## UF_UDOBJ_add_areas (view source)

**Defined in: uf_udobj.h**

### Overview

Adds area data to the convertible data area. Any number of doubles
can be added to the areas array. Each double represents a unit of
area. These calls are cumulative, if you make one call to add two
areas, and then a second call to add three areas, you will end up
with 5 areas on the UDO, and UF_UDOBJ_edit_areas would expect you
to pass in 5 elements.

### Environment

Internal and External

### See Also

UF_UDOBJ_edit_areas
UF_UDOBJ_delete_areas

### Required License(s)

gateway

```
int UF_UDOBJ_add_areas
(
    tag_t udo_tag,
    unsigned int num_areas,
    double * areas
)
```

| | | | |
|---|---|---|---|
| tag_t | **udo_tag** | Input | The tag for the UDO |
| unsigned int | **num_areas** | Input | The number of areas to add. |
| double * | **areas** | Input | The area data to add to the UDO. |

---

## UF_UDOBJ_add_doubles (view source)

**Defined in: uf_udobj.h**

### Overview

Adds doubles to the free form data area of an UDO. Any number of
valid double elements can be added. These calls are cumulative, if you
make one call to add two doubles, and then a second call to add three
doubles, you will end up with 5 doubles on the UDO, and UF_UDOBJ_edit_doubles
would expect you to pass in 5 elements.

### Environment

Internal and External

### See Also

UF_UDOBJ_edit_doubles
UF_UDOBJ_delete_doubles

### Required License(s)

gateway

```
int UF_UDOBJ_add_doubles
(
    tag_t udo_tag,
    unsigned int num_doubles,
    double * doubles
)
```

| | | | |
|---|---|---|---|
| tag_t | **udo_tag** | Input | The tag for the UDO |
| unsigned int | **num_doubles** | Input | The number of doubles to add. |
| double * | **doubles** | Input | The doubles to add to the UDO. |

---

## UF_UDOBJ_add_integers (view source)

**Defined in: uf_udobj.h**

### Overview
Adds integers to the free form data area of an UDO. Any number of
valid integer elements can be added. These calls are cumulative, if you
make one call to add two integers, and then a second call to add three
integers, you will end up with 5 integers on the UDO, and UF_UDOBJ_edit_integers
would expect you to pass in 5 elements.

### Environment
Internal and External

### See Also
UF_UDOBJ_edit_integers
UF_UDOBJ_delete_integers

### Required License(s)
gateway

```
int UF_UDOBJ_add_integers
(
    tag_t udo_tag,
    unsigned int num_ints,
    int * ints
)
```

| | | | |
|---|---|---|---|
| tag_t | **udo_tag** | Input | The tag for the UDO |
| unsigned int | **num_ints** | Input | The number of integers to add. |
| int * | **ints** | Input | The integers to add to the UDO. |

---

## UF_UDOBJ_add_lengths (view source)

**Defined in: uf_udobj.h**

### Overview

Adds length data to the convertible data area. Any number of doubles can be added to the lengths array. Each double represents a unit of length. These calls are cumulative, if you make one call to add two lengths, and then a second call to add three lengths, you will end up with 5 lengths on the UDO, and UF_UDOBJ_edit_lengths would expect you to pass in 5 elements.

### Environment

Internal and External

### See Also

UF_UDOBJ_edit_lengths
UF_UDOBJ_delete_lengths

### Required License(s)

gateway

```
int UF_UDOBJ_add_lengths
(
    tag_t udo_tag,
    unsigned int num_lengths,
    double * lengths
)
```

| | | | |
|---|---|---|---|
| tag_t | **udo_tag** | Input | The tag for the UDO |
| unsigned int | **num_lengths** | Input | The number of lengths to add. |
| double * | **lengths** | Input | The length data to add to the UDO. |

---

## UF_UDOBJ_add_links (view source)

**Defined in: uf_udobj.h**

### Overview

Adds a link to the specified NX objects defined by the link_defs[ ] array.
Links cannot be cyclical. Also, a type 2 link cannot be used to link UDOs to features, solid faces, or solid edges. You can only link to objects found in uf_object_types.h. These calls are cumulative, if you make one call to add two links, and then a second call to add three links, you will end up with 5 links on the UDO, and UF_UDOBJ_edit_links would expect you to pass in 5 elements.

The order in which NX objects are passed into this routine is NOT preserved. So, no assumptions should be made in this regard.

### Environment

Internal and External

### See Also

UF_UDOBJ_edit_links
UF_UDOBJ_delete_link
UF_UDOBJ_link_t

### Required License(s)

gateway

```
int UF_UDOBJ_add_links
(
    tag_t udo_tag,
    unsigned int num_links,
    UF_UDOBJ_link_t link_defs [ ]
)
```

| | | | |
|---|---|---|---|
| tag_t | **udo_tag** | Input | The tag for the UDO |
| unsigned int | **num_links** | Input | The number of objects to link to the UDO. |
| UF_UDOBJ_link_t | **link_defs [ ]** | Input | The link types and the associated NX objects to link to. |

---

## UF_UDOBJ_add_owning_links (view source)

**Defined in: uf_udobj.h**

### Overview

Adds objects by owning links to a UDO. These calls are cumulative, if you make one call to add two links, and then a second call to add three links, you will end up with 5 links on the UDO.

If the udo_tag is a UDO Feature, then the array of input tags can not include any solids, faces, edges, sheets or features. The array of tags should only be wireframe data.

The order in which NX objects are passed into this routine is NOT preserved. So, no assumptions should be made in this regard.

### Environment

Internal and External

### See Also

UF_UDOBJ_delete_owning_link

### Required License(s)

gateway

```
int UF_UDOBJ_add_owning_links
(
    tag_t udo_tag,
    unsigned int num_links,
    tag_t ug_objs [ ]
)
```

| | | | |
|---|---|---|---|
| tag_t | **udo_tag** | Input | The tag of the UDO to which to add owning links. |

| | | | |
|---|---|---|---|
| unsigned int | **num_links** | Input | The number of objects to link to the UDO by owning links. |
| tag_t | **ug_objs [ ]** | Input | An array of tags of objects to link to the UDO by owning links. |

## UF_UDOBJ_add_strings (view source)

**Defined in: uf_udobj.h**

### Overview
Adds strings to the free form data area of an UDO. Any number of valid null terminated string elements can be added. These calls are cumulative, if you make one call to add two strings, and then a second call to add three strings, you will end up with 5 strings on the UDO, and UF_UDOBJ_edit_strings would expect you to pass in 5 elements.

### Environment
Internal and External

### See Also
UF_UDOBJ_edit_strings
UF_UDOBJ_delete_strings

### Required License(s)
gateway

```
int UF_UDOBJ_add_strings
(
    tag_t udo_tag,
    unsigned int num_strings,
    char * strings [ ]
)
```

| | | | |
|---|---|---|---|
| tag_t | **udo_tag** | Input | The tag for the UDO |
| unsigned int | **num_strings** | Input | The number of strings to add. |
| char * | **strings [ ]** | Input | The strings to add to the UDO. |

## UF_UDOBJ_add_volumes (view source)

**Defined in: uf_udobj.h**

### Overview
Adds volume data to the convertible data area. Any number of doubles can be added to the volumes array. Each double represents a unit of volume. These calls are cumulative, if you make one call to add two volumes, and then a second call to add three volumes, you will end up with 5 volumes on the UDO, and UF_UDOBJ_edit_areas would expect you to pass in 5 elements.

### Environment
Internal and External

### See Also
UF_UDOBJ_edit_volumes
UF_UDOBJ_delete_volumes

### Required License(s)
gateway

```
int UF_UDOBJ_add_volumes
(
    tag_t udo_tag,
    unsigned int num_volumes,
    double * volumes
)
```

| | | | |
|---|---|---|---|
| tag_t | **udo_tag** | Input | The tag for the UDO |
| unsigned int | **num_volumes** | Input | The number of volumes to add. |
| double * | **volumes** | Input | The volume data to add to the UDO. |

---

## UF_UDOBJ_ask_class_data (view source)

**Defined in: uf_udobj.h**

### Overview
Finds both the class name and the user friendly name for the specified class identifier. The friendly_name appears in the class selection dialog if it is enabled.

### Environment
Internal and External

### Required License(s)
gateway

```
int UF_UDOBJ_ask_class_data
(
    UF_UDOBJ_class_t class_id,
    char * * class_name,
    char * * friendly_name
)
```

| | | | |
|---|---|---|---|
| UF_UDOBJ_class_t | **class_id** | Input | The class identifier from a previously created class. |
| char * * | **class_name** | Output to UF_*free* | The name of the class of UDOs. Use UF_free to<br>deallocate memory when done. |

| | | | |
|---|---|---|---|
| char * * | **friendly_name** | Output to UF_*free* | The user-friendly class name. Use UF_free to deallocate memory when done. |

---

## UF_UDOBJ_ask_class_id_of_name (view source)

**Defined in: uf_udobj.h**

### Overview
Finds the class id associated with the given class name.

### Environment
Internal and External

### Required License(s)
gateway

```
int UF_UDOBJ_ask_class_id_of_name
(
    const char * class_name,
    UF_UDOBJ_class_t * class_id
)
```

| | | | |
|---|---|---|---|
| const char * | **class_name** | Input | Name of a udo class - NOTE: do not use the "friendly" class name |
| UF_UDOBJ_class_t * | **class_id** | Output | Class id associated with the given class name |

---

## UF_UDOBJ_ask_owned_objects (view source)

**Defined in: uf_udobj.h**

### Overview
Queries the number of associated NX objects that have owning links
to the specified UDO.

### Environment
Internal and External

### Required License(s)
gateway

```
int UF_UDOBJ_ask_owned_objects
(
    tag_t udo_tag,
    int* num_owned_objects,
    tag_t* * owned_objects
)
```

| | | | |
|---|---|---|---|
| tag_t | **udo_tag** | Input | The tag of the UDO to query |

| | | | |
|---|---|---|---|
| int* | **num_owned_objects** | Output | The number of NX objects linked to the UDO by owning links. |
| tag_t* * | **owned_objects** | Output to UF_*free* | An array of tags of NX objects linked to the UDO by owning links. |

---

## UF_UDOBJ_ask_owning_udo (view source)

**Defined in: uf_udobj.h**

### Overview
Queries if the specified NX object is linked to a UDO by an owning link. Returns the tag of the UDO with the owning link. Returns NULL_TAG if there is no UDO with an owning link to the specified NX object.

### Environment
Internal and External

### Required License(s)
gateway

```
int UF_UDOBJ_ask_owning_udo
(
    tag_t obj_tag,
    tag_t* udo_tag
)
```

| | | | |
|---|---|---|---|
| tag_t | **obj_tag** | Input | The tag of the NX object to query |
| tag_t* | **udo_tag** | Output | The tag of the UDO which has an owning link to the specified NX object. |

---

## UF_UDOBJ_ask_udo_class_name (view source)

**Defined in: uf_udobj.h**

### Overview
Finds the class name associated with the UDO.

### Environment
Internal and External

### Required License(s)
gateway

```
int UF_UDOBJ_ask_udo_class_name
(
    tag_t udo_tag,
    char * * udo_class_name
```

)

| | | | |
|---|---|---|---|
| tag_t | **udo_tag** | Input | Tag for the UDO. |
| char * * | **udo_class_name** | Output to UF_*free* | The class name assigned to the UDO. Ensure the memory is freed using UF_free once the information is no longer needed. |

---

## UF_UDOBJ_ask_udo_data (view source)

**Defined in: uf_udobj.h**

### Overview

Finds all of the data associated with an UDO and returns it in the
data structure UF_UDOBJ_all_data_s. The order of the links returned by this
routine need not match with the order in which they are created through
UF_UDOBJ_add_links or UF_UDOBJ_add_owning_links. No assumptions should be made
in this regard.

### Environment

Internal and External

### See Also

UF_UDOBJ_free_udo_data
UF_UDOBJ_add_links
UF_UDOBJ_add_owning_links
UF_UDOBJ_all_data_t

### Required License(s)

gateway

```
int UF_UDOBJ_ask_udo_data
(
    tag_t udo_tag,
    UF_UDOBJ_all_data_t * all_data
)
```

| | | | |
|---|---|---|---|
| tag_t | **udo_tag** | Input | The tag for the UDO |
| UF_UDOBJ_all_data_t * | **all_data** | Output to UF_*free* | All of the data associated with the UDO. The structure that this points to becomes fully populated following this call. Certain structure members must be freed when no longer needed. This can be freed by calling UF_UDOBJ_free_udo_data |

---

## UF_UDOBJ_ask_udo_feature_of_udo (view source)

**Defined in: uf_udobj.h**

### Overview

Inquires the UDO feature tag of the input UDO object. A UDO object may have only one UDO feature associated to it.

### Environment
Internal & External

### History
This function was originally released in V15.0.

### Required License(s)
gateway

```
int UF_UDOBJ_ask_udo_feature_of_udo
(
    tag_t udo_tag,
    tag_t * udo_feature_tag
)
```

| | | | |
|---|---|---|---|
| tag_t | **udo_tag** | Input | The tag of the UDO object whose feature we are returning. |
| tag_t * | **udo_feature_tag** | Output | The tag of the UDO feature of the input object. NULL_TAG will be returned if there is no UDO feature for the input object. |

## UF_UDOBJ_ask_udo_links_to_obj (view source)

**Defined in: uf_udobj.h**

### Overview
Finds all of the UDO links to a particular NX object.

### Environment
Internal and External

### Required License(s)
gateway

```
int UF_UDOBJ_ask_udo_links_to_obj
(
    tag_t ug_tag,
    int * num_links,
    UF_UDOBJ_link_t * * udo_links
)
```

| | | | |
|---|---|---|---|
| tag_t | **ug_tag** | Input | The tag of the NX object for which to retrieve the UDO links. |
| int * | **num_links** | Output | The number of UDO links associated with the NX object. |
| UF_UDOBJ_link_t * * | **udo_links** | Output to UF_*free* | The links between UDOs and the NX object. Note that the member element udo_links[i].assoc_ug_tag will be the tag of |

| | | | |
|---|---|---|---|
| | | | the i+1 UDO that links to this object.<br>When done, use UF_free to deallocate memory for these links. |

## UF_UDOBJ_ask_udo_of_udo_feature (view source)

**Defined in: uf_udobj.h**

### Overview
Inquires the tag of the UDO object that is associated to the input UDO feature

### Environment
Internal & External

### History
This function was originally released in V15.0.

### Required License(s)
gateway

```
int UF_UDOBJ_ask_udo_of_udo_feature
(
    tag_t udo_feature_tag,
    tag_t * udo_tag
)
```

| | | | |
|---|---|---|---|
| tag_t | **udo_feature_tag** | Input | The tag of the UDO feature whose UDO object we are returning. |
| tag_t * | **udo_tag** | Output | The tag of the UDO object of the input object. |

## UF_UDOBJ_clear_link_status (view source)

**Defined in: uf_udobj.h**

### Overview
Clears the link status of a UDO.

### Environment
Internal and External

### Required License(s)
gateway

```
int UF_UDOBJ_clear_link_status
(
    tag_t udo_tag,
    UF_UDOBJ_link_p_t link_to_clear
)
```

| | | | |
|---|---|---|---|
| tag_t | **udo_tag** | Input | The tag of the UDO whose link's status is to be cleared. |
| UF_UDOBJ_link_p_t | **link_to_clear** | Input | The link whose status is to be cleared. |

## UF_UDOBJ_clear_udo_status (view source)

**Defined in: uf_udobj.h**

### Overview
Clears the out-of-date indicator of an UDO.

### Environment
Internal and External

### Required License(s)
gateway

```
int UF_UDOBJ_clear_udo_status
(
    tag_t udo_tag
)
```

| | | | |
|---|---|---|---|
| tag_t | **udo_tag** | Input | Tag for the UDO whose out-of-date indicator is to be cleared. |

## UF_UDOBJ_create_class (view source)

**Defined in: uf_udobj.h**

### Overview
Creates the class, given a class name, and friendly name for a particular type of UDO. Returns a class identifier which is used by UF_UDOBJ_create_udo to create the UDO and return its tag. UF_UI_add_to_class_sel can be used to enable the friendly name on the class selection dialog.

### Environment
Internal and External

### See Also
UF_UI_add_to_class_sel

### Required License(s)
gateway

```
int UF_UDOBJ_create_class
(
    const char * class_name,
    const char * friendly_name,
    UF_UDOBJ_class_t * class_id
```

)

| | | | |
|---|---|---|---|
| const char * | **class_name** | Input | The class name for the particular type of User Defined Object. |
| const char * | **friendly_name** | Input | A user-friendly name that appears in the class selection dialog (if enabled) and in other locations where a name is useful. |
| UF_UDOBJ_class_t * | **class_id** | Output | The class identifier. This identifier is not persistent across NX sessions nor is it stored in NX part files. |

---

## UF_UDOBJ_create_udo (view source)

**Defined in: uf_udobj.h**

### Overview

Creates a UDO of the class specified by the input class identifier. This creates a UDO "template" with a name and status. Note that creating a UDO will not display the UDO. This is so that you can add data that your display routines may need prior to the UDO being displayed. When the UDO has been fully created, call UF_DISP_add_item_to_display to actually display the UDO.

### Environment

Internal and External

### See Also

UF_DISP_add_item_to_display

### Required License(s)

gateway

```
int UF_UDOBJ_create_udo
(
    UF_UDOBJ_class_t class_id,
    tag_t * udo_tag
)
```

| | | | |
|---|---|---|---|
| UF_UDOBJ_class_t | **class_id** | Input | The class identifier from a previously created class. |
| tag_t * | **udo_tag** | Output | The tag for the UDO |

---

## UF_UDOBJ_create_udo_feature (view source)

**Defined in: uf_udobj.h**

### Overview

Creates both a UDO object and a UDO feature from an existing
class identifier. Note that UF_MODL_update() must be called to complete the

creation of the UDO feature and have it appear in the Model Navigation Tool.

### Environment
Internal & External

### History
This function was originally released in V15.0.

### Required License(s)
gateway

```
int UF_UDOBJ_create_udo_feature
(
    UF_UDOBJ_class_t class_id,
    tag_t * udo_tag,
    tag_t * udo_feature_tag
)
```

| | | | |
|---|---|---|---|
| UF_UDOBJ_class_t | **class_id** | Input | The class identifier from a previously created class |
| tag_t * | **udo_tag** | Output | The tag for the UDO |
| tag_t * | **udo_feature_tag** | Output | The tag for the UDO feature |

---

## UF_UDOBJ_create_udo_feature_from_udo (view source)

**Defined in: uf_udobj.h**

### Overview
Creates a UDO feature associated with a UDO object and returns the tag of the newly created feature. This function returns an error if you attempt to create a UDO feature for a UDO object which is already associated to a UDO feature.

### Environment
Internal & External

### History
This function was originally released in V15.0.

### Required License(s)
gateway

```
int UF_UDOBJ_create_udo_feature_from_udo
(
    tag_t udo_tag,
    tag_t * udo_feature_tag
)
```

| | | | |
|---|---|---|---|
| tag_t | **udo_tag** | Input | The UDO to create the UDO feature from |

| | | | |
|---|---|---|---|
| tag_t * | **udo_feature_tag** | Output | The newly created UDO feature |

## UF_UDOBJ_cycle_udos_by_class (view source)

**Defined in: uf_udobj.h**

### Overview

Cycles the UDOs in the part with the specified input tag. Input a NULL_TAG to start the cycle. The cycle ends when a NULL_TAG is returned.

Do not attempt to delete objects when cycling the database in a loop. Problems can occur when trying to read the next object when the current object has been deleted. To delete objects, save an array with the objects in it, and then when you have completed cycling, use UF_OBJ_delete_array_of_objects to delete the saved array of objects.

### Environment

Internal and External

### Required License(s)

gateway

```
int UF_UDOBJ_cycle_udos_by_class
(
    tag_t part_tag,
    UF_UDOBJ_class_t class_id,
    tag_t * udo_tag
)
```

| | | | |
|---|---|---|---|
| tag_t | **part_tag** | Input | The part in which to cycle UDOs. |
| UF_UDOBJ_class_t | **class_id** | Input | The UDO class to cycle. |
| tag_t * | **udo_tag** | Input / Output | The UDO found. The cycle starts and ends with a NULL_TAG. The tag from the previous cycle should be input for the next cycle. |

## UF_UDOBJ_delete_areas (view source)

**Defined in: uf_udobj.h**

### Overview

Deletes the convertible areas area of a UDO starting at the designated location and concluding with the specified numbers of areas to delete.

### Environment

Internal and External

### See Also

UF_UDOBJ_add_areas
UF_UDOBJ_edit_areas

### Required License(s)

gateway

**int UF_UDOBJ_delete_areas**
**(**
**tag_t udo_tag,**
**unsigned int start_loc,**
**unsigned int num_to_delete**
**)**

| | | | |
|---|---|---|---|
| tag_t | **udo_tag** | Input | The tag of the UDO whose areas are to be deleted. |
| unsigned int | **start_loc** | Input | The location within the UDO area values to start deleting from. Valid values are 1 through the number of areas in the UDO. |
| unsigned int | **num_to_delete** | Input | The numbers of areas to delete. |

## UF_UDOBJ_delete_doubles (view source)

**Defined in: uf_udobj.h**

### Overview

Deletes the free form doubles area of a UDO starting at the designated location and concluding with the specified numbers of doubles to delete.

### Environment

Internal and External

### See Also

UF_UDOBJ_add_doubles
UF_UDOBJ_edit_doubles

### Required License(s)

gateway

**int UF_UDOBJ_delete_doubles**
**(**
**tag_t udo_tag,**
**unsigned int start_loc,**
**unsigned int num_to_delete**
**)**

| | | | |
|---|---|---|---|
| tag_t | **udo_tag** | Input | The tag of the UDO whose doubles are to be deleted. |
| unsigned int | **start_loc** | Input | The location within the UDO double area to start deleting from. Valid values are 1 through the number of doubles in the UDO. |

| | | | |
|---|---|---|---|
| unsigned int | **num_to_delete** | Input | The numbers of doubles to delete. |

---

## UF_UDOBJ_delete_integers (view source)

**Defined in: uf_udobj.h**

### Overview
Deletes the free form integer area of a UDO starting at the designated location and concluding with the specified numbers of integers to delete.

### Environment
Internal and External

### See Also
UF_UDOBJ_add_integers
UF_UDOBJ_edit_integers

### Required License(s)
gateway

```
int UF_UDOBJ_delete_integers
(
    tag_t udo_tag,
    unsigned int start_loc,
    unsigned int num_to_delete
)
```

| | | | |
|---|---|---|---|
| tag_t | **udo_tag** | Input | The tag of the UDO whose integers are to be deleted. |
| unsigned int | **start_loc** | Input | The location within the UDO integer area to start deleting from. Valid values are 1 through the number of integers in the UDO. |
| unsigned int | **num_to_delete** | Input | The numbers of integers to delete. |

---

## UF_UDOBJ_delete_lengths (view source)

**Defined in: uf_udobj.h**

### Overview
Deletes the convertible lengths area of a UDO starting at the designated location and concluding with the specified numbers of lengths to delete.

### Environment
Internal and External

### See Also
UF_UDOBJ_add_lengths
UF_UDOBJ_edit_lengths

### Required License(s)
gateway

**int UF_UDOBJ_delete_lengths**
**(**
**tag_t udo_tag,**
**unsigned int start_loc,**
**unsigned int num_to_delete**
**)**

| | | | |
|---|---|---|---|
| tag_t | **udo_tag** | Input | The tag of the UDO whose lengths are to be deleted. |
| unsigned int | **start_loc** | Input | The location within the UDO length area to start deleting from. Valid values are 1 through the number of lengths in the UDO. |
| unsigned int | **num_to_delete** | Input | The numbers of lengths to delete. |

---

## UF_UDOBJ_delete_link (view source)

**Defined in: uf_udobj.h**

### Overview
Deletes a UDO's link.

### Environment
Internal and External

### See Also
UF_UDOBJ_add_links
UF_UDOBJ_edit_links

### Required License(s)
gateway

**int UF_UDOBJ_delete_link**
**(**
**tag_t udo_tag,**
**UF_UDOBJ_link_p_t link_to_delete**
**)**

| | | | |
|---|---|---|---|
| tag_t | **udo_tag** | Input | The tag of the UDO whose link is to be deleted. |
| UF_UDOBJ_link_p_t | **link_to_delete** | Input | The link to delete from the UDO. |

---

## UF_UDOBJ_delete_owning_link (view source)

**Defined in: uf_udobj.h**

### Overview

Deletes the owning link between a UDO and its associated NX object.

### Environment

Internal and External

### See Also

UF_UDOBJ_add_owning_links

### Required License(s)

gateway

```
int UF_UDOBJ_delete_owning_link
(
    tag_t udo_tag,
    tag_t ug_tag
)
```

| | | | |
|---|---|---|---|
| tag_t | **udo_tag** | Input | The tag UDO whose owning link is to be deleted. |
| tag_t | **ug_tag** | Input | The tag of the NX object whose UDO owning link is to be deleted. |

---

## UF_UDOBJ_delete_strings (view source)

**Defined in: uf_udobj.h**

### Overview

Deletes the free form strings area of a UDO starting at the designated location and concluding with the specified numbers of strings to delete.

### Environment

Internal and External

### See Also

UF_UDOBJ_add_strings
UF_UDOBJ_edit_strings

### Required License(s)

gateway

```
int UF_UDOBJ_delete_strings
(
    tag_t udo_tag,
    unsigned int start_loc,
    unsigned int num_to_delete
)
```

| | | | |
|---|---|---|---|
| tag_t | **udo_tag** | Input | The tag of the UDO whose strings are to be deleted. |
| unsigned int | **start_loc** | Input | The location within the UDO string area to start deleting from. Valid values are 1 through the number of strings in the UDO. |
| unsigned int | **num_to_delete** | Input | The numbers of strings to delete. |

---

## UF_UDOBJ_delete_volumes (view source)

**Defined in: uf_udobj.h**

### Overview
Deletes the convertible volumes area of a UDO starting at the designated location and concluding with the specified numbers of volumes to delete.

### Environment
Internal and External

### See Also
UF_UDOBJ_add_volumes
UF_UDOBJ_edit_volumes

### Required License(s)
gateway

```
int UF_UDOBJ_delete_volumes
(
    tag_t udo_tag,
    unsigned int start_loc,
    unsigned int num_to_delete
)
```

| | | | |
|---|---|---|---|
| tag_t | **udo_tag** | Input | The tag of the UDO whose volumes are to be deleted. |
| unsigned int | **start_loc** | Input | The location within the UDO volume area to start deleting from. Valid values are 1 through the number of volumes in the UDO. |
| unsigned int | **num_to_delete** | Input | The numbers of volumes to delete. |

---

## UF_UDOBJ_edit_areas (view source)

**Defined in: uf_udobj.h**

### Overview
Edits the areas convertible data area of a UDO. The input array must have elements for all of the areas currently on the UDO. You should know the data

model for your UDO, but if you are unsure of the number of areas on a UDO, UF_UDOBJ_ask_udo_data will return this information.

### Environment
Internal and External

### See Also
UF_UDOBJ_delete_areas
UF_UDOBJ_add_areas
UF_UDOBJ_ask_udo_data

### Required License(s)
gateway

```
int UF_UDOBJ_edit_areas
(
    tag_t udo_tag,
    double areas [ ]
)
```

| | | | |
|---|---|---|---|
| tag_t | **udo_tag** | Input | The tag of the UDO whose areas are to be edited. |
| double | **areas [ ]** | Input | The new areas for the UDO. |

---

## UF_UDOBJ_edit_doubles (view source)

**Defined in: uf_udobj.h**

### Overview
Edits the free form double area of a UDO. The input array must have elements for all of the doubles currently on the UDO. You should know the data model for your UDO, but if you are unsure of the number of doubles on a UDO, UF_UDOBJ_ask_udo_data will return this information.

### Environment
Internal and External

### See Also
UF_UDOBJ_delete_doubles
UF_UDOBJ_add_doubles
UF_UDOBJ_ask_udo_data

### Required License(s)
gateway

```
int UF_UDOBJ_edit_doubles
(
    tag_t udo_tag,
    double doubles [ ]
)
```

| | | | |
|---|---|---|---|
| tag_t | **udo_tag** | Input | The tag of the UDO whose doubles are to be edited. |

| | | | |
|---|---|---|---|
| double | **doubles [ ]** | Input | The new doubles for the UDO. |

---

## UF_UDOBJ_edit_integers (view source)

**Defined in: uf_udobj.h**

### Overview
Edits the free form integer area of a UDO. The input array must have elements for all of the integers currently on the UDO. You should know the data model for your UDO, but if you are unsure of the number of integers on a UDO, UF_UDOBJ_ask_udo_data will return this information.

### Environment
Internal and External

### See Also
UF_UDOBJ_delete_integers
UF_UDOBJ_add_integers
UF_UDOBJ_ask_udo_data

### Required License(s)
gateway

```
int UF_UDOBJ_edit_integers
(
    tag_t udo_tag,
    int integers [ ]
)
```

| | | | |
|---|---|---|---|
| tag_t | **udo_tag** | Input | The tag of the UDO whose integers are to be edited. |
| int | **integers [ ]** | Input | The new integers for the UDO. |

---

## UF_UDOBJ_edit_lengths (view source)

**Defined in: uf_udobj.h**

### Overview
Edits the lengths convertible data area of a UDO. The input array must have elements for all of the lengths currently on the UDO. You should know the data model for your UDO, but if you are unsure of the number of lengths on a UDO, UF_UDOBJ_ask_udo_data will return this information.

### Environment
Internal and External

### See Also
UF_UDOBJ_delete_lengths
UF_UDOBJ_add_lengths
UF_UDOBJ_ask_udo_data

### Required License(s)

gateway

```
int UF_UDOBJ_edit_lengths
(
    tag_t udo_tag,
    double lengths [ ]
)
```

| | | | |
|---|---|---|---|
| tag_t | **udo_tag** | Input | The tag of the UDO whose lengths are to be edited. |
| double | **lengths [ ]** | Input | The new lengths for the UDO. |

---

## UF_UDOBJ_edit_link (view source)

**Defined in: uf_udobj.h**

### Overview

Edits a UDO's link with the following edit features:
Replaces all the links for a specific link type within a UDO for a particular NX object with another NX object.
Only affects a single link type at a time. Therefore, if there are multiple links within a UDO to the same object, each with separate link types, this function must be used once for each link type.
If there are multiple links to the same NX object with the same link type, all the links are affected.

### Environment

Internal and External

### See Also

UF_UDOBJ_ask_udo_data

### Required License(s)

gateway

```
int UF_UDOBJ_edit_link
(
    tag_t udo_tag,
    UF_UDOBJ_link_p_t link_to_edit,
    tag_t new_assoc_ug_tag
)
```

| | | | |
|---|---|---|---|
| tag_t | **udo_tag** | Input | The tag of the UDO whose link is to be edited. |
| UF_UDOBJ_link_p_t | **link_to_edit** | Input | The link within the UDO whose reference is to be changed. |
| tag_t | **new_assoc_ug_tag** | Input | The tag to insert into the link in place of the tag currently in use. |

---

## UF_UDOBJ_edit_links (view source)

**Defined in: uf_udobj.h**

### Overview

Edit the link records of a UDO. The input array should contain all of the link records of the UDO. The edit will start at location link_defs[start_location] and continue for count elements of the array.

### Environment

Internal and External

### Required License(s)

gateway

```
int UF_UDOBJ_edit_links
(
    tag_t udo_tag,
    const UF_UDOBJ_link_t * link_def,
    unsigned int start_location,
    unsigned int count
)
```

| | | | |
|---|---|---|---|
| tag_t | **udo_tag** | Input | The tag of the UDO whose links are to be edited |
| const UF_UDOBJ_link_t * | **link_def** | Input | The new links for the UDO |
| unsigned int | **start_location** | Input | The location in the link_defs array where editing is to begin (This should be a "C" based array index.) |
| unsigned int | **count** | Input | The count of links to be edited from start_location |

---

## UF_UDOBJ_edit_strings (view source)

**Defined in: uf_udobj.h**

### Overview

Edits the free form string area of a UDO. The input array must have elements for all of the strings currently on the UDO. You should know the data model for your UDO, but if you are unsure of the number of strings on a UDO, UF_UDOBJ_ask_udo_data will return this information.

### Environment

Internal and External

### See Also

UF_UDOBJ_delete_strings
UF_UDOBJ_add_strings
UF_UDOBJ_ask_udo_data

### Required License(s)

gateway

```
int UF_UDOBJ_edit_strings
(
    tag_t udo_tag,
    char * strings [ ]
)
```

| | | | |
|---|---|---|---|
| tag_t | **udo_tag** | Input | The tag of the UDO whose strings are to be edited. |
| char * | **strings [ ]** | Input | The new strings for the UDO. |

---

## UF_UDOBJ_edit_udo_of_udo_feature (view source)

**Defined in: uf_udobj.h**

### Overview

Edits a UDO feature by replacing the existing tag of the UDO object with the input tag of the UDO object. The old UDO object is no longer associated to a UDO feature and may be associated to another UDO feature. If the input UDO object is associated to another UDO feature, the editing is not done and this routine returns an error.

### Environment

Internal & External

### History

This function was originally released in V15.0

### Required License(s)

gateway

```
int UF_UDOBJ_edit_udo_of_udo_feature
(
    tag_t udo_feature_tag,
    tag_t udo_tag
)
```

| | | | |
|---|---|---|---|
| tag_t | **udo_feature_tag** | Input | The UDO feature to edit. |
| tag_t | **udo_tag** | Input | The UDO to edit the UDO feature with |

---

## UF_UDOBJ_edit_volumes (view source)

**Defined in: uf_udobj.h**

### Overview

Edits the volumes convertible data area of a UDO. The input array must have elements for all of the volumes currently on the UDO. You should know the data model for your UDO, but if you are unsure of the number of volumes on a UDO, UF_UDOBJ_ask_udo_data will return this information.

### Environment

Internal and External

### See Also

UF_UDOBJ_delete_volumes
UF_UDOBJ_add_volumes
UF_UDOBJ_ask_udo_data

### Required License(s)

gateway

```
int UF_UDOBJ_edit_volumes
(
    tag_t udo_tag,
    double volumes [ ]
)
```

| | | | |
|---|---|---|---|
| tag_t | **udo_tag** | Input | The tag of the UDO whose volumes are to be edited. |
| double | **volumes [ ]** | Input | The new volumes for the UDO. |

## UF_UDOBJ_free_udo_data (view source)

**Defined in: uf_udobj.h**

### Overview

Frees all of the data that was dynamically allocated to the address of the UF_UDOBJ_all_data_s structure. This structure is populated with data after a call by UF_UDOBJ_ask_udo_data .

### Environment

Internal and External

### See Also

UF_UDOBJ_ask_udo_data

### Required License(s)

gateway

```
int UF_UDOBJ_free_udo_data
(
    UF_UDOBJ_all_data_p_t all_data
)
```

| | | | |
|---|---|---|---|
| UF_UDOBJ_all_data_p_t | **all_data** | Input | A structure previously populated by UF_UDOBJ_ask_udo_data. This function frees all the data dynamically allocated by that function. |

## UF_UDOBJ_is_obj_linked_to_udo (view source)

**Defined in: uf_udobj.h**

### Overview
Queries whether UDOs reference the specified NX object.

### Environment
Internal and External

### Required License(s)
gateway

```
int UF_UDOBJ_is_obj_linked_to_udo
(
    tag_t ug_tag,
    logical * linked
)
```

| | | | |
|---|---|---|---|
| tag_t | **ug_tag** | Input | The tag of the NX object to query as to whether UDOs reference it or not. |
| logical * | **linked** | Output | TRUE = UDOs reference NX object<br>FALSE = UDOs do not reference NX object. |

## UF_UDOBJ_is_owned (view source)

**Defined in: uf_udobj.h**

### Overview
Queries if an NX object has an owning link by a UDO.

### Environment
Internal and External

### Required License(s)
gateway

```
int UF_UDOBJ_is_owned
(
    tag_t obj_tag,
    logical * owned
)
```

| | | | |
|---|---|---|---|
| tag_t | **obj_tag** | Input | The tag of the NX object to query |
| logical * | **owned** | Output | TRUE = NX Object has an owning link to a UDO.<br>FALSE = No owning link |

## UF_UDOBJ_is_udo_feature (view source)

**Defined in: uf_udobj.h**

### Overview
Inquires whether a feature is a UDO feature.

### Environment
Internal & External

### History
This function was originally released in V15.0.

### Required License(s)
gateway

```
int UF_UDOBJ_is_udo_feature
(
    tag_t feature_tag,
    logical * is_udo_feature
)
```

| | | | |
|---|---|---|---|
| tag_t | **feature_tag** | Input | the feature to check |
| logical * | **is_udo_feature** | Output | TRUE if a UDO feature else FALSE |

## UF_UDOBJ_log_udo_feature_for_update (view source)

**Defined in: uf_udobj.h**

### Overview
Explicitly log the UDO feature for update. This may be called for edits that do not implicitly log the UDO feature. UF_UDOBJ_edit_integers(), for example, does not cause the UDO feature to be logged for update. If you consider the UDO to be out of date, call this routine and then call UF_MODL_update().

### Environment
Internal and External

### History
Originally released in V16.0

### Required License(s)
gateway

```
int UF_UDOBJ_log_udo_feature_for_update
(
    tag_t udo_feature_tag
)
```

| | | | |
|---|---|---|---|
| tag_t | **udo_feature_tag** | Input | The UDO feature to log for update |

## UF_UDOBJ_register_attn_pt_cb (view source)

**Defined in: uf_udobj.h**

### Overview

Registers the attention point function (method) to call when the UDOs with the input class identifier pass through the attention point computation event. The attention point is a temporary display point near an object that is used during particular NX events. For example, during Info-->Object a temporary number is placed near the object. NX calculates the location for the attention point based on the primitives used in constructing the UDO.

### Environment

Internal and External

### Required License(s)

gateway

```
int UF_UDOBJ_register_attn_pt_cb
(
    UF_UDOBJ_class_t class_id,
    UF_UDOBJ_attn_pt_f_t attn_pt_func
)
```

| | | | |
|---|---|---|---|
| UF_UDOBJ_class_t | **class_id** | Input | The class identifier from a previously created class. |
| UF_UDOBJ_attn_pt_f_t | **attn_pt_func** | Input | The function to invoke when UDOs for this class pass through attention point computations. |

## UF_UDOBJ_register_delete_cb (view source)

**Defined in: uf_udobj.h**

### Overview

Register's the delete function (method) to call when a UDO with a type 2 or 3 linked object associated to the UDO passes through delete. Objects associated to the UDO with a type 1 link do not invoke the method because the UDO is deleted when its associated object is deleted. Objects associated to a UDO with a UDO owning link do not cause the method to be invoked because the associated object can not be deleted directly.

The following restrictions apply:

. The callback routine must not call UF_MODL_update.
. No Features may be deleted from this callback.
. The work part must not be changed.

This callback should not call any functions that change the work or displayed part. This includes functions that change the work part as part of their operation, such as UF_PART_import.

### Environment
Internal and External

### Required License(s)
gateway

```
int UF_UDOBJ_register_delete_cb
(
    UF_UDOBJ_class_t class_id,
    UF_UDOBJ_delete_f_t delete_func
)
```

| | | | |
|---|---|---|---|
| UF_UDOBJ_class_t | **class_id** | Input | The class identifier from a previously created class. |
| UF_UDOBJ_delete_f_t | **delete_func** | Input | The function to invoke when objects associated to UDOs in this class pass through delete. |

---

## UF_UDOBJ_register_display_cb (view source)

**Defined in: uf_udobj.h**

### Overview
Registers the display function (method) to call when the UDOs with the input class identifier pass through the display event.

### Environment
Internal and External

### Required License(s)
gateway

```
int UF_UDOBJ_register_display_cb
(
    UF_UDOBJ_class_t class_id,
    UF_UDOBJ_display_f_t display_func
)
```

| | | | |
|---|---|---|---|
| UF_UDOBJ_class_t | **class_id** | Input | The class identifier from a previously created class. |
| UF_UDOBJ_display_f_t | **display_func** | Input | The function to invoke when UDOs for this class pass through the display. |

---

## UF_UDOBJ_register_edit_cb (view source)

**Defined in: uf_udobj.h**

### Overview
Registers the function (method) which is invoked when editing this UDO.

### Environment
Internal and External

### Required License(s)
gateway

```
int UF_UDOBJ_register_edit_cb
(
    UF_UDOBJ_class_t class_id,
    UF_UDOBJ_edit_f_t edit_func
)
```

| | | | |
|---|---|---|---|
| UF_UDOBJ_class_t | **class_id** | Input | The class identifier from a previously created class. |
| UF_UDOBJ_edit_f_t | **edit_func** | Input | The function to invoke when editing the UDO. |

## UF_UDOBJ_register_fit_cb (view source)

**Defined in: uf_udobj.h**

### Overview
Registers the fit function (method) to call when the UDOs with the input class identifier pass through the fit into view event. NX calculates the extents of clipping required based on the primitives used in the UDO's construction.

### Environment
Internal and External

### Required License(s)
gateway

```
int UF_UDOBJ_register_fit_cb
(
    UF_UDOBJ_class_t class_id,
    UF_UDOBJ_fit_f_t fit_func
)
```

| | | | |
|---|---|---|---|
| UF_UDOBJ_class_t | **class_id** | Input | The class identifier from a previously created class. |
| UF_UDOBJ_fit_f_t | **fit_func** | Input | The function to invoke when UDOs for this class pass through fit to the view display code. |

## UF_UDOBJ_register_info_obj_cb (view source)

**Defined in: uf_udobj.h**

### Overview
Registers the Info-->Object method to call when the UDOs with the input class identifier are queried interactively in NX with the Info-->Object menu option. It is safe to register a method in external mode, but Info-->Object functionality is only available in internal mode.

### Environment
Internal and External

### Required License(s)
gateway

```
int UF_UDOBJ_register_info_obj_cb
(
    UF_UDOBJ_class_t class_id,
    UF_UDOBJ_info_obj_f_t info_object_method
)
```

| | | | |
|---|---|---|---|
| UF_UDOBJ_class_t | **class_id** | Input | The class identifier for which to register the Info-->Object method. |
| UF_UDOBJ_info_obj_f_t | **info_object_method** | Input | The method to use in interactive NX when you invoke Info-->Object on a UDO of this class. |

---

## UF_UDOBJ_register_is_occurrenceable_cb (view source)

**Defined in: uf_udobj.h**

### Overview
Registers the function (method) which determines if this UDO is occurrenceable.

### Environment
Internal and External

### Required License(s)
gateway

```
int UF_UDOBJ_register_is_occurrenceable_cb
(
    UF_UDOBJ_class_t class_id,
    UF_UDOBJ_is_occurrenceable_f_t is_occurrenceable_func
)
```

| | | | |
|---|---|---|---|
| UF_UDOBJ_class_t | **class_id** | Input | The class identifier from a previously created class. |

| | | | |
|---|---|---|---|
| UF_UDOBJ_is_occurrenceable_f_t | **is_occurrenceable_func** | Input | The function to invoke to determine if this UDO should be occurrenceable. |

## UF_UDOBJ_register_screen_size_fit_cb (view source)

**Defined in: uf_udobj.h**

### Overview

Registers the screen size fit function (method) to call when the UDOs with the input class identifier pass through the add screen size objects to fit into view event. "Screen Size Fit" applies to primitives whose size remains the same on the screen regardless of the scale of the view. The size of other primitives on the screen changes as the view scale changes. NX calculates the extents of clipping required based on the screen size primitives used in the UDO's construction.

You should only call this function if your UDO contains geometry which can participate in "Screen Size Fit." As of NX 8.0, this is only ScreenStandardText and AbsoluteRotationScreenSizeText, which are available in class NX/Open class UserDefinedObjects::UserDefinedObjectDisplayContext.

### Environment

Internal and External

### History

NX 8.0

### Required License(s)

gateway

```
int UF_UDOBJ_register_screen_size_fit_cb
(
    UF_UDOBJ_class_t class_id,
    UF_UDOBJ_screen_size_fit_f_t screen_size_fit_func
)
```

| | | | |
|---|---|---|---|
| UF_UDOBJ_class_t | **class_id** | Input | The class identifier from a previously created class. |
| UF_UDOBJ_screen_size_fit_f_t | **screen_size_fit_func** | Input | The function to invoke when UDOs for this class pass through add screen size objects to fit to the view display code. |

## UF_UDOBJ_register_select_cb (view source)

**Defined in: uf_udobj.h**

### Overview

Registers the select function (method) to call when the UDOs with the input class identifier pass through the selection event. This UDO

callback informs NX of the extents of the UDO via the
display primitives, then NX's internal algorithms decide to
select the UDO or not. UF_UI_add_to_class_sel can be used to enable
the friendly name on the class selection dialog.

### Environment
Internal and External

### See Also
UF_UI_add_to_class_sel

### Required License(s)
gateway

```
int UF_UDOBJ_register_select_cb
(
    UF_UDOBJ_class_t class_id,
    UF_UDOBJ_select_f_t select_func
)
```

| | | | |
|---|---|---|---|
| UF_UDOBJ_class_t | **class_id** | Input | The class identifier from a previously created class. |
| UF_UDOBJ_select_f_t | **select_func** | Input | The function to invoke when UDOs for this class pass through selection. |

## UF_UDOBJ_register_suppress_cb (view source)

**Defined in: uf_udobj.h**

### Overview
Registers the function (method) which is invoked when suppressing this UDO feature.

Note this method is not called unless you have a UDO FEATURE. Also it may not
get called when the system automatically suppresses the feature during update.

Also note the user should call UF_MODL_ask_suppress_feature() in their callback to
see if the input udo feature is currently getting suppressed or unsuppressed.

### Environment
Internal and External

### Required License(s)
gateway

```
int UF_UDOBJ_register_suppress_cb
(
    UF_UDOBJ_class_t class_id,
    UF_UDOBJ_suppress_f_t suppress_func
)
```

| | | | |
|---|---|---|---|
| UF_UDOBJ_class_t | **class_id** | Input | The class identifier from a previously created class. |

| | | | |
|---|---|---|---|
| UF_UDOBJ_suppress_f_t | **suppress_func** | Input | The function to invoke when suppressing or unsuppressing the UDO feature. |

---

## UF_UDOBJ_register_update_cb (view source)

**Defined in: uf_udobj.h**

### Overview

Registers the update function (method) to call when the UDOs with
the input class identifier pass through the update event. The method is
called when a UDO with either a link type 1 or a link type 3
association to it passes through update. Link type 2 and the UDO
owning link do not invoke the method because objects associated to
UDOs with link type 2 and UDO owning links do not add the UDOs
to the update list, by definition.

While within the update callback, you may freely query the data
model. You may also edit the free form data areas in the UDO.
Additionally, you may display a dialog (in internal Open API) to
inform the NX user of the affect the edit may have on the
UDO.

This callback should not call any functions that change the work or displayed
part. This includes functions that change the work part as part of their
operation, such as UF_PART_import.

Note that if an object you are linked to (with a type 1 or a type 3 link) is
deleted, the UDO update method will be called. If your update method does
nothing, you will remain linked to a condemned object, and you must take that
into account when you try to use the linked object. If you want, your update
method can loop over all linked objects, calling UF_OBJ_ask_status, if you
are linked to an object that has a status of UF_OBJ_CONDEMNED, then the update
method can decide what is the appropriate action to take.

The following restrictions apply:

. The update process must not be explicitly invoked again with UF_MODL_update.
. The work part must not be changed.
. No Features may be deleted from this callback.

### Environment

Internal and External

### Required License(s)

gateway

```
int UF_UDOBJ_register_update_cb
(
    UF_UDOBJ_class_t class_id,
    UF_UDOBJ_update_f_t update_func
)
```

| | | | |
|---|---|---|---|
| UF_UDOBJ_class_t | **class_id** | Input | The class identifier from a previously created class. |

| | | | |
|---|---|---|---|
| UF_UDOBJ_update_f_t | **update_func** | Input | The function to invoke when UDOs for this class pass through update. |

## UF_UDOBJ_set_owned_object_selection (view source)

**Defined in: uf_udobj.h**

### Overview

Set the behavior of selection of owned objects for the UDO class. The default selection mode is UF_UDOBJ_DONT_ALLOW_SELECTION. This means that an owned object can not be selected independently.

Setting the selection of owned objects to UF_UDOBJ_ALLOW_SELECTION means that if an owned object and the UDO are both selectable, selecting the owned object will result in the owned NX Object being selected and Up One Level button activated, in interactive selection. If the UDO is not selectable selection of the owned object would still be possible.

You may want to set UF_UDOBJ_ALLOW_SELECTION when your UDO contains only one owned object, and you want to inherit the characteristics of that object. For instance if your owned object is a spline, which your UDO computes the knot points for, you may want the spline to be selectable by modeling operations, such as extrude. In this case you would set your UDO class to allow selection of the owned object.

Even if UF_UDOBJ_ALLOW_SELECTION is set, the owned object is not eligible to be selected interactively for deletion.

### Environment

Internal and External

### History

Originally released in V16.0.2

### Required License(s)

gateway

```
int UF_UDOBJ_set_owned_object_selection
(
    UF_UDOBJ_class_t class_id,
    UF_UDOBJ_owned_object_selection_t value
)
```

| | | | |
|---|---|---|---|
| UF_UDOBJ_class_t | **class_id** | Input | The class of the UDO which is being set. |
| UF_UDOBJ_owned_object_selection_t | **value** | Input | The selection type that is to be set for this UDO class. |

## UF_UDOBJ_set_query_class_id (view source)

**Defined in: uf_udobj.h**

### Overview

Set the behavior of querying the class id given the class name. The default mode is UF_UDOBJ_DONT_ALLOW_QUERY_CLASS_ID. This means that if you call UF_UDOBJ_ask_class_id_of_name with the UDO's class name, an error code will be returned, and the class id returned will be 0. This mechanism is used to help protect the integrity of a proprietary UDO.

Setting the query to UF_UDOBJ_ALLOW_QUERY_CLASS_ID means that given a class name, you can find the class id. This mechanism is used to pass the class id across multiple shared libraries.

### Environment

Internal and External

### History

Originally released in V16.0.2

### Required License(s)

gateway

```
int UF_UDOBJ_set_query_class_id
(
    UF_UDOBJ_class_t class_id,
    UF_UDOBJ_query_class_id_t value
)
```

| | | | |
|---|---|---|---|
| UF_UDOBJ_class_t | **class_id** | Input | The class of the UDO which is being set. |
| UF_UDOBJ_query_class_id_t | **value** | Input | The query type that is to be set for this UDO class. |

---

## UF_UDOBJ_set_user_warn_flag (view source)

**Defined in: uf_udobj.h**

### Overview

Set the behavior of warning the user if a UDO of the given class is found in a part, but the code implementing the methods for the UDO is not loaded. The default action is to not warn the user. If the UDO author sets this flag to TRUE, all UDO's of this class that are created will be marked so that the user will be warned if the UDO methods have not been loaded, but a UDO of the class is in the part. This warning will be issued to the listing window, when the first object of the given class is retrieved. This warning will only be given once per session.

This flag is set on every UDO object. Therefore for any part, there may be a mixture UDO objects of a given class, some having this flag set to TRUE and some objects having the flag set to FALSE. This is particularly true since all UDO objects created before NX 3.0 will have this flag set to FALSE. If the UDO methods for a class are not loaded, any one UDO with this flag set to TRUE in a part is enough for the warning to be issued to the listing window.

### Environment

Internal and External

### History

Originally released in NX 3.0

### Required License(s)

gateway

**int UF_UDOBJ_set_user_warn_flag**
**(**
**UF_UDOBJ_class_t class_id,**
**logical value**
**)**

| | | | |
|---|---|---|---|
| UF_UDOBJ_class_t | **class_id** | Input | The class of the UDO which is being set. |
| logical | **value** | Input | Should the user be warned if the UDO methods are not loaded?<br>TRUE - the user will be warned<br>FALSE - the user will not be warned.<br>Note that the default if this routine is not called is FALSE. |

---

## UF_UDOBJ_version_udo (view source)

**Defined in: uf_udobj.h**

### Overview

Assigns an existing UDO to a new class id so that it can adopt the callback/behaviors of the new class.

### Environment

Internal and External

### Required License(s)

gateway

**int UF_UDOBJ_version_udo**
**(**
**tag_t udo_tag,**
**UF_UDOBJ_class_t class_id**
**)**

| | | | |
|---|---|---|---|
| tag_t | **udo_tag** | Input | Tag of the UDO whose class you wish to change. |
| UF_UDOBJ_class_t | **class_id** | Input | The udo will now belong in this class. |

---