", the function searches for "UG_gridSpacingY_MU".

If you specify NONE for units, the function searches for the name as specified. If the name is not found it searches again with _EU appended to the name. If the name is still not found, then it appends _MU to the name for its final search.

Return
return code:
0 = Success
1 = Record not found
<0 or >1 = Use UF_get_fail_message to get error.

Environment
Internal and External

See Also
See the example

Required License(s)
gateway

```
int UF_get_customer_default
(
    char * name,
    int units,
    char * * default_value
)
```

char *	**name**	Input	Name of Module ID and default
int	**units**	Input	Units are enumerated as follows: 0 = NONE 1 = METRIC 2 = ENGLISH
char * *	**default_value**	Output to UF_*free*	String containing default value. If default_value is NULL then the value was not found or an error occurred. Use UF_free to deallocate memory when no longer required.

UF_get_fail_message (view source)

Defined in: uf.h

Overview
Returns the message string associated with an error code.

Environment
Internal and External

Required License(s)
gateway

```
int UF_get_fail_message
(
    int fail_code,
    char message [ 133 ]
)
```

int	**fail_code**	Input	the failure code to translate
char	**message [133]**	Output	the message corresponding to the fail code

---

UF_get_release (view source)

Defined in: uf.h

Overview
Returns the current major release version number. For example, if you call this function from V10.4.1, it returns V10.4. You must use UF_free to deallocate space used by release.

Environment
Internal and External

Required License(s)
gateway

```
int UF_get_release
(
    char * * release
)
```

char * *	**release**	Output to UF_*free*	Major release version number

---

UF_get_reserved_licenses (view source)

Defined in: uf.h

Overview
Returns the licenses that are currently reserved against a context.
This method will also output the licenses reserved against the default context in the syslog.

Environment

Internal and External

Required License(s)

gateway

```
int UF_get_reserved_licenses
(
    const char* context_name,
    int * n_licenses,
    char* * * licenses
)
```

const char*	**context_name**	Input	Name of context to release against. Will use the default context if NULL is specified.
int *	**n_licenses**	Output	The number of licenses.
char* * *	**licenses**	Output to UF_*free*	The list of license reserved against the specified context. This must be freed by calling UF_free_string_array.

UF_get_translated_fail_message (view source)

Defined in: uf.h

Overview

Returns the message string associated with an error code in current user interface language.

Environment

Internal and External

History

Originally released in NX 10.0

Required License(s)

gateway

```
int UF_get_translated_fail_message
(
    int fail_code,
    char * * message
)
```

int	**fail_code**	Input	the failure code to translate
char * *	**message**	Output	the message corresponding to the fail code

---

UF_initialize (view source)

Defined in: uf.h

Overview

Initializes the Open C API environment and allocates an Open C API execute license. If you do not call UF_initialize, then subsequent calls to Open C API fail with an error code of UF_err_program_not_initialized. This function must be the first function you call after you have declared your variables.

In external mode, you may only call UF_initialize once. Once UF_terminate is called, there is no way to reconnect to the Open C environment.

Programs need to call UF_terminate before exiting to ensure licenses are released and avoid errors in downstream NXOpen programs.

Environment

Internal and External

See Also

Refer to the UGII_UGOPEN_SESSION variable

Required License(s)

gateway

```
int UF_initialize
(
    void
)
```

UF_initialize_dm (view source)

Defined in: uf.h

Overview

Same as UF_initialize with the addition that Direct Model is also initialized.

Environment

Internal and External

Required License(s)

gateway

```
int UF_initialize_dm
(
    void
)
```

UF_is_initialized (view source)

Defined in: uf.h

Overview
Checks if the Open C API environment has been successfully initialized.

Return
return code:
0 = Open C API is not initialized.
1 = Open C API has been properly initialized.

Environment
Internal and External

Required License(s)
gateway

```
int UF_is_initialized
(
    void
)
```

UF_is_ugmanager_active (view source)

Defined in: uf.h

Overview
Checks to see if an NX Manager process is active. It returns true if NX Manager is active.

Environment
Internal and External

Required License(s)
gateway

```
int UF_is_ugmanager_active
(
    logical * is_active
)
```

logical *	**is_active**	Output	Returns true if NX Manager is active

---

UF_load_library (view source)

Defined in: uf.h

Overview
Loads the shared library specified by library_name and finds the entry point specified by symbol_name in that library. The reference to the specified entry point is output as a function pointer declared to accept

and return void (i.e. type UF_load_f_p_t).

For the entry point to be found on Windows NT, you must declare it with the DllExport macro, e.g.

extern DllExport int my_function(...

without the DllExport, the name will not be exported. Since the DllExport also works on Unix systems, the best practice is just to always use the DllExport macro for routines you want exported.

If library_name is found but symbol_name is not, an appropriate error code returns, but the loaded library may not be unloaded. The user should call UF_unload_library to explicitly unload library_name on failure, if appropriate.

To load a shared library into an external application - the library must not be linked against any interactive libraries. This can be accomplished by using the "-noui" option on Unix.

NOTE: UF_load_library is intended for function (i.e. "text" entry point) lookup only. It should NOT be used for data symbol lookup.

Environment
Internal and External

See Also
UF_unload_library
The UF_load_library example is split into an, internal program which is loaded by the external program.

Required License(s)
gateway

```
int UF_load_library
(
    const char * library_name,
    const char * symbol_name,
    UF_load_f_p_t * function_ptr
)
```

const char *	**library_name**	Input	Name of shared library to load.
const char *	**symbol_name**	Input	Name of symbol (function) to find.
UF_load_f_p_t *	**function_ptr**	Output	Pointer to symbol (function) in library.

UF_MISC_set_program_name (view source)

Defined in: uf.h

Overview
Each part file has an internal history storage area where the program name can be stored when the system saves the part file. This function enables you to specify either the entire program name (External mode) or append a name to the program name (Internal mode). You should call this routine before you call UF_PART_open since it does

not set the program name on parts that are already opened. A utility called ug_inspect enables you to read the program name stored in the history area.

Interactively (Internal mode), when you are in NX, the program name will take the version of NX currently being run (e.g. NX9.0.0.19) and append the string being passed into UF_MISC_set_program_name. For example, if you specify "XYZ version 9.0" as the name string to this routine, then the program name in the history would look like:
NX 9.0.0.19XYZ version 9.0
To separate the version and string, put a leading space in the name being passed into this routine. " XYZ version 9.0" passed in will create:
NX 9.0.0.19 XYZ version 9.0

Once UF_terminate is called, the appended string is removed and it will default back to the NX version number the next time the part is saved.

In External mode, the program name is what you specify in the name string. From our previous example this would be XYZ version 9.0. The limitation for the length of the name string depends on which mode you are in, Internal or External. The maximum length of the entire program name is 128 characters. In Internal mode, because the name string is concatenated to the program name you are limited to 64 characters. In External mode, since the name string you specify becomes the program name you are limited to 128 characters. In either mode, if you specify a name that is to long, the name is truncated to either 64 or 128 characters (Internal and External mode respectively).

The internal history area is like a log file. If you resave the part file with a different program a new history entry is added to the part file. That is, the history grows larger. Thus you are likely to see two lines of history if you have saved the part file twice and specified a name with each save.

Return
void

Environment
Internal and External

Required License(s)
gateway

```
void UF_MISC_set_program_name
(
    const char * name
)
```

const char *	**name**	Input	Name of program to store in the internal history

---

UF_print_syslog (view source)

Defined in: uf.h

Overview

Output a message to the syslog.

This message can be used to print run time information about the NX Open program to the syslog and is useful for state checking, validation and debugging.

Printing the trace is especially useful while debugging and reporting a problem.

Environment
Internal and External

History
Released in V18.0

Required License(s)
gateway

int UF_print_syslog
(
char * message,
logical trace
)

char *	**message**	Input	Message to be output to the syslog
logical	**trace**	Input	TRUE = Print the stack trace along with the message FALSE = Print only the message

---

UF_reallocate_memory (view source)

Defined in: uf.h

Overview
Reallocates a block of storage and returns a pointer to it. Use UF_free to free the allocated memory when no longer required.

Return
void - a pointer to the block of storage, aligned so that any data type can be stored in it.
when error occur or nbytes = 0, a NULL pointer is returned

Environment
Internal and External

History
This function was originally released in V16.0

Required License(s)
gateway

void * UF_reallocate_memory
(

```
    void * data,
    unsigned int nbytes,
    int * error_code
)
```

void *	**data**	Input	Pointer to previously allocated memory
unsigned int	**nbytes**	Input	The number of bytes to be allocated. If 0, NULL is returned
int *	**error_code**	Output	NX standard error code

UF_release (view source)

Defined in: uf.h

Overview
Release a license against the specified license context.

Environment
Internal and External

Required License(s)
gateway

```
int UF_release
(
    const char* license,
    const char* context_name
)
```

const char*	**license**	Input	License feature name to be released
const char*	**context_name**	Input	Name of context to release license against. Will use the default context if NULL is specified.

UF_release_all (view source)

Defined in: uf.h

Overview
Release all licenses against the specified license context. If the context specified is a user-defined context it will be deleted.

Environment
Internal and External

Required License(s)
gateway

int UF_release_all
(
const char* context_name
)

const char*	**context_name**	Input	Name of context release to release. Will use the default context if NULL is specified.

UF_remove_callback_function (view source)

Defined in: uf.h

Overview
Deregisters the callback specified by the identifier.

Environment
Internal and External

See Also
UF_add_callback_function

Required License(s)
gateway

int UF_remove_callback_function
(
UF_registered_fn_p_t function_id
)

UF_registered_fn_p_t	**function_id**	Input	Identifier from UF_add_callback_function

---

UF_reserve (view source)

Defined in: uf.h

Overview
Reserve a license against the specified license context.
If the specified user-defined context does not already exist it will be created.

Environment
Internal and External

Required License(s)
gateway

int UF_reserve
(

```
    const char* license,
    const char* context_name
)
```

const char*	**license**	Input	License feature name to be reserved
const char*	**context_name**	Input	Name of context to reserve license against. Will use the default context if NULL is specified.

UF_set_grip_args (view source)

Defined in: uf_ugopenint.h

Overview

Updates the contents of the matching GRIP argument list with the contents of the Open API addresses. This is used to return data to the GRIP program.

Environment

Internal

Required License(s)

gateway

```
int UF_set_grip_args
(
    int argument_count,
    UF_args_p_t gruf_arg_list
)
```

int	**argument_count**	Input	Count of arguments to set in GRIP.
UF_args_p_t	**gruf_arg_list**	Input	The arguments to set in GRIP. An array of structures where each element in the array is a structure that contains an arguments type, size, and address.

UF_set_retiring_flag (view source)

Defined in: uf.h

Overview

Set a flag that determines if an Open C executable will issue warnings when a function that is scheduled to be retired is called. This can be used to alert the NX Open programmer that modification to their program is required.

When routines are planned to be removed from the NX Open interface, the prototype will be moved to uf_retiring.h, along with a description of what the NX Open programmer should do. A warning can be issued when any of these routines are called by an NX Open application. This routine

allows the NX Open programmer to set the value of this flag.

This routine overrides the environment variable UGII_USING_RETIRING_FUNCTIONS which can be set by the user to get the same warnings. This routine is meant to be used when the programmer wants to force a particular value for a program.

Environment
Internal and External

History
Released in V16.0

Required License(s)
gateway

```
int UF_set_retiring_flag
(
    int value
)
```

int	**value**	Input	0 = Don't warn user at all, this is the default 1 = Output a single message if any function is called that is planned to be obsoleted 2 = Output a message for each function called that is planned to be obsoleted. The message will include the routine name, and the number of times called.

UF_set_variable (view source)

Defined in: uf.h

Overview
Sets an environment variable to the string specified in value. You must supply the complete name of variable.

Environment
Internal and External

See Also
See the example

Required License(s)
gateway

```
int UF_set_variable
(
    const char * variable,
    const char * value
)
```

const char *	**variable**	Input	Environment Variable to set in Operating System
const char *	**value**	Input	Value or equivalence string for the variable.

UF_sys_exit (view source)

Defined in: uf.h

Overview

Force NX to shut down immediately. This is a substitute for the standard c function: exit(0)

Warning: depending on the current state of NX, forcing a shutdown may produce warnings/errors as NX attempts to close itself.

Environment

Internal and External

History

NX 6.0.1

Required License(s)

gateway

```
void UF_sys_exit
(
    void
)
```

UF_TAG_ask_handle_from_tag (view source)

Defined in: uf.h

Overview

This function replaces UF_TAG_ask_handle_of_tag.

Return the persistent HANDLE for an object tag.

Multiple parts can be loaded into NX during a single session, and the Tag (i.e. Object Identifier) cannot be guaranteed between sessions. Consequently, "handles" were created to identify objects between sessions. This handle is persistent between sessions once the object is saved to a part file. A handle of a new object in this session which is never saved is not guaranteed to stay the same until the object is saved. Recreating the same object in a new session may not be the same even if seemingly identical steps were taken to create the object. This is particularly true across versions of software.

Handles should not be compared directly for checking identity as they contain version information. Handles should be broken apart using UF_TAG_decompose_handle and the file_data strings compared and the sub_file_id integers compared but the version should be ignored.

Operations which change the name of the part will change the handle as well - so "Save As" will cause handles to change.

This function will return an error when the part data of the handle contains a character that cannot be represented. For this reason

it is suggested that UF_TEXT_set_text_mode(UF_TEXT_ALL_UTF8) be set when you are working with handles that may come from parts that have characters that are outside of your locale (e.g. parts with Japanese characters on an english system).

Output:
The name of handle for object tag. This string must be freed by calling UF_free.

Environment
Internal and External

See Also
UF_TAG_ask_tag_of_handle

History
Originally released in NX 10.0

Required License(s)
gateway

int UF_TAG_ask_handle_from_tag
(
tag_t object_tag,
char * * handle
)

tag_t	**object_tag**	Input	tag of object to query for persistent handle
char * *	**handle**	Output to UF_*free*	Output handle for the tag.

UF_TAG_ask_new_tag_of_entity (view source)

Defined in: uf.h

Overview
Returns the new tag of the object corresponding to the known object ID in the V9 part. This only works for the Object IDs of the last V9 part loaded into this session.
NOTE: This was renamed from UF_ask_new_tag_of_entity()

Return
New tag

Environment
Internal and External

See Also
UF_TAG_ask_handle_from_tag

Required License(s)
gateway

tag_t UF_TAG_ask_new_tag_of_entity
(
tag_t v9_eid

)

tag_t	**v9_eid**	Input	Known Object ID (EID) of V9 part

UF_TAG_ask_tag_of_handle (view source)

Defined in: uf.h

Overview
Return the object tag of a persistent HANDLE.

Return
Returns object tag of object handle

Environment
Internal and External

See Also
UF_TAG_ask_handle_from_tag

Required License(s)
gateway

tag_t UF_TAG_ask_tag_of_handle
(
char * object_handle
)

char *	**object_handle**	Input	Name of persistent handle

UF_TAG_compose_handle (view source)

Defined in: uf.h

Overview
Given the constituent parts of data manager, filename, subfile id and version, this function returns the resulting handle.

Environment
Internal and External

Required License(s)
gateway

int UF_TAG_compose_handle
(
const char * file_data,
unsigned int sub_file_id,
unsigned int version,
char * * handle

)

const char *	**file_data**	Input	Data about the file. This string would come from a call to UF_TAG_decompose_handle.
unsigned int	**sub_file_id**	Input	sub_file identifier. The sub_file is an internal file in the part file database.
unsigned int	**version**	Input	Version of part
char * *	**handle**	Output to UF_*free*	String containing the composed handle

---

UF_TAG_decompose_handle (view source)

Defined in: uf.h

Overview

Given the handle, the constituent parts of data manager, filename, subfile id and version are returned.

Note this function will return an error if you have UF_TEXT_set_text_mode() set to locale and the file that the data is coming from has characters outside of your locale. It is suggested that you set your text mode to UF_TEXT_ALL_UTF8 when you are working with handles.

Environment

Internal and External

See Also

UF_TAG_ask_new_tag_of_entity
UF_TAG_ask_handle_from_tag

Required License(s)

gateway

int UF_TAG_decompose_handle
(
char * handle,
char * * file_data,
unsigned int * sub_file_id,
unsigned int * version
)

char *	**handle**	Input	Handle to decompose
char * *	**file_data**	Output to UF_*free*	Data about the file. This can be used by UF_TAG_compose_handle. This argument must be freed by calling UF_free.
unsigned int *	**sub_file_id**	Output	Sub-file identifier for handle. The sub_file is an internal file in the part file database.
unsigned int *	**version**	Output	Version of part in which handle is located.

UF_TAG_register_event_cb (view source)

Defined in: uf.h

Overview

Register a new tag event callback. All callbacks are called on all tags for all events. This returns an identifier for the callback which is used to allow it removal.

Please note the following items:
. The routines registered can not call any other Open C routines In particular, the tag cannot be queried or edited in any way.
. The implementation of any routine so registered must be very efficient as it is called many times.
. Recall that tags get reused. This means that the order of events is important so the event is interpreted with respect to the right "incarnation" of the tag.

Environment

Internal and External

See Also

UF_TAG_unregister_event_cb

History

This function was originally released in V15.0.

Required License(s)

gateway

```
int UF_TAG_register_event_cb
(
    UF_TAG_event_fn_t callback,
    void * closure,
    int * callback_id
)
```

UF_TAG_event_fn_t	**callback**	Input	Callback function
void *	**closure**	Input	Pointer passed to callback on each call
int *	**callback_id**	Output	Identifier for callback

UF_TAG_unregister_event_cb (view source)

Defined in: uf.h

Overview

Unregister a previously registered callback.

Environment

Internal and External

See Also
UF_TAG_register_event_cb

History
This function was originally released in V15.0.

Required License(s)
gateway

int UF_TAG_unregister_event_cb
(
int callback_id
)

int	**callback_id**	Input	- identifier of callback

UF_terminate (view source)

Defined in: uf.h

Overview
Terminates the Open C API environment and deallocates a Open C API execute license. If UF_terminate is not called, the Open C API execute license may not be freed. Once UF_terminate is called in an external Open C API program, the program will exit. You will not be able to call UF_initialize again and continue operation.

Calling this method will release all licenses in the default context.

Once UF_terminate has completed in an external program, you can no longer write to the console window. It must be reinitialized with system calls.

Environment
Internal and External

See Also
Refer to the UGII_UGOPEN_SESSION variable

Required License(s)
gateway

int UF_terminate
(
void
)

UF_translate_variable (view source)

Defined in: uf.h

Overview
Translates environment variables to their equivalence strings. You must supply the complete name of the variable argument. The pointer passed back points to an operating system string. This string must not be changed.

Environment
Internal and External

See Also
See the example

Required License(s)
gateway

```
int UF_translate_variable
(
    const char * variable,
    char * * translation
)
```

const char *	**variable**	Input	Environment Variable to Translate
char * *	**translation**	Output	Translated variable name or NULL if nonexistent.

UF_unload_library (view source)

Defined in: uf.h

Overview
Unloads the shared library specified by library_name. UF_unload_library unloads the library unconditionally, effectively invalidating all function pointers previously set by calls to UF_load_library for the specified library.

Environment
Internal and External

See Also
UF_load_library
The two UF_load_library example one and example two use UF_unload_library.

Required License(s)
gateway

```
int UF_unload_library
(
    char * library_name
)
```

char *	**library_name**	Input	Name of shared library to unload.

---

ufusr (view source)

Defined in: uf_ugopenint.h

Overview

Internal Open API Entry Point.

You use ufusr as a main function entry point for all Internal Open API programs. The ufusr function may also be used for user exits.

For internal programs written in C, a third argument needs to be passed into the "ufusr" function. This third variable is the length of the character array param.

Return

void

Environment

Internal

See Also

ufusr_ask_unload
See the discussion on user exits in uf_exit.h
Refer to the example of an internal program.

```
void ufusr
(
    char * param,
    int * retcod,
    int param_len
)
```

char *	**param**	Input	If this function is called from menuscript, this argument will contain the name of the menu button selected. Otherwise this argument is unused.
int *	**retcod**	Output	Return Code. Not used by Internal Open API.
int	**param_len**	Input	Length of 'param' Argument. NX handles this input for you.

---

ufusr_ask_unload (view source)

Defined in: uf_ugopenint.h

Overview

User supplied routine to specify how an Internal Open API shared image is handled upon exit. This function gives you the capability to unload an

internal Open API or user exit from NX. You can specify any one of the three constants as a return value to determine the type of unload to perform: immediately after Open API execution, via an unload selection dialog, or when NX terminates.

If you choose UF_UNLOAD_SEL_DIALOG, then you have the option to unload your image by selecting File--Utilities--Unload Shared Image.

Please note the following restrictions:
A program which uses UF_UI_append_menubar_menu should not use the option to unload an Open API image. Additionally, if your code which defines UF_UI_append_menubar_menu itself loads a shared library, this code should not attempt to unload the library. NX always makes a strong attempt to prevent unloading a library which was coded by using UF_UI_append_menubar_menu.

A shared library that contains registered functions, such as any of the User Defined Object registration methods, should not be unloaded. NX must keep the library loaded to access the address of the registered function.

Unload immediately and Unload via dialog should not be used if the program has registered any sort of callbacks.

Additionally, you have the option of coding the cleanup routine ufusr_cleanup to perform any house keeping chores that may need to be performed. If you code the cleanup routine it is automatically called by NX.

Return
UF_UNLOAD_IMMEDIATELY - unload immediately after Open API executes
UF_UNLOAD_SEL_DIALOG - via an unload selection dialog
UF_UNLOAD_UG_TERMINATE - when NX terminates

Environment
Internal

See Also
ufusr
See the example

```
int ufusr_ask_unload
(
    void
)
```

---

ufusr_cleanup (view source)

Defined in: uf_ugopenint.h

Overview
User supplied routine to clean up before the user function program is unloaded.

NX calls this entry point if it is defined in your source code. The entry point is called immediately before a shared library is unloaded and allows you to perform any cleanup tasks that must be

performed within NX. For example, if your internal program opened up a database program you could use the call to ufusr_cleanup as an opportunity to close any open database files before the image is unloaded.

NOTE: You use ufusr_cleanup in conjunction with ufusr_ask_unload. If you code ufusr_cleanup without defining ufusr_ask_unload, then the ufusr_cleanup entry point is ignored.

Return
void

Environment
Internal

See Also
See the example

void ufusr_cleanup
(
void
)
