UF_ROUTE_add_segment_to_stock (view source)

Defined in: uf_route.h

Overview
Adds a segment to a given stock object; this is basically the operation of assigning the stock to the segment.

Environment
Internal and External

History
Original release was in V15.0.

Required License(s)
(routing_base or routing_advanced)

```
int UF_ROUTE_add_segment_to_stock
(
    tag_t stock,
    tag_t segment
)
```

tag_t	**stock**	Input	Stock object to modify
tag_t	**segment**	Input	Segment to add to the stock

UF_ROUTE_add_terminal_ports (view source)

Defined in: uf_route.h

Overview
Add given terminal ports to the given port

Environment
Internal and External

See Also
UF_ROUTE_add_virtual_ports

History
Originally released in V16.0

Required License(s)
routing_base

```
int UF_ROUTE_add_terminal_ports
(
    tag_t multi,
    int num_terms,
    tag_t * terms
)
```

tag_t	**multi**	Input	Tag of multiport
int	**num_terms**	Input	Number of terminal ports
tag_t *	**terms**	Input	Array of terminal port tags

UF_ROUTE_add_virtual_ports (view source)

Defined in: uf_route.h

Overview
Add given virtual ports to the given port

Environment
Internal and External

See Also
UF_ROUTE_add_virtual_ports

History
Originally released in V16.0

Required License(s)
routing_base

```
int UF_ROUTE_add_virtual_ports
(
    tag_t multi,
    int num_terms,
    char * * terms
)
```

tag_t	**multi**	Input	Tag of multiport
int	**num_terms**	Input	Number of virtual ports
char * *	**terms**	Input	Array of virtual port unique ids

UF_ROUTE_align_stock (view source)

Defined in: uf_route.h

Overview
Modifies the stock object rotation about the segment to point in the direction of the given vector.

Environment
Internal and External

History
Original release was in V15.0.

Required License(s)
routing_base

```
int UF_ROUTE_align_stock
(
    tag_t stock,
    double rotate_vec [ 3 ]
)
```

tag_t	**stock**	Input	Stock object to modify
double	**rotate_vec [3]**	Input	New vector for rotation

UF_ROUTE_are_ports_connectable (view source)

Defined in: uf_route.h

Overview
Determines whether two ports would make a valid connection. Which means they satisfy the following conditions: The port positions are equivalent, the port alignment vectors are colinear and opposed, the port rotation vectors (if defined) match, and neither port is interior to a part.

Return
TRUE = Ports make a valid connection
FALSE = Ports make invalid connection

Environment
Internal and External

History
Original release was in V15.0.

Required License(s)
gateway

```
logical UF_ROUTE_are_ports_connectable
(
    tag_t port1,
    tag_t port2
)
```

tag_t	**port1**	Input	Port to test
tag_t	**port2**	Input	Port to test

UF_ROUTE_are_segments_tangent (view source)

Defined in: uf_route.h

Overview

Tests to see whether two segments are tangent to one another where they both are connected at the given RCP.

Return

Return code:
TRUE = Segments are tangent
FALSE = Segments are not tangent

Environment

Internal and External

Required License(s)

gateway

```
logical UF_ROUTE_are_segments_tangent
(
    tag_t segment1,
    tag_t segment,
    tag_t rcp
)
```

tag_t	**segment1**	Input	Segment to test
tag_t	**segment**	Input	Segment to test
tag_t	**rcp**	Input	RCP common to both segments

UF_ROUTE_ask_anchor_position (view source)

Defined in: uf_route.h

Overview

Returns the position (in absolute coordinates) of an anchor. If the tag is an occurrence of an anchor, then the position of the occurrence is returned.

Environment

Internal and External

Required License(s)

gateway

```
int UF_ROUTE_ask_anchor_position
(
    tag_t anchor_tag,
    double position [ 3 ]
)
```

tag_t	**anchor_tag**	Input	Anchor to query
double	**position [3]**	Output	Position of anchor (ABS)

UF_ROUTE_ask_anchor_stock (view source)

Defined in: uf_route.h

Overview
When stock is assigned to a segment, or segments, an anchor is chosen for the placement of the stock on the centerline. This routine inquires, for a given anchor, which stock is using this anchor for that position.

Environment
Internal and External

Required License(s)
gateway

```
int UF_ROUTE_ask_anchor_stock
(
    tag_t anchor_tag,
    int * num_stocks,
    tag_t * * stocks
)
```

tag_t	**anchor_tag**	Input	Tag of anchor to query
int *	**num_stocks**	Output	The number of stock objects using this anchor (should be 1)
tag_t * *	**stocks**	Output to UF_*free*	The array of stock objects. Use UF_free to deallocate memory when no longer required.

UF_ROUTE_ask_anchor_stock_data (view source)

Defined in: uf_route.h

Overview
A stock data object has 0 or more anchor objects associated with it to choose from when assigning it as stock. This routine allows you to query an anchor object for any stock data objects which may reference it.

Environment
Internal and External

Required License(s)
gateway

```
int UF_ROUTE_ask_anchor_stock_data
(
    tag_t anchor_tag,
    int * num_stock_data,
    tag_t * * stock_datas
)
```

tag_t	**anchor_tag**	Input	Tag of anchor to query
int *	**num_stock_data**	Output	Count of stock data objects returned
tag_t * *	**stock_datas**	Output to UF_*free*	Array of stock data objects. This must be freed by calling UF_free.

UF_ROUTE_ask_app_view_corners (view source)

Defined in: uf_route.h

Overview

Returns an integer value which is a bit mask indicating the allowable corner types for the given Application View. The bits set in this value can be tested using symbols defined in uf_route.h. These are: UF_ROUTE_AV_CORNERS_BEND, UF_ROUTE_AV_CORNERS_COPE, UF_ROUTE_AV_CORNERS_MITER and UF_ROUTE_AV_CORNERS_SBEND.

Environment

Internal and External

Required License(s)

gateway

```
int UF_ROUTE_ask_app_view_corners
(
    UF_ROUTE_application_view_t * app_view,
    int * curves
)
```

UF_ROUTE_application_view_t *	**app_view**	Input	Address of the Application View structure
int *	**curves**	Output	Flags indicating allowed corner types

---

UF_ROUTE_ask_app_view_curves (view source)

Defined in: uf_route.h

Overview

Returns an integer value which is a bit mask indicating the allowable curve types for the given Application View. The bits set in this value can be tested using symbols defined in uf_route.h. These are:

UF_ROUTE_AV_CURVES_LINES,
UF_ROUTE_AV_CURVES_ARCS, and
UF_ROUTE_AV_CURVES_SPLINES.

Environment
Internal and External

Required License(s)
gateway

```
int UF_ROUTE_ask_app_view_curves
(
    UF_ROUTE_application_view_t * app_view,
    int * curves
)
```

UF_ROUTE_application_view_t *	**app_view**	Input	Address of the Application View structure
int *	**curves**	Output	Flags indicating allowed curve types

UF_ROUTE_ask_app_view_def_stock (view source)

Defined in: uf_route.h

Overview
Inquires the default stock and anchor of the application view. Either or both may be returned as NULL. The default stock only makes sense when the FILE_SELECTION type of part library is being used.

Environment
Internal and External

Required License(s)
gateway

```
int UF_ROUTE_ask_app_view_def_stock
(
    UF_ROUTE_application_view_t * app_view,
    char * * stock,
    char * * anchor
)
```

UF_ROUTE_application_view_t *	**app_view**	Input	Application View to query
char * *	**stock**	Output to UF_*free*	Name of default stock
char * *	**anchor**	Output to UF_*free*	Name of default anchor

---

UF_ROUTE_ask_app_view_def_style (view source)

Defined in: uf_route.h

Overview
Returns the default stock style defined in the application view. The style could be one of
UF_ROUTE_STYLE_NONE,
UF_ROUTE_STYLE_SOLID, or
UF_ROUTE_STYLE_DETAIL.

Environment
Internal and External

Required License(s)
gateway

```
int UF_ROUTE_ask_app_view_def_style
(
    UF_ROUTE_application_view_t * app_view,
    int * style
)
```

UF_ROUTE_application_view_t *	**app_view**	Input	Address of the Application View structure
int *	**style**	Output	Default stock style

UF_ROUTE_ask_app_view_desc (view source)

Defined in: uf_route.h

Overview
Returns a string containing the description of the Application View.

Environment
Internal and External

Required License(s)
gateway

```
int UF_ROUTE_ask_app_view_desc
(
    UF_ROUTE_application_view_t * app_view,
    char * * description
)
```

UF_ROUTE_application_view_t *	**app_view**	Input	Address of Application View structure
char * *	**description**	Output	Application View description.

UF_ROUTE_ask_app_view_ext_plib (view source)

Defined in: uf_route.h

Overview

Returns strings containing the shared library path and the entry point function in the shared library.

Environment

Internal and External

Required License(s)

gateway

int UF_ROUTE_ask_app_view_ext_plib
(
UF_ROUTE_application_view_t * app_view,
char * * library,
char * * entry
)

UF_ROUTE_application_view_t *	**app_view**	Input	Address of Application View structure
char * *	**library**	Output to UF_*free*	Name of external library.
char * *	**entry**	Output to UF_*free*	Name of entry point within the given external shared library

---

UF_ROUTE_ask_app_view_fab_charx (view source)

Defined in: uf_route.h

Overview

Returns the number of fabrication characteristics and a structure containing the characteristic types, values and titles. These characteristics must be set for any fabrication created out of a Routing assembly.

This routine is deprecated. Please use NXOpen::Preferences::RoutingApplicationView::GetFabricationCharacteristics.

Environment

Internal and External

Required License(s)

gateway

int UF_ROUTE_ask_app_view_fab_charx
(
UF_ROUTE_application_view_t * app_view,
int * num_charx,
UF_ROUTE_char_desc_p_t * entry

)

UF_ROUTE_application_view_t *	**app_view**	Input	Address of Application View structure
int *	**num_charx**	Output	Number of fabrication characteristics.
UF_ROUTE_char_desc_p_t *	**entry**	Output to UF_*free*	Structure containing the characteristic types, titles and values.

UF_ROUTE_ask_app_view_filename (view source)

Defined in: uf_route.h

Overview
Returns a string containing the filename of the application view.

Environment
Internal and External

Required License(s)
gateway

int UF_ROUTE_ask_app_view_filename
(
UF_ROUTE_application_view_t * app_view,
char * * filename
)

UF_ROUTE_application_view_t *	**app_view**	Input	Address of Application View structure
char * *	**filename**	Output	Filename of the application view.

UF_ROUTE_ask_app_view_name (view source)

Defined in: uf_route.h

Overview
Returns a string containing the name of the Application View.

Environment
Internal and External

Required License(s)
gateway

```
int UF_ROUTE_ask_app_view_name
(
    UF_ROUTE_application_view_t * app_view,
    char * * name
)
```

UF_ROUTE_application_view_t *	**app_view**	Input	Address of Application View structure
char * *	**name**	Output	Name of the Application View.

UF_ROUTE_ask_app_view_opt_charx (view source)

Defined in: uf_route.h

Overview

Returns the intersection of the optional stock characteristics and optional part characteristics for the current discipline of the application view.

This routine is deprecated. Please use NXOpen::Preferences::RoutingApplicationView::GetOptionalCharacteristics.

Environment

Internal and External

Required License(s)

gateway

```
int UF_ROUTE_ask_app_view_opt_charx
(
    UF_ROUTE_application_view_t * app_view,
    int * num_charx,
    UF_ROUTE_char_desc_p_t * charx
)
```

UF_ROUTE_application_view_t *	**app_view**	Input	Address of Application View structure
int *	**num_charx**	Output	Number of optional characteristics
UF_ROUTE_char_desc_p_t *	**charx**	Output to UF_*free*	Optional characteristics

---

UF_ROUTE_ask_app_view_plib_type (view source)

Defined in: uf_route.h

Overview

Inquires the type of part library used by the given application view.

Environment

Internal and External

Required License(s)

gateway

```
int UF_ROUTE_ask_app_view_plib_type
(
    UF_ROUTE_application_view_t * app_view,
    int * type
)
```

UF_ROUTE_application_view_t *	**app_view**	Input	Application view to query
int *	**type**	Output	Part library type: UF_ROUTE_LIBRARY_TYPE_FILE_SELECT UF_ROUTE_LIBRARY_TYPE_VIEW UF_ROUTE_LIBRARY_TYPE_EXTERNAL

UF_ROUTE_ask_app_view_plib_view (view source)

Defined in: uf_route.h

Overview

Inquires the part library view used by the given application view. This should only be called if the part library type is UF_ROUTE_LIBRARY_TYPE_VIEW.

Environment

Internal and External

See Also

UF_ROUTE_ask_app_view_plib_type

Required License(s)

gateway

```
int UF_ROUTE_ask_app_view_plib_view
(
    UF_ROUTE_application_view_t * app_view,
    UF_EPLIB_part_library_view_p_t * part_lib_view
)
```

UF_ROUTE_application_view_t *	**app_view**	Input	Application view to query
UF_EPLIB_part_library_view_p_t *	**part_lib_view**	Output to UF_*free*	Part library view used by application view.

UF_ROUTE_ask_app_view_req_charx (view source)

Defined in: uf_route.h

Overview

Returns the intersection of the required stock characteristics and required part characteristics for the current discipline of the application view.

This routine is deprecated. Please use NXOpen::Preferences::RoutingApplicationView::GetRequiredCharacteristics.

Environment

Internal and External

Required License(s)

gateway

```
int UF_ROUTE_ask_app_view_req_charx
(
    UF_ROUTE_application_view_t * app_view,
    int * num_charx,
    UF_ROUTE_char_desc_p_t * charx
)
```

UF_ROUTE_application_view_t *	**app_view**	Input	Address of Application View structure
int *	**num_charx**	Output	Number of required characteristics
UF_ROUTE_char_desc_p_t *	**charx**	Output to UF_*free*	Required characteristics

---

UF_ROUTE_ask_bend_radius (view source)

Defined in: uf_route.h

Overview

Returns the radius of a bend.

Environment

Internal and External

History

Original release was in V15.0.

Required License(s)

gateway

```
int UF_ROUTE_ask_bend_radius
(
    tag_t bend_tag,
    double * radius
)
```

tag_t	**bend_tag**	Input	Tag of bend object to query

double *	**radius**	Output	Bend radius of bend

UF_ROUTE_ask_bend_rcp (view source)

Defined in: uf_route.h

Overview
Inquire of a bend, which RCP it is associated with.

Environment
Internal and External

History
Original release was in V15.0.

Required License(s)
gateway

```
int UF_ROUTE_ask_bend_rcp
(
    tag_t bend_tag,
    tag_t * rcp
)
```

tag_t	**bend_tag**	Input	Tag of bend
tag_t *	**rcp**	Output	Tag of rcp which has bend corner assigned

---

UF_ROUTE_ask_bend_segment (view source)

Defined in: uf_route.h

Overview
Returns the Bend segment object identifier for a given Bend corner.
When a Bend Corner is created at the intersection of two Segments,
via the UF_ROUTE_create_bend_by_radius or
UF_ROUTE_create_bend_by_ratio routine, two new objects are
created. One is the Bend Corner object, which holds the information
about the corner, e.g., the radius, and the other is a Routing Segment
which models the bend or fillet between the two original Segments.
Returns the tag of this bend segment.

Environment
Internal and External

See Also
UF_ROUTE_create_bend_by_radius
UF_ROUTE_create_bend_by_ratio

History

Original release was in V13.0.

Required License(s)

gateway

```
int UF_ROUTE_ask_bend_segment
(
    tag_t bend_obj,
    tag_t * seg_id
)
```

tag_t	**bend_obj**	Input	Object identifier of the Bend corner object
tag_t *	**seg_id**	Output	Object identifier of the segment that corresponds to the Bend.

UF_ROUTE_ask_built_in_path_objs (view source)

Defined in: uf_route.h

Overview

Ask built-in path curves

Environment

Internal and External

History

New in V17

Required License(s)

gateway

```
int UF_ROUTE_ask_built_in_path_objs
(
    tag_t bip,
    int * num_objs,
    tag_t * * objects
)
```

tag_t	**bip**	Input	Tag of built-in path object
int *	**num_objs**	Output	Number of curves in path
tag_t * *	**objects**	Output to UF_*free*	Array of curve tags Free this array using UF_free

---

UF_ROUTE_ask_built_in_paths (view source)

Defined in: uf_route.h

Overview
Ask all built-in paths in a routing part

Environment
Internal and External

History
New in V17

Required License(s)
gateway

```
int UF_ROUTE_ask_built_in_paths
(
    tag_t part,
    int * num_paths,
    tag_t * * paths,
    char * * * bip_names
)
```

tag_t	**part**	Input	Tag of part to be queried
int *	**num_paths**	Output	Number of built-in paths
tag_t * *	**paths**	Output to UF_*free*	Array of built-in path tags Free this array using UF_free
char * * *	**bip_names**	Output to UF_*free*	Array of built-in path names Free this array using UF_free_string_array

---

UF_ROUTE_ask_characteristics (view source)

Defined in: uf_route.h

Overview
Returns the characteristics data of a routing object.

Environment
Internal and External

History
Original release was in V13.0.

Required License(s)
gateway

```
int UF_ROUTE_ask_characteristics
(
    tag_t obj_id,
    int c_type,
    int * charx_count,
    UF_ROUTE_charx_p_t * list
)
```

tag_t	**obj_id**	Input	Object identifier of the routing object.
int	**c_type**	Input	Type of characteristic desired. Valid values are: UF_ROUTE_CHARX_TYPE_INT UF_ROUTE_CHARX_TYPE_REAL UF_ROUTE_CHARX_TYPE_STR UF_ROUTE_CHARX_TYPE_ANY
int *	**charx_count**	Output	Count of the characteristics.
UF_ROUTE_charx_p_t *	**list**	Output to UF_*free*	List of all characteristics. The allocated list must be freed by calling UF_ROUTE_free_charx_array.

UF_ROUTE_ask_charx_env (view source)

Defined in: uf_route.h

Overview
Returns the number of characteristics and the array containing characteristic titles, values and types.

Environment
Internal and External

History
Original release was in V14.0.

Required License(s)
gateway

```
int UF_ROUTE_ask_charx_env
(
    int * num_charx,
    UF_ROUTE_charx_p_t * charx
)
```

int *	**num_charx**	Output	Number of the characteristics.
UF_ROUTE_charx_p_t *	**charx**	Output to UF_*free*	List of all characteristics.

UF_ROUTE_ask_connection_ports (view source)

Defined in: uf_route.h

Overview
Finds the tags of the two ports of a connection object.

Environment

Internal and External

History
Original release was in V14.0.

Required License(s)
gateway

int UF_ROUTE_ask_connection_ports
(
    tag_t conn_tag,
    tag_t ports [2]
)

tag_t	**conn_tag**	Input	Object identifier of the connection.
tag_t	**ports [2]**	Output	Array of the two port tags at this connection.

UF_ROUTE_ask_cross_curves (view source)

Defined in: uf_route.h

Overview
Inquire the curves associated with a cross section.

Environment
Internal and External

History
Original release was in V15.0.

Required License(s)
gateway

int UF_ROUTE_ask_cross_curves
(
    tag_t cross_tag,
    int * num_curves,
    tag_t * * curves
)

tag_t	**cross_tag**	Input	Cross section object to query.
int *	**num_curves**	Output	Number of curves in the cross section.
tag_t * *	**curves**	Output to UF_*free*	Array of cross section curves. Use UF_free to deallocate memory when no longer required.

UF_ROUTE_ask_cross_offsets (view source)

Defined in: uf_route.h

Overview

Returns the cross section offset values. These are the values used to thicken the cross section about the cross section curves.

Environment

Internal and External

History

Original release was in V15.0.

Required License(s)

gateway

```
int UF_ROUTE_ask_cross_offsets
(
    tag_t cross_tag,
    double offsets [ 2 ]
)
```

tag_t	**cross_tag**	Input	Tag of cross section to query.
double	**offsets [2]**	Output	Cross section offset values.

UF_ROUTE_ask_cross_stock_data (view source)

Defined in: uf_route.h

Overview

Returns all stock data objects referencing the given cross section object.

Environment

Internal and External

History

Original release was in V15.0.

Required License(s)

gateway

```
int UF_ROUTE_ask_cross_stock_data
(
    tag_t cross_tag,
    int * num_stock_data,
    tag_t * * stock_data_tags
)
```

tag_t	**cross_tag**	Input	Cross section object to query

int *	**num_stock_data**	Output	Number of stock data objects
tag_t * *	**stock_data_tags**	Output to UF_*free*	Array of stock data objects. Use UF_free to deallocate memory when no longer required.

UF_ROUTE_ask_cross_style (view source)

Defined in: uf_route.h

Overview
Returns the stock style for the specified cross section.

Environment
Internal and External

History
Original release was in V15.0.

Required License(s)
gateway

```
int UF_ROUTE_ask_cross_style
(
    tag_t cross_tag,
    int * style
)
```

tag_t	**cross_tag**	Input	Cross section object to query
int *	**style**	Output	Style this cross section is associated with: UF_ROUTE_STYLE_SIMPLE UF_ROUTE_STYLE_DETAIL

UF_ROUTE_ask_current_app_view (view source)

Defined in: uf_route.h

Overview
Returns the current application view after UF_ROUTE_set_current_app_view has been used to set the application view.

Return
Returns the current application view which may be NULL.

Environment
Internal and External

See Also
UF_ROUTE_set_current_app_view
Please refer to the example

History
Original release was in V13.0.

Required License(s)
gateway

```
UF_ROUTE_application_view_t * UF_ROUTE_ask_current_app_view
(
    void
)
```

UF_ROUTE_ask_harness_comps (view source)

Defined in: uf_route.h

Overview
Ask the tags of the components in the given harness.

Environment
Internal and External

History
Released in V17.0

Required License(s)
gateway

```
int UF_ROUTE_ask_harness_comps
(
    tag_t harness,
    int * num_comps,
    tag_t * * comps
)
```

tag_t	**harness**	Input	, the harness to query
int *	**num_comps**	Output	, the number of components
tag_t * *	**comps**	Output to UF_*free*	, the array of component tags, free with UF_free

---

UF_ROUTE_ask_harness_wires (view source)

Defined in: uf_route.h

Overview
Ask the wires in the given harness.

Environment

Internal and External

History
Released in V17.0

Required License(s)
gateway

```
int UF_ROUTE_ask_harness_wires
(
    tag_t harness,
    int * num_wires,
    tag_t * * wires
)
```

tag_t	**harness**	Input	, the harness to query
int *	**num_wires**	Output	, the number of wires
tag_t * *	**wires**	Output to UF_*free*	, the array of wire tags, free with UF_free.

UF_ROUTE_ask_heal_pos (view source)

Defined in: uf_route.h

Overview
Locates the rcp position coordinates in context to the work view coordinate system for a standard heal path.

Environment
Internal and External

Required License(s)
gateway

```
int UF_ROUTE_ask_heal_pos
(
    int method,
    double end_pos [ 2 ] [ 3 ] ,
    double end_uvecs [ 2 ] [ 3 ] ,
    double extensions [ 2 ] ,
    int * num_pos,
    double * heal_pos [ 3 ]
)
```

int	**method**	Input	The type of standard heal path to be created.
double	**end_pos [2] [3]**	Input	The position of the ends to be joined by the heal path in work coordinates.
double	**end_uvecs [2] [3]**	Input	Unit vectors associated with the objects being joined. Directed away from the

			object.
double	**extensions [2]**	Input	The length of the extension segments which will be created between end positions and the standard heal path.
int *	**num_pos**	Output	The number of rcps created in the standard heal path.
double *	**heal_pos [3]**	Output to UF_*free*	Array of positions of where the rcps are created in the standard heal path. Must be freed using UF_free.

UF_ROUTE_ask_length_tolerance (view source)

Defined in: uf_route.h

Overview
Return the length tolerance used in the Routing module.

Environment
Internal and External

Required License(s)
gateway

```
int UF_ROUTE_ask_length_tolerance
(
    double * tol
)
```

double *	**tol**	Output	Tolerance used for distance and length comparisons in Routing

---

UF_ROUTE_ask_loaded_bend_tables (view source)

Defined in: uf_route.h

Overview
This function will provide a list of the names of all the currently loaded bend radius tables. Use UF_free_string_array to free the array of names. An error of UF_ROUTE_err_table_not_loaded means that there are no tables loaded in the current application view.

Environment
Internal and External

History
Released in V18.0

Required License(s)
gateway

```
int UF_ROUTE_ask_loaded_bend_tables
(
    int * num_tables,
    char * * * tables
)
```

int *	**num_tables**	Output	number of entries
char * * *	**tables**	Output to UF_*free*	array of table names. This must be freed by calling UF_free_string_array

UF_ROUTE_ask_multiport_strings (view source)

Defined in: uf_route.h

Overview
Queries the virtual port names of the given multiport .

Environment
Internal and External

History
Released in V17.0

Required License(s)
gateway

```
int UF_ROUTE_ask_multiport_strings
(
    tag_t port_tag,
    int * num_strings,
    const char * * * strings
)
```

tag_t	**port_tag**	Input	the tag of the port
int *	**num_strings**	Output	the number of strings
const char * * *	**strings**	Output to UF_*free*	the array of names, don't free the individual strings, just call UF_free(strings);

UF_ROUTE_ask_multiport_tags (view source)

Defined in: uf_route.h

Overview
Gets the tags of the virtual ports associated with the multiport.

Environment
Internal and External

History
Released in V17.0

Required License(s)
gateway

int UF_ROUTE_ask_multiport_tags
(
tag_t port_tag,
int * num_tags,
tag_t * * tags
)

tag_t	**port_tag**	Input	, the tag of the port
int *	**num_tags**	Output	, the number of tags
tag_t * *	**tags**	Output to UF_*free*	, the array of virtual ports, use UF_free to free up the array.

UF_ROUTE_ask_multiport_terminals (view source)

Defined in: uf_route.h

Overview
Query the terminal and virtual ports of the given multiport.

Environment
Internal and External

History
Originally released in V16.0

Required License(s)
gateway

int UF_ROUTE_ask_multiport_terminals
(
tag_t multi,
int * num_terms,
tag_t * * terms,
int * num_virts,
const char * * * virts
)

tag_t	**multi**	Input	Multiport to query
int *	**num_terms**	Output	Number of terminal ports

tag_t * *	**terms**	Output to UF_*free*	Array of terminal port tags. This must be freed by calling UF_free.
int *	**num_virts**	Output	Number of virtual ports
const char * * *	**virts**	Output to UF_*free*	Array of virtual port identifiers. This array must be freed by calling UF_free.

UF_ROUTE_ask_multiport_termname (view source)

Defined in: uf_route.h

Overview
Queries the name of the charx of which virtual port names are considered values.

Environment
Internal and External

History
Released in V17.0

Required License(s)
gateway

```
int UF_ROUTE_ask_multiport_termname
(
    tag_t port_tag,
    char * * name
)
```

tag_t	**port_tag**	Input	the tag of the port to query
char * *	**name**	Output to UF_*free*	the name of the virtual port, free using UF_free

---

UF_ROUTE_ask_obj_bend_radius (view source)

Defined in: uf_route.h

Overview
Returns the "bend corner" radius when supplied with any one of the following:
. The RCP at which the Bend was created.
. The Bend Corner object id obtained after calling one of the UF_ROUTE_create_bend routines.
. The segment created as a result of the Bend Corner.

Returns
True = If the obj_id is a valid bend object.
False = invalid object.

Environment
Internal and External

History
Original release was in V14.0.

Required License(s)
gateway

logical UF_ROUTE_ask_obj_bend_radius
(
tag_t obj_id,
double * radius
)

tag_t	**obj_id**	Input	Object identifier to interrogate. Can be a RCP, a Bend segment, or a Corner (only bend corner) object.
double *	**radius**	Output	Radius at the Bend Corner.

UF_ROUTE_ask_obj_corner_info (view source)

Defined in: uf_route.h

Overview
Returns the "corner" information such as type, RCP id and the Corner object id when supplied with any one of the following:
The RCP at which the corner was created.
The Corner object id obtained after calling one of the UF_ROUTE_create_bend or UF_ROUTE_create_miter routines.
The Segment created as a result of the Bend corner. Miter corner does not create a segment. The input may be an existing RCP, a Bend segment or a Corner object.

Return
TRUE object is some kind of corner
FALSE otherwise.

Environment
Internal and External

History
Original release was in V13.0.

Required License(s)
gateway

logical UF_ROUTE_ask_obj_corner_info
(
tag_t obj_id,
int * crn_typ,
tag_t * crn_rcp,
tag_t * crn_obj
)

tag_t	**obj_id**	Input	Object identifier to interrogate. Can be an RCP, a Bend Segment, or a Corner (Bend or Miter) object .
int *	**crn_typ**	Output	Corner type such as UF_ROUTE_CORNER_NONE UF_ROUTE_CORNER_BEND UF_ROUTE_CORNER_MITER
tag_t *	**crn_rcp**	Output	Object Identifier of the RCP at this Corner only, else NULL_TAG
tag_t *	**crn_obj**	Output	Object Identifier of the Corner object. This is equal to obj_id if the input is a Corner. If the input is an RCP or Bend Segment this is the Corner object to that RCP or segment

---

UF_ROUTE_ask_object_port (view source)

Defined in: uf_route.h

Overview
Use this function to find the ports attached to an object. The object can be a segment, stock, port, rcp, or part. Success is indicated by a non-zero count of ports. The function always returns ERROR_OK. The array of ports must be freed by the user with UF_free.

Environment
Internal and External

History
Released in V18.0

Required License(s)
gateway

```
int UF_ROUTE_ask_object_port
(
    tag_t object,
    int * num_ports,
    tag_t * * ports
)
```

tag_t	**object**	Input	object
int *	**num_ports**	Output	count of ports
tag_t * *	**ports**	Output to UF_*free*	array of ports, this must be freed by calling UF_free

---

UF_ROUTE_ask_object_stock (view source)

Defined in: uf_route.h

Overview

Returns the stock tag assigned to the Segment or attached to the Stock Port or the stock associated with the Stock Solid feature. The input may be a Port of a stock, a Segment, Stock Solid feature tag or a Curve used to define the segment that is assigned the stock. If no stock is assigned to the segment a NULL_TAG is returned.

Environment

Internal and External

History

Original release was in V14.0.

Required License(s)

gateway

```
int UF_ROUTE_ask_object_stock
(
    tag_t obj_id,
    tag_t * stock
)
```

tag_t	**obj_id**	Input	Object identifier to interrogate. Can be a Port of a Stock, a Segment, the Stock solid feature or a Curve object.
tag_t *	**stock**	Output	Stock tag attached to the input object else NULL_TAG.

UF_ROUTE_ask_part_duplicate_rcps (view source)

Defined in: uf_route.h

Overview

Find all the duplicate RCPs occurring in a given part. If part tag is a NULL_TAG, the current work part will be searched for duplicate RCPs. An output flag will indicate if any duplicate RCPs were found in the part. If duplicate RCPs are found, the output of lists of duplicate RCPs (at each location) will be appropriately populated.
Returns an error code if any error occurs in the function.

E.g. Consider a part containing 10 unique RCPs with tags 10, 20, 30,..., 90, 100. If 10, 20, 30 were moved such that 10 & 60 now lie at the same location and 20 & 30 ended up at the same location as 70, we would have the following structure of the output from this function.

found_duplicates - TRUE
num_part_dup_rcp_lists - 2
part_dup_rcp_lists - [10 60
20 30 70]

Environment

Internal and External

History

New in V17

Required License(s)

gateway

int UF_ROUTE_ask_part_duplicate_rcps
(
tag_t part,
double tolerance,
logical * found_duplicates,
int * num_part_dup_rcp_lists,
UF_ROUTE_tag_list_p_t * * part_dup_rcp_lists
)

tag_t	**part**	Input	Part tag
double	**tolerance**	Input	Tolerance to be used to determine duplicates. Defaults to Internal tolerance limits if <= 0
logical *	**found_duplicates**	Output	Indicates if duplicate RCPs were found
int *	**num_part_dup_rcp_lists**	Output	Number of locations where duplicate RCPs were found
UF_ROUTE_tag_list_p_t * *	**part_dup_rcp_lists**	Output to UF_*free*	Function_to_free = UF_ROUTE_free_array_of_tag_lists Lists of duplicate RCP tags found at each of the above locations. Free using UF_ROUTE_free_array_of_tag_lists.

UF_ROUTE_ask_part_duplicate_segs (view source)

Defined in: uf_route.h

Overview

Find all the duplicate segments originating in duplicate RCPs that
occur in a given part. This routine cannot find duplicate segments that have
the same two end rcps. If part tag is a NULL_TAG, the current work part
will be searched for duplicate segments.
An output flag will indicate if any duplicate segments were found in the
part. If duplicate segments are found, the output of lists of duplicate
segments (at each location) will be appropriately populated.
Returns an error code if any error occurs in the function.

E.g. Consider a part containing 10 unique segments with tags
10, 20, 30,..., 90, 100. If 10, 20, 30 were moved such that
10 & 60 now lie at the same location and 20 & 30 ended up
at the same location as 70, we would have the following
structure of the output from this function.

found_duplicates - TRUE
num_part_dup_seg_lists - 2
part_dup_seg_lists - [10 60
20 30 70]

Environment
Internal and External

History
New in V17

Required License(s)
gateway

```
int UF_ROUTE_ask_part_duplicate_segs
(
    tag_t part,
    double tolerance,
    logical * found_duplicates,
    int * num_part_dup_seg_lists,
    UF_ROUTE_tag_list_p_t * * part_dup_seg_lists
)
```

tag_t	**part**	Input	Part tag
double	**tolerance**	Input	Tolerance to be used to determine duplicates. Defaults to Internal tolerance limits if <= 0
logical *	**found_duplicates**	Output	Indicates if duplicate segments were found
int *	**num_part_dup_seg_lists**	Output	Number of locations where duplicate segments were found
UF_ROUTE_tag_list_p_t * *	**part_dup_seg_lists**	Output to UF_*free*	Function_to_free = UF_ROUTE_free_array_of_tag_lists Lists of duplicate segment tags found at each of the above locations. Free using UF_ROUTE_free_array_of_tag_lists.

UF_ROUTE_ask_part_num_rcps (view source)

Defined in: uf_route.h

Overview
Find the number of RCPs occurring in a given part. If the part tag is a NULL_TAG , the current work part will taken as the default. Returns an error code if any error occurs in the function.

Environment

Internal and External

History

New in V17

Required License(s)

gateway

```
int UF_ROUTE_ask_part_num_rcps
(
    tag_t part,
    int * num_part_rcps
)
```

tag_t	**part**	Input	Part tag, can be a NULL_TAG
int *	**num_part_rcps**	Output	Number of RCPs in the part

UF_ROUTE_ask_part_num_segs (view source)

Defined in: uf_route.h

Overview

Find the number of segments occurring in a given part. If the part tag is a NULL_TAG , the current work part will taken as the default.
Returns an error code if any error occurs in the function.

Environment

Internal and External

History

New in V17

Required License(s)

gateway

```
int UF_ROUTE_ask_part_num_segs
(
    tag_t part,
    int * num_part_segs
)
```

tag_t	**part**	Input	Part tag, can be a NULL_TAG
int *	**num_part_segs**	Output	Number of segments in the part

---

UF_ROUTE_ask_part_occ_ports (view source)

Defined in: uf_route.h

Overview
Returns all the port occurrences on the given part occurrence.

Environment
Internal and External

History
Original release was in V14.0.

Required License(s)
gateway

```
int UF_ROUTE_ask_part_occ_ports
(
    tag_t part_tag,
    int * num_ports,
    tag_t * * ports
)
```

tag_t	**part_tag**	Input	Object identifier of a part occurrence.
int *	**num_ports**	Output	Number of port occurrences in the given part occurrence.
tag_t * *	**ports**	Output to UF_*free*	Tags of port occurrences of the part occurrence else NULL_TAG. Allocated array must be freed with UF_free.

UF_ROUTE_ask_part_part_type (view source)

Defined in: uf_route.h

Overview
Determines the Routing type of the given part. The type returned will be one of:
UF_ROUTE_PART_TYPE_PART
UF_ROUTE_PART_TYPE_STOCK
UF_ROUTE_PART_TYPE_FABRICATION,
or UF_ROUTE_PART_TYPE_UNKNOWN if the part is not a Routing part or if the part is not loaded.

Environment
Internal and External

See Also
UF_ROUTE_is_part_fabrication

History
Original release was in V14.0.

Required License(s)
gateway

```
int UF_ROUTE_ask_part_part_type
(
        tag_t r_part,
        int * type
    )
```

tag_t	**r_part**	Input	The tag of part or part occurrence to be inquired.
int *	**type**	Output	The type of Routing part

UF_ROUTE_ask_part_rcps (view source)

Defined in: uf_route.h

Overview
Find the tags of all RCPs occurring in a given part. If the part tag is a NULL_TAG , the current work part will taken as the default. Returns an error code if any error occurs in the function.

Environment
Internal and External

History
New in V17

Required License(s)
gateway

```
    int UF_ROUTE_ask_part_rcps
    (
        tag_t part,
        int * num_part_rcps,
        tag_t * * part_rcps
    )
```

tag_t	**part**	Input	Part tag, can be a NULL_TAG
int *	**num_part_rcps**	Output	Number of RCPs in the part, 0 if there are none.
tag_t * *	**part_rcps**	Output to UF_*free*	Array of part RCP tags, NULL if there are none. Free using UF_free.

UF_ROUTE_ask_part_search_path (view source)

Defined in: uf_route.h

Overview
Retrieves the tag of a search path.

Environment
Internal and External

See Also

UF_DIRPATH_ask_dirs
UF_DIRPATH_ask_dir_count
UF_DIRPATH_ask_dir_index
UF_DIRPATH_ask_nth_dir
UF_DIRPATH_ask_prev_dir
UF_DIRPATH_ask_curr_dir
UF_DIRPATH_ask_next_dir

History
Original release was in V15.0.

Required License(s)
gateway

```
int UF_ROUTE_ask_part_search_path
(
    tag_t * path
)
```

tag_t *	**path**	Output	tag of search path or NULL_tag

UF_ROUTE_ask_part_segs (view source)

Defined in: uf_route.h

Overview
Find the tags of all segments occurring in a given part. If the part tag is
a NULL_TAG , the current work part will taken as the default.
Returns an error code if any error occurs in the function.

Environment
Internal and External

History
New in V17

Required License(s)
gateway

```
int UF_ROUTE_ask_part_segs
(
    tag_t part,
    int * num_part_segs,
    tag_t * * part_segs
)
```

tag_t	**part**	Input	Part tag, can be a NULL_TAG
int *	**num_part_segs**	Output	Number of segments in the part, 0 if there are none.
tag_t * *	**part_segs**	Output to UF_*free*	Array of part segment tags, NULL if there are none. Free using UF_free.

UF_ROUTE_ask_places_transform (view source)

Defined in: uf_route.h

Overview

This routine returns the origin and CSYS matrix data associated with a Routing "placement" solution returned by UF_ROUTE_solve_places.

First use UF_ROUTE_solve_places to get solutions, then call this routine to obtain the origin and csys arrays that should be applied to the part instance with UF_ASSEM_reposition_instance.

Environment

Internal and External

Required License(s)

gateway

```
int UF_ROUTE_ask_places_transform
(
    UF_ROUTE_place_solution_p_t places,
    double origin [ 3 ] ,
    double csys_matrix [ 6 ]
)
```

UF_ROUTE_place_solution_p_t	**places**	Input	UF_ROUTE_place_solution_p_t pointer returned by UF_ROUTE_solve_places
double	**origin [3]**	Output	Origin data for UF_ASSEM_reposition_instance
double	**csys_matrix [6]**	Output	CSYS data for UF_ASSEM_reposition_instance

UF_ROUTE_ask_port_align_flag (view source)

Defined in: uf_route.h

Overview

Inquire whether the given port uses an alignment vector.

Environment

Internal and External

History

Original release was in V15.0.

Required License(s)

gateway

```
int UF_ROUTE_ask_port_align_flag
(
    tag_t port_tag,
    logical * flag
)
```

tag_t	**port_tag**	Input	Port to query
logical *	**flag**	Output	Port alignment flag: TRUE = Port uses an alignment vector. FALSE = Port does not use an alignment vector.

UF_ROUTE_ask_port_align_vector (view source)

Defined in: uf_route.h

Overview
Returns the alignment vector of a port.

Environment
Internal and External

See Also
UF_ROUTE_ask_port_align_flag

History
Original release was in V15.0.

Required License(s)
gateway

```
int UF_ROUTE_ask_port_align_vector
(
    tag_t port_tag,
    double vector [ 3 ]
)
```

tag_t	**port_tag**	Input	Tag of port to query
double	**vector [3]**	Output	Alignment vector of port

UF_ROUTE_ask_port_back_extension (view source)

Defined in: uf_route.h

Overview
This function returns the back extension value for a port.

Environment
Internal and External

History
Released in V18.0

Required License(s)
gateway

```
int UF_ROUTE_ask_port_back_extension
(
    tag_t port,
    double* ext
)
```

tag_t	**port**	Input	, the port
double*	**ext**	Output	, the port back extension

UF_ROUTE_ask_port_back_extension_obj (view source)

Defined in: uf_route.h

Overview
This function returns the back extension object for a port.
The returned object is a UF_scalar_type.

Environment
Internal and External

History
Released in V18.0

Required License(s)
gateway

```
int UF_ROUTE_ask_port_back_extension_obj
(
    tag_t port,
    tag_t* ext
)
```

tag_t	**port**	Input	, the port
tag_t*	**ext**	Output	, the port back extension object

UF_ROUTE_ask_port_charx (view source)

Defined in: uf_route.h

Overview

Ask a specific charx value from a port. If the information is not found attached to the port, the part is queried.

Environment
Internal and External

History
Released in V17.0

Required License(s)
gateway

```
int UF_ROUTE_ask_port_charx
(
    char * charx_name,
    int expected_type,
    tag_t port_tag,
    UF_ROUTE_charx_p_t desired_charx
)
```

char *	**charx_name**	Input	, the name of the charx to get
int	**expected_type**	Input	, the expected type of the charx
tag_t	**port_tag**	Input	, the tag of the port to query
UF_ROUTE_charx_p_t	**desired_charx**	Output	, the charx value returned

---

UF_ROUTE_ask_port_clock_increment (view source)

Defined in: uf_route.h

Overview
This function returns the clock angle increment value for a port.
The clocking increment value on a port is used to determine the valid clocking angle values. The valid clocking angles are the angle values that can be used for setting the angle between the rotation vectors of two ports that are connected to each other.

Environment
Internal and External

History
Released in V18.0

Required License(s)
gateway

```
int UF_ROUTE_ask_port_clock_increment
(
    tag_t port,
    double * increment
)
```

tag_t	**port**	Input	The port to query.
double *	**increment**	Output	The increment of the clock angle.

UF_ROUTE_ask_port_conn_port (view source)

Defined in: uf_route.h

Overview
Inquires the port to which a given port is connected.

Return
Return code:
TRUE = curr_port is connected
FALSE = curr_port is not connected

Environment
Internal and External

History
Original release was in V15.0.

Required License(s)
gateway

```
logical UF_ROUTE_ask_port_conn_port
(
    tag_t curr_port,
    tag_t * connected_port
)
```

tag_t	**curr_port**	Input	Port to query.
tag_t *	**connected_port**	Output	Tag of connected port or NULL_TAG.

UF_ROUTE_ask_port_connected_port (view source)

Defined in: uf_route.h

Overview
Finds the port connected to a given port within the work part.

Environment
Internal and External

History
Released in V17.0

Required License(s)
gateway

```
int UF_ROUTE_ask_port_connected_port
(
    tag_t curr_port,
    tag_t * connected_port,
    logical * connected
)
```

tag_t	**curr_port**	Input	the current port or port occ
tag_t *	**connected_port**	Output	the connected port or NULL_TAG
logical *	**connected**	Output	true if current port is connected

---

UF_ROUTE_ask_port_connection (view source)

Defined in: uf_route.h

Overview
Returns the tag of the connection object of the port. If this port is not connected, NULL_TAG is returned.

Environment
Internal and External

History
Original release was in V14.0.

Required License(s)
gateway

```
int UF_ROUTE_ask_port_connection
(
    tag_t port_tag,
    tag_t * connection
)
```

tag_t	**port_tag**	Input	Object identifier of the port.
tag_t *	**connection**	Output	Connection tag at this port else if the port is not connected then NULL_TAG.

UF_ROUTE_ask_port_cut_back_length (view source)

Defined in: uf_route.h

Overview
Looks at characteristics of given port to determine
the cut back length. If cut back length is not found, look at component of

the port for same characteristic.

Environment
Internal and External

History
Released in V17.0

Required License(s)
gateway

```
int UF_ROUTE_ask_port_cut_back_length
(
    tag_t port,
    double * cut_back_length
)
```

tag_t	**port**	Input	Tag of port beings asked
double *	**cut_back_length**	Output	Cut back length

UF_ROUTE_ask_port_engage_obj (view source)

Defined in: uf_route.h

Overview
Inquires the associative scalar object which defines the engagement distance of a port.

Environment
Internal and External

See Also
UF_SO_ask_double_of_scalar
UF_SO_set_double_of_scalar

History
Original release was in V15.0.

Required License(s)
gateway

```
int UF_ROUTE_ask_port_engage_obj
(
    tag_t port_tag,
    tag_t * engage_obj
)
```

tag_t	**port_tag**	Input	Tag of port
tag_t *	**engage_obj**	Output	Tag of scalar object defining the engagement distance.

UF_ROUTE_ask_port_engaged_pos (view source)

Defined in: uf_route.h

Overview
Inquires the position of the port taking into account the engagement distance.

Environment
Internal and External

History
Original release was in V15.0.

Required License(s)
gateway

```
int UF_ROUTE_ask_port_engaged_pos
(
    tag_t port,
    double position [ 3 ]
)
```

tag_t	**port**	Input	Tag of port to query.
double	**position [3]**	Output	The engaged position of the port.

UF_ROUTE_ask_port_engagement (view source)

Defined in: uf_route.h

Overview
Inquires the engagement distance of a port, i.e. the distance behind the port that another fitting or stock may engage.

Environment
Internal and External

History
Original release was in V15.0.

Required License(s)
gateway

```
int UF_ROUTE_ask_port_engagement
(
    tag_t port_tag,
    double * distance
)
```

tag_t	**port_tag**	Input	Tag of port to query.
double *	**distance**	Output	Engagement distance

UF_ROUTE_ask_port_extension (view source)

Defined in: uf_route.h

Overview
This function returns the extension value for a port.

Environment
Internal and External

History
Released in V18.0

Required License(s)
gateway

```
int UF_ROUTE_ask_port_extension
(
    tag_t port,
    double* ext
)
```

tag_t	**port**	Input	, the port
double*	**ext**	Output	, the port extension

UF_ROUTE_ask_port_extension_obj (view source)

Defined in: uf_route.h

Overview
This function returns the extension object for a port.
The returned object is a UF_scalar_type.

Environment
Internal and External

History
Released in V18.0

Required License(s)
gateway

```
int UF_ROUTE_ask_port_extension_obj
(
    tag_t port,
    tag_t* ext
)
```

tag_t	**port**	Input	, the port
tag_t*	**ext**	Output	, the port extension object

UF_ROUTE_ask_port_lock_info (view source)

Defined in: uf_route.h

Overview

This function returns information about a lock on a part occurrence given the from or to (child or parent) port occurrence that was used to create the lock.

If an error occurs, UF_ROUTE_err_invalid_port_mate will be returned.

Environment

Internal and External

History

Released in V18.0

Required License(s)

gateway

```
int UF_ROUTE_ask_port_lock_info
(
    tag_t port_occ,
    logical * is_locked,
    logical * is_rotation_locked,
    logical * is_from_port
)
```

tag_t	**port_occ**	Input	The FROM or TO port occurrence
logical *	**is_locked**	Output	TRUE if given port participates in a lock
logical *	**is_rotation_locked**	Output	TRUE if rotation is locked
logical *	**is_from_port**	Output	TRUE if given port is the FROM port

---

UF_ROUTE_ask_port_multiport (view source)

Defined in: uf_route.h

Overview

Find the multiport associated with the given port.
If given port is a multiport, return itself. If its

a terminal port, return its multiport.

Environment
Internal and External

History
Released in V17.0

Required License(s)
gateway

```
int UF_ROUTE_ask_port_multiport
(
    tag_t port,
    tag_t * multi
)
```

tag_t	**port**	Input	, the port to query
tag_t *	**multi**	Output	, the tag of the multiport associated with the port

UF_ROUTE_ask_port_occ_of_port (view source)

Defined in: uf_route.h

Overview
Returns the tag of the port occurrence of given extract port
If the port tag given is an occurrence, NULL_TAG is returned

Environment
Internal and External

History
Original release was in NX3

Required License(s)
gateway

```
int UF_ROUTE_ask_port_occ_of_port
(
    tag_t port_tag,
    tag_t * port_occ
)
```

tag_t	**port_tag**	Input	Object identifier of the port.
tag_t *	**port_occ**	Output	Corresponding Occurrence tag of this port

UF_ROUTE_ask_port_on_segment (view source)

Defined in: uf_route.h

Overview
Inquires the port derived by the segment and the given end index.

Environment
Internal and External

See Also
UF_ROUTE_ask_segment_end_idx

History
Original release was in V15.0.

Required License(s)
gateway

```
int UF_ROUTE_ask_port_on_segment
(
    tag_t segment,
    int segend,
    tag_t * port
)
```

tag_t	**segment**	Input	Segment to query.
int	**segend**	Input	Segment end index.
tag_t *	**port**	Output	Port derived by the segment at the given end.

UF_ROUTE_ask_port_part_occ (view source)

Defined in: uf_route.h

Overview
Returns the tag of the part_occurrence containing the given port occurrence. If the port tag given is not an occurrence, NULL_TAG is returned.

Environment
Internal and External

History
Original release was in V14.0.

Required License(s)
gateway

```
int UF_ROUTE_ask_port_part_occ
(
    tag_t port_tag,
    tag_t * part_occ
```

)

tag_t	**port_tag**	Input	Object identifier of the port.
tag_t *	**part_occ**	Output	Occurrence tag of the part to which this port belongs.

UF_ROUTE_ask_port_position (view source)

Defined in: uf_route.h

Overview
Returns the position of a port.

Environment
Internal and External

History
Original release was in V15.0.

Required License(s)
gateway

```
int UF_ROUTE_ask_port_position
(
    tag_t port_tag,
    double position [ 3 ]
)
```

tag_t	**port_tag**	Input	Tag of port to query.
double	**position [3]**	Output	Position of port.

UF_ROUTE_ask_port_rotate_flag (view source)

Defined in: uf_route.h

Overview
Inquires whether the given port uses a rotation vector.

Environment
Internal and External

History
Original release was in V15.0.

Required License(s)
gateway

```
int UF_ROUTE_ask_port_rotate_flag
(
    tag_t port_tag,
    logical * flag
)
```

tag_t	**port_tag**	Input	Port to query.
logical *	**flag**	Output	Port rotation flag.

UF_ROUTE_ask_port_rotate_vector (view source)

Defined in: uf_route.h

Overview
Returns the rotation vector of a port.

Environment
Internal and External

See Also
UF_ROUTE_ask_port_rotate_flag

History
Original release was in V15.0.

Required License(s)
gateway

```
int UF_ROUTE_ask_port_rotate_vector
(
    tag_t port_tag,
    double vector [ 3 ]
)
```

tag_t	**port_tag**	Input	Tag of port to query.
double	**vector [3]**	Output	rotation vector of port.

UF_ROUTE_ask_port_segment (view source)

Defined in: uf_route.h

Overview
If the port position and alignment are derived from a segment, return that segment.

Environment
Internal and External

History

Original release was in V15.0.

Required License(s)

gateway

int UF_ROUTE_ask_port_segment
(
tag_t port_tag,
tag_t * segment
)

tag_t	**port_tag**	Input	Tag of port to query.
tag_t *	**segment**	Output	Tag of segment.

UF_ROUTE_ask_port_stock (view source)

Defined in: uf_route.h

Overview

Returns the tag of the stock object to which the port is attached. If the port does not belong to a stock, NULL_TAG is returned. Only connection ports are considered, not fixture ports.

Environment

Internal and External

History

Original release was in V14.0.

Required License(s)

gateway

int UF_ROUTE_ask_port_stock
(
tag_t port_tag,
tag_t * stock_tag
)

tag_t	**port_tag**	Input	Object identifier of the port.
tag_t *	**stock_tag**	Output	Tag of the stock object.

UF_ROUTE_ask_port_terminal_ports (view source)

Defined in: uf_route.h

Overview

Find terminal ports associated with the given port.
If given port is a multiport, return all terminal

ports. If its a terminal port, return itself. If the
input port is an occurrence, return occurrences of
terminal ports in given ports part occurrence tree.

Environment

Internal and External

History

Released in V17.0

Required License(s)

gateway

```
int UF_ROUTE_ask_port_terminal_ports
(
    tag_t port,
    int * num_terms,
    tag_t * * terms
)
```

tag_t	**port**	Input	, the port to query
int *	**num_terms**	Output	, the number of associated terminal ports
tag_t * *	**terms**	Output to UF_*free*	the array of terminal ports, use UF_free to free up the array.

UF_ROUTE_ask_rcp_at_term_port (view source)

Defined in: uf_route.h

Overview

This function will query a terminal port to find out
the tag of the rcp located at its position.
Three methods are used. First the direct derivation
is checked. Second, the method to identify terminal
wires is used (Smart Point -> BCURVE -> ROUTE_segment
-> segment_ends). Last, if the terminal port is at
the same location of a multiport, the multiport is
queried through the use of ES_ROUTE_ask_rcp_on_port

Environment

Internal and External

History

Released in V17.0

Required License(s)

gateway

```
int UF_ROUTE_ask_rcp_at_term_port
(
    tag_t port,
    tag_t * rcp,
    logical* at
```

)

tag_t	**port**	Input	the terminal port to query
tag_t *	**rcp**	Output	the rcp at the terminal port
logical*	**at**	Output	true if there is an rcp at the term port

---

UF_ROUTE_ask_rcp_corner (view source)

Defined in: uf_route.h

Overview
Returns the corner assigned to the RCP. If no corner is assigned, then NULL_TAG is returned.

Environment
Internal and External

Required License(s)
gateway

```
int UF_ROUTE_ask_rcp_corner
(
    tag_t rcp_tag,
    tag_p_t corner
)
```

tag_t	**rcp_tag**	Input	Tag of Route Control Point.
tag_p_t	**corner**	Output	Tag of corner assigned to the RCP (or NULL_TAG).

UF_ROUTE_ask_rcp_on_port (view source)

Defined in: uf_route.h

Overview
Inquires about any RCP that is associated with the port. This can be either from one of the segment ends of the segment from which the port is derived, or from an RCP that is derived from this port.

Environment
Internal and External

Required License(s)
gateway

```
int UF_ROUTE_ask_rcp_on_port
(
    tag_t port_tag,
    tag_t * rcp_tag
)
```

tag_t	**port_tag**	Input	Tag of port to query.
tag_t *	**rcp_tag**	Output	Tag of RCP (or NULL_TAG).

UF_ROUTE_ask_rcp_ports (view source)

Defined in: uf_route.h

Overview
Finds all the ports connected to a given RCP.

Environment
Internal and External

Required License(s)
gateway

```
int UF_ROUTE_ask_rcp_ports
(
    tag_t rcp,
    int * num_ports,
    tag_t * * ports
)
```

tag_t	**rcp**	Input	Tag of Route Control Point to query.
int *	**num_ports**	Output	Number of ports returned.
tag_t * *	**ports**	Output to UF_*free*	Array of ports. Use UF_free to deallocate memory when no longer required.

UF_ROUTE_ask_rcp_position (view source)

Defined in: uf_route.h

Overview
Obtains the position of the RCP in absolute coordinate system.

Environment
Internal and External

Required License(s)
gateway

```
int UF_ROUTE_ask_rcp_position
(
    tag_t rcp_id,
    double rcp_pos [ 3 ]
)
```

tag_t	**rcp_id**	Input	Object Identifier of the RCP to inquire
double	**rcp_pos [3]**	Output	Position of the RCP in absolute csys.

UF_ROUTE_ask_rcp_segments (view source)

Defined in: uf_route.h

Overview
Inquires the segments attached to a given RCP.

Environment
Internal and External

Required License(s)
gateway

```
int UF_ROUTE_ask_rcp_segments
(
    tag_t rcp_tag,
    int * num_segs,
    tag_t * * segments
)
```

tag_t	**rcp_tag**	Input	Tag of Route Control Point to query
int *	**num_segs**	Output	Number of segments returned
tag_t * *	**segments**	Output to UF_*free*	Array of segments. Use UF_free to deallocate memory when no longer required.

UF_ROUTE_ask_rcp_segs (view source)

Defined in: uf_route.h

Overview
Returns the number of segments and their tags attached to a given RCP.

Environment
Internal and External

Required License(s)
gateway

```
int UF_ROUTE_ask_rcp_segs
(
    tag_t rcp_id,
    int * num_segs,
    tag_t * * segments
)
```

tag_t	**rcp_id**	Input	Object identifier of the RCP.
int *	**num_segs**	Output	Number of segments at this RCP
tag_t * *	**segments**	Output to UF_*free*	Pointer to an allocated array of segments. This must be freed using UF_free.

UF_ROUTE_ask_route_end (view source)

Defined in: uf_route.h

Overview
Asks end object of route.

Environment
Internal and External

History
Released in V17.0

Required License(s)
gateway

```
int UF_ROUTE_ask_route_end
(
    tag_t route,
    tag_t * end
)
```

tag_t	**route**	Input	, the route to query
tag_t *	**end**	Output	, the end object

UF_ROUTE_ask_route_objs (view source)

Defined in: uf_route.h

Overview
Asks route objects of an existing route.

Environment

Internal and External

History
Released in V17.0

Required License(s)
gateway

```
int UF_ROUTE_ask_route_objs
(
    tag_t route,
    int * num_objs,
    tag_t * * objs
)
```

tag_t	**route**	Input	, the route to query
int *	**num_objs**	Output	, the number objects that make up the route
tag_t * *	**objs**	Output to UF_*free*	, the array of objects in the route. Use UF_free to free up returned array.

UF_ROUTE_ask_route_start (view source)

Defined in: uf_route.h

Overview
Asks start object of route.

Environment
Internal and External

History
Released in V17.0

Required License(s)
gateway

```
int UF_ROUTE_ask_route_start
(
    tag_t route,
    tag_t * start
)
```

tag_t	**route**	Input	, the route to query
tag_t *	**start**	Output	, the start object

UF_ROUTE_ask_seg_curve (view source)

Defined in: uf_route.h

Overview

Finds the curve which a segment is "following"

Environment

Internal and External

See Also

UF_ROUTE_create_seg_on_curve

Required License(s)

gateway

```
int UF_ROUTE_ask_seg_curve
(
    tag_t segment,
    tag_t * curve
)
```

tag_t	**segment**	Input	Object identifier of the segment.
tag_t *	**curve**	Output	Object identifier of the follow curve. If the given segment does not follow a curve, NULL_TAG is returned.

UF_ROUTE_ask_seg_rcps (view source)

Defined in: uf_route.h

Overview

Returns the tags of the end RCPs for a given segment.

Environment

Internal and External

Required License(s)

gateway

```
int UF_ROUTE_ask_seg_rcps
(
    tag_t segment,
    tag_t rcp [ 2 ]
)
```

tag_t	**segment**	Input	Object identifier of the segment.
tag_t	**rcp [2]**	Output	Object identifiers of the end RCPs.

UF_ROUTE_ask_segment_bend_crnr (view source)

Defined in: uf_route.h

Overview

Returns the bend corner that this segment represents. The corner is NULL_TAG if segment is not a bend segment.

Environment

Internal and External

History

Original release was in V15.0.

Required License(s)

gateway

```
int UF_ROUTE_ask_segment_bend_crnr
(
    tag_t segment,
    tag_p_t corner
)
```

tag_t	**segment**	Input	Tag of segment to query.
tag_p_t	**corner**	Output	Tag of bend corner (or NULL_TAG).

UF_ROUTE_ask_segment_branch_angle (view source)

Defined in: uf_route.h

Overview

Ask the passed segment its branch angle attribute.

Environment

Internal and External

History

Released in V17.0

Required License(s)

gateway

```
int UF_ROUTE_ask_segment_branch_angle
(
    tag_t segment,
    double * branch_angle
)
```

tag_t	**segment**	Input	, the segment to query

double *	**branch_angle**	Output	, the branch angle.

UF_ROUTE_ask_segment_bundle_stock (view source)

Defined in: uf_route.h

Overview
Ask any stock on a given segment that belongs to specified wire harness.

Environment
Internal and External

History
Released in V17.0

Required License(s)
gateway

```
int UF_ROUTE_ask_segment_bundle_stock
(
    tag_t segment,
    tag_t harness,
    tag_t * stock
)
```

tag_t	**segment**	Input	the segment to query
tag_t	**harness**	Input	the harness to query
tag_t *	**stock**	Output	the stock tag

UF_ROUTE_ask_segment_end_idx (view source)

Defined in: uf_route.h

Overview
Inquires whether the given RCP is segment end 0 or segment end 1.
This index is used by some other UF_ROUTE functions.

Environment
Internal and External

History
Original release was in V15.0.

Required License(s)
gateway

```
int UF_ROUTE_ask_segment_end_idx
(
    tag_t segment,
    tag_t end_object,
    int * index
)
```

tag_t	**segment**	Input	Tag of segment to query.
tag_t	**end_object**	Input	Tag of RCP at segment end.
int *	**index**	Output	End index of RCP on segment (0 or 1).

UF_ROUTE_ask_segment_end_pnts (view source)

Defined in: uf_route.h

Overview
Returns the start and end positions of a segment.

Environment
Internal and External

Required License(s)
gateway

```
int UF_ROUTE_ask_segment_end_pnts
(
    tag_t segment,
    double start [ 3 ] ,
    double end [ 3 ]
)
```

tag_t	**segment**	Input	Tag of segment to query.
double	**start [3]**	Output	Position of start of segment.
double	**end [3]**	Output	Position of end of segment.

UF_ROUTE_ask_segment_end_props (view source)

Defined in: uf_route.h

Overview
Returns curve parameters of segment at given end index.

Environment
Internal and External

See Also

UF_ROUTE_ask_segment_end_idx

History
Original release was in V15.0.

Required License(s)
gateway

```
int UF_ROUTE_ask_segment_end_props
(
    tag_t segment,
    int end,
    double * parameter,
    double * norm_parameter,
    double point [ 3 ] ,
    double tangent [ 3 ]
)
```

tag_t	**segment**	Input	Segment to query
int	**end**	Input	End of segment to query
double *	**parameter**	Output	Curve parameter at end
double *	**norm_parameter**	Output	Normalized parameter at end
double	**point [3]**	Output	Position of end
double	**tangent [3]**	Output	Tangent vector at end

UF_ROUTE_ask_segment_int_part (view source)

Defined in: uf_route.h

Overview
Inquires the part (fitting) to which the segment is interior.

Environment
Internal and External

Required License(s)
gateway

```
int UF_ROUTE_ask_segment_int_part
(
    tag_t segment,
    tag_p_t part
)
```

tag_t	**segment**	Input	Tag of segment to query.
tag_p_t	**part**	Output	Tag of part occurrence

UF_ROUTE_ask_segment_int_parts (view source)

Defined in: uf_route.h

Overview
Inquires the parts (fittings) to which the segment is interior.

Environment
Internal and External

Required License(s)
gateway

```
int UF_ROUTE_ask_segment_int_parts
(
    tag_t segment,
    int * num_parts,
    tag_p_t * parts
)
```

tag_t	**segment**	Input	Tag of segment to query.
int *	**num_parts**	Output	Number of part_occurrences
tag_p_t *	**parts**	Output to UF_*free*	Array of part occurrences

UF_ROUTE_ask_segment_length (view source)

Defined in: uf_route.h

Overview
Returns the length of the Segment.

Environment
Internal and External

Required License(s)
gateway

```
int UF_ROUTE_ask_segment_length
(
    tag_t segment,
    double * length
)
```

tag_t	**segment**	Input	Object identifier of the Segment.
double *	**length**	Output	Length of the segment

UF_ROUTE_ask_segment_paths (view source)

Defined in: uf_route.h

Overview
Ask the paths that the segment belongs to.

Environment
Internal and External

History
Released in V17.0

Required License(s)
gateway

```
int UF_ROUTE_ask_segment_paths
(
    tag_t segment,
    int * number_of_paths,
    tag_p_t * paths
)
```

tag_t	**segment**	Input	, the segment to query
int *	**number_of_paths**	Output	, the number of paths
tag_p_t *	**paths**	Output to UF_*free*	, the array of paths

UF_ROUTE_ask_segment_routes (view source)

Defined in: uf_route.h

Overview
Inquire as to the ROUTE_route object(s) associated with the specified Segment. (An empty list can be returned, indicating that no Routes had been assigned to the Segment.)

Environment
Internal and External

History
Released in V17.0

Required License(s)
gateway

```
int UF_ROUTE_ask_segment_routes
(
    tag_t segment,
    int * num_routes,
    tag_t * * routes
)
```

tag_t	**segment**	Input	, segment to query
int *	**num_routes**	Output	, the number of routes
tag_t * *	**routes**	Output to UF_*free*	, the array of routes, use UF_free to free up the array

---

UF_ROUTE_ask_segment_stock (view source)

Defined in: uf_route.h

Overview

Returns the Stock object(s) associated with the specified Segment.
Can return an empty list, which indicates that no Stock was assigned to the Segment.

Environment

Internal and External

Required License(s)

gateway

```
int UF_ROUTE_ask_segment_stock
(
    tag_t segment,
    int * num_stock,
    tag_t * * stock
)
```

tag_t	**segment**	Input	Tag of segment to query.
int *	**num_stock**	Output	Number of stock objects returned.
tag_t * *	**stock**	Output to UF_*free*	Array of stock objects. Use UF_free to deallocate memory when no longer required.

UF_ROUTE_ask_segment_wires (view source)

Defined in: uf_route.h

Overview

Ask the tags of the wires that are associated with the given segment.

Environment

Internal and External

History

Released in V17.0

Required License(s)

gateway

```
int UF_ROUTE_ask_segment_wires
(
    tag_t segment,
    int * num_wires,
    tag_t * * wires
)
```

tag_t	**segment**	Input	, the segment to query
int *	**num_wires**	Output	, the number of wires
tag_t * *	**wires**	Output to UF_*free*	, the array of wires, free with UF_free.

UF_ROUTE_ask_segments_is_path (view source)

Defined in: uf_route.h

Overview

Given a set of segments, this function determines if they comprise a single unique path object. For this to be the case, there must be a 1 to 1 mapping between the segments passed in and the segments in the path.

Each segment will be asked its path. If any of the segments don't belong to a path, the segments cannot be an offset path object. The offset path object that is common to all of the segments is then checked to determine if the number of segments passed in and the number of segments in the path match. If they do not, the set of segments is not a path. Only in the case that all the segments belong to a common offset path object and the number of segments match is the set of segments that path object.

Environment

Internal and External

History

Released in V17.0

Required License(s)

gateway

```
int UF_ROUTE_ask_segments_is_path
(
    int number_of_segments,
    tag_t * segments,
    tag_t path,
    logical * is_path
)
```

int	**number_of_segments**	Input	, the number of segs to eval
tag_t *	**segments**	Input	, the array of segs to eval
tag_t	**path**	Input	, the path tag
logical *	**is_path**	Output	, true if the segments are a path

---

UF_ROUTE_ask_segments_paths (view source)

Defined in: uf_route.h

Overview

Given a set of segments, this function
finds the path objects that are defined entirely using the given
set of segments.

Eg: Multiple Master path objects may be defined using the same
set of segments, but with different creation options. In this case
this routine may be used to make sure we are not duplicating
master paths at the time of creation.

Environment

Internal and External

History

Released in V17.0

Required License(s)

gateway

```
int UF_ROUTE_ask_segments_paths
(
    int num_segments,
    tag_t * segments,
    int * num_paths,
    tag_t * * paths,
    logical * share_path
)
```

int	**num_segments**	Input	, the number of segs in this path
tag_t *	**segments**	Input	, the array of segments
int *	**num_paths**	Output	, the number of paths for these segs

tag_t * *	**paths**	Output to UF_*free*	, the array of paths, use UF_free to free up the array.
logical *	**share_path**	Output	, true if all segs share at least one path

---

UF_ROUTE_ask_stock_anchor (view source)

Defined in: uf_route.h

Overview
Returns the current anchor being used by the stock object.

Environment
Internal and External

History
Original release was in V15.0.

Required License(s)
gateway

```
int UF_ROUTE_ask_stock_anchor
(
    tag_t stock_tag,
    tag_t * anchor
)
```

tag_t	**stock_tag**	Input	Tag of stock to query
tag_t *	**anchor**	Output	Tag of anchor object

UF_ROUTE_ask_stock_body (view source)

Defined in: uf_route.h

Overview
Returns the solid body which represents the stock object. There may be no stock object (NULL_TAG) if the stock style is assigned as simple.

Environment
Internal and External

History
Original release was in V15.0.

Required License(s)
gateway

```
int UF_ROUTE_ask_stock_body
(
    tag_t stock_tag,
    tag_t * body
)
```

tag_t	**stock_tag**	Input	Tag of stock object to query
tag_t *	**body**	Output	Tag of body representing stock object. May be NULL_TAG.

UF_ROUTE_ask_stock_cross_sect (view source)

Defined in: uf_route.h

Overview
Inquires the current cross section object being referenced from the stock data, which is being used to determine the swept body of the stock.

Environment
Internal and External

History
Original release was in V15.0.

Required License(s)
gateway

```
int UF_ROUTE_ask_stock_cross_sect
(
    tag_t stock_tag,
    tag_t * cross_section
)
```

tag_t	**stock_tag**	Input	Tag of stock object
tag_t *	**cross_section**	Output	Tag of cross section object being used by the stock. NULL_TAG if no cross section is being used.

UF_ROUTE_ask_stock_data_anchors (view source)

Defined in: uf_route.h

Overview
Returns the anchors which are associated with a particular stock data.

Environment
Internal and External

History

Original release was in V15.0.

Required License(s)

gateway

```
int UF_ROUTE_ask_stock_data_anchors
(
    tag_t stock_data_tag,
    int * num_anchors,
    tag_t * * anchors
)
```

tag_t	**stock_data_tag**	Input	Tag of stock data object
int *	**num_anchors**	Output	Count of anchors returned.
tag_t * *	**anchors**	Output to UF_*free*	Array of anchors. Use UF_free to deallocate memory when no longer required.

UF_ROUTE_ask_stock_data_cross (view source)

Defined in: uf_route.h

Overview

Returns all the cross section objects associated with a stock data object.

Environment

Internal and External

History

Original release was in V15.0.

Required License(s)

gateway

```
int UF_ROUTE_ask_stock_data_cross
(
    tag_t stock_data_tag,
    int * num_cross_sections,
    tag_t * * cross_sections
)
```

tag_t	**stock_data_tag**	Input	Tag of stock data object to query.
int *	**num_cross_sections**	Output	Count of cross sections returned.
tag_t * *	**cross_sections**	Output to UF_*free*	Array of cross section objects. Use UF_free to deallocate memory when no longer required.

UF_ROUTE_ask_stock_data_stock (view source)

Defined in: uf_route.h

Overview
Returns all the stock objects referencing the given stock data object.

Environment
Internal and External

History
Original release was in V15.0.

Required License(s)
gateway

```
int UF_ROUTE_ask_stock_data_stock
(
    tag_t stock_data_tag,
    int * num_stock,
    tag_t * * stock
)
```

tag_t	**stock_data_tag**	Input	Tag of stock data object to query.
int *	**num_stock**	Output	Count of stock objects returned.
tag_t * *	**stock**	Output to UF_*free*	Array of stock objects. Use UF_free to deallocate memory when no longer required.

UF_ROUTE_ask_stock_diameter (view source)

Defined in: uf_route.h

Overview
Returns the diameter of the Stock. Routing determines the diameter associated with Stock by reading the DIAMETER characteristic assigned to the Stock_Data object referenced by the Stock object. If no such characteristic exists, 0.0 is returned as the diameter.

Environment
Internal and External

History
Original release was in V14.0.

Required License(s)
gateway

```
int UF_ROUTE_ask_stock_diameter
(
    tag_t stock,
    double * diameter
)
```

tag_t	**stock**	Input	Object identifier of the Stock.
double *	**diameter**	Output	Diameter of the Stock.

UF_ROUTE_ask_stock_feature (view source)

Defined in: uf_route.h

Overview
Returns the feature which represents the stock object.

Environment
Internal and External

History
Original release was in V15.0.

Required License(s)
gateway

```
int UF_ROUTE_ask_stock_feature
(
    tag_t stock_tag,
    tag_t * feature
)
```

tag_t	**stock_tag**	Input	Tag of stock object
tag_t *	**feature**	Output	Tag of feature

UF_ROUTE_ask_stock_harness (view source)

Defined in: uf_route.h

Overview
Ask the harness associated with the stock tag.

Environment
Internal and External

History
Released in V17.0

Required License(s)
gateway

int UF_ROUTE_ask_stock_harness
(
    tag_t stock,
    int * num_harness,
    tag_t * * harness
)

tag_t	**stock**	Input	, the stock tag to query
int *	**num_harness**	Output	, the number of harnesses
tag_t * *	**harness**	Output to UF_*free*	, the array of harness tags, free array with UF_free.

UF_ROUTE_ask_stock_part_occ (view source)

Defined in: uf_route.h

Overview
This function is used to get the part occurrence tag associated with a piece of stock.

Environment
Internal and External

History
Released in V19.0

Required License(s)
gateway

int UF_ROUTE_ask_stock_part_occ
(
    tag_t stock,
    tag_t* stock_component
)

tag_t	**stock**	Input	Tag of the stock object.
tag_t*	**stock_component**	Output	Tag of the stock component

UF_ROUTE_ask_stock_ports (view source)

Defined in: uf_route.h

Overview
Returns the two end ports of the Stock.

Environment
Internal and External

History
Original release was in V14.0.

Required License(s)
gateway

```
int UF_ROUTE_ask_stock_ports
(
    tag_t stock,
    tag_t ports [ 2 ]
)
```

tag_t	**stock**	Input	Object identifier of the Stock.
tag_t	**ports [2]**	Output	Two end ports of the Stock.

UF_ROUTE_ask_stock_profile_port (view source)

Defined in: uf_route.h

Overview
Inquire which end of the stock the profile curves are placed to create the stock feature. The index is 0 or 1 as would be returned by UF_ROUTE_ask_stock_ports.

Environment
Internal and External

See Also
UF_ROUTE_ask_stock_ports

History
Original release was in V15.0.

Required License(s)
gateway

```
int UF_ROUTE_ask_stock_profile_port
(
    tag_t stock_tag,
    int * profile_port
)
```

tag_t	**stock_tag**	Input	Tag of stock to query
int *	**profile_port**	Output	Index of stock port at which the profile curves are placed: 0 or 1.

UF_ROUTE_ask_stock_rotation (view source)

Defined in: uf_route.h

Overview
Inquires the current rotation of the stock object.

Environment
Internal and External

History
Original release was in V15.0.

Required License(s)
gateway

```
int UF_ROUTE_ask_stock_rotation
(
    tag_t stock_tag,
    double * rotation
)
```

tag_t	**stock_tag**	Input	Tag of stock to query
double *	**rotation**	Output	Stock rotation (radians)

UF_ROUTE_ask_stock_segments (view source)

Defined in: uf_route.h

Overview
Returns the number and tags of the Routing segments to which the given stock object has been assigned.

Environment
Internal and External

History
Original release was in V14.0.

Required License(s)
gateway

```
int UF_ROUTE_ask_stock_segments
(
    tag_t stock,
    int * num_segments,
    tag_t * * segments
)
```

tag_t	**stock**	Input	Object identifier of the Stock.

int *	**num_segments**	Output	Number of segments for which this Stock is applied.
tag_t * *	**segments**	Output to UF_*free*	Array of segments to which this Stock applies. Use UF_free to free this allocated array.

UF_ROUTE_ask_stock_stock_data (view source)

Defined in: uf_route.h

Overview
Returns the Stock Data object of the Stock. This function can be used to retrieve all the characteristics related to a Stock object. Use UF_ROUTE_ask_characteristics on the Stock object followed by UF_ROUTE_ask_characteristics on the Stock Data object to read all the specific as well as the common characteristics of a Stock.

Environment
Internal and External

History
Original release was in V14.0.

Required License(s)
gateway

```
int UF_ROUTE_ask_stock_stock_data
(
    tag_t stock,
    tag_t * stock_data
)
```

tag_t	**stock**	Input	Object identifier of the Stock.
tag_t *	**stock_data**	Output	Stock Data object of the Stock.

UF_ROUTE_ask_stock_style (view source)

Defined in: uf_route.h

Overview
Returns the stock style assigned to the stock object.

Environment
Internal and External

History
Original release was in V15.0.

Required License(s)

gateway

```
int UF_ROUTE_ask_stock_style
(
    tag_t stock_tag,
    int * style
)
```

tag_t	**stock_tag**	Input	Tag of stock object to query.
int *	**style**	Output	Stock style: UF_ROUTE_STYLE_NONE UF_ROUTE_STYLE_SOLID UF_ROUTE_STYLE_DETAIL

UF_ROUTE_ask_stock_units (view source)

Defined in: uf_route.h

Overview

Returns the units associated with the supplied stock. Routing
allows stock of a particular unit, e.g., millimeters, to be assigned to
the segments of a part of different units, e.g., inches.
Routing stock created prior to V14.0.1 does not have
the units information and any such stock returns 0 as its units value.

Environment

Internal and External

History

Original release was in V15.0.

Required License(s)

gateway

```
int UF_ROUTE_ask_stock_units
(
    tag_t stock_tag,
    int * units
)
```

tag_t	**stock_tag**	Input	Tag of stock object to query
int *	**units**	Output	Units of the stock. This value will be either: UF_METRIC, UF_ENGLISH, or 0

UF_ROUTE_ask_stock_wires (view source)

Defined in: uf_route.h

Overview
Ask the tags of the wires associated with the given stock.

Environment
Internal and External

History
Released in V17.0

Required License(s)
gateway

```
int UF_ROUTE_ask_stock_wires
(
    tag_t stock,
    int * num_wires,
    tag_t * * wires
)
```

tag_t	**stock**	Input	, the stock to query
int *	**num_wires**	Output	, the number of wires
tag_t * *	**wires**	Output to UF_*free*	, the array of wires, free with UF_free

---

UF_ROUTE_ask_terminal_multiport (view source)

Defined in: uf_route.h

Overview
Queries the multiport of the given terminal port

Environment
Internal and External

See Also
UF_ROUTE_remove_virtual_ports

History
Originally released in V16.0

Required License(s)
gateway

```
int UF_ROUTE_ask_terminal_multiport
(
    tag_t terminal,
    tag_t * multi
)
```

tag_t	**terminal**	Input	Terminal port to query

tag_t *	**multi**	Output	Multiport

UF_ROUTE_ask_terminal_port_uid (view source)

Defined in: uf_route.h

Overview
Find the "uid" value for the characteristic
ROUTE_PORT_ID_CHARX_TITLE of given terminal port. If not
found on the terminal, look for it on its prototype if it
is an occurrence.

Environment
Internal and External

History
Released in V17.0

Required License(s)
gateway

```
int UF_ROUTE_ask_terminal_port_uid
(
    tag_t terminal,
    char * * uid
)
```

tag_t	**terminal**	Input	the terminal port being asked
char * *	**uid**	Output to UF_*free*	the uid of the terminal port. Use UF_free to free.

UF_ROUTE_ask_user_preferences (view source)

Defined in: uf_route.h

Overview
Fills in an array of preference structures. The key and type fields must
be set in each structure and the function returns the value for each
preference. If the preference is not found, the key value is set to
UF_ROUTE_USER_PREF_TYPE_ANY and the integer value set to 0.

Environment
Internal and External

See Also
UF_ROUTE_set_user_preferences

History
Original release was in V14.0.

Required License(s)

gateway

int UF_ROUTE_ask_user_preferences
(
    int n_pref,
    UF_ROUTE_user_preference_t * prefs
)

int	**n_pref**	Input	Number of preferences to query.
UF_ROUTE_user_preference_t *	**prefs**	Output to UF_*free*	Array of preference structures. The key and type fields must be set in each structure for the query. UF_ROUTE_free_user_prefs_data must be called to free the data allocated inside of the prefs array after this function call.

UF_ROUTE_ask_wire_harness (view source)

Defined in: uf_route.h

Overview
Ask the harnesses associated with a wire.

Environment
Internal and External

History
Released in V17.0

Required License(s)
gateway

int UF_ROUTE_ask_wire_harness
(
    tag_t wire,
    int * num_harness,
    tag_t * * harness
)

tag_t	**wire**	Input	, the wire to query
int *	**num_harness**	Output	, the number of harnesses
tag_t * *	**harness**	Output to UF_*free*	, the array of harnesses, free with UF_free.

UF_ROUTE_ask_wire_segments (view source)

Defined in: uf_route.h

Overview
Ask the segments in the given wire.

Environment
Internal and External

History
Released in V17.0

Required License(s)
gateway

```
int UF_ROUTE_ask_wire_segments
(
    tag_t wire,
    int * num_segments,
    tag_t * * segments
)
```

tag_t	**wire**	Input	, the wire to query
int *	**num_segments**	Output	, the number of segments
tag_t * *	**segments**	Output to UF_*free*	, the array of segments, free array with UF_free

---

UF_ROUTE_ask_wire_stock (view source)

Defined in: uf_route.h

Overview
Ask the stock tags associated with the wire.

Environment
Internal and External

History
Released in V17.0

Required License(s)
gateway

```
int UF_ROUTE_ask_wire_stock
(
    tag_t wire,
    int * num_stock,
    tag_t * * stock
)
```

tag_t	**wire**	Input	, the wire to query

int *	**num_stock**	Output	, the number of stock tags
tag_t * *	**stock**	Output to UF_*free*	, the array of stock tags, free with UF_free.

UF_ROUTE_assign_stock (view source)

Defined in: uf_route.h

Overview
Assigns stock to a set of segments. The segments need not belong to the same path. This function uses the Stock Data, Anchor, and Cross Section objects returned by "loading" the stock. Once the stock has been "loaded", using UF_ROUTE_load_stock_data, several calls to UF_ROUTE_assign_stock may be made (for various sets of segments) without the need to "load" another.

Environment
Internal and External

See Also
UF_ROUTE_load_stock_data
Please refer to the example

History
Original release was in V13.0.

Required License(s)
routing_base

```
int UF_ROUTE_assign_stock
(
    tag_t stock_data_tag,
    tag_t anchor_tag,
    tag_t cross_tag,
    int seg_count,
    tag_t * segments
)
```

tag_t	**stock_data_tag**	Input	Object identifier of the stock data obtained after loading the stock by UF_ROUTE_load_stock_data( ).
tag_t	**anchor_tag**	Input	Object identifier of the stock anchor obtained after loading the stock.
tag_t	**cross_tag**	Input	Object Identifier of the cross section tag obtained after loading stock.
int	**seg_count**	Input	Number of segments to assign this stock.
tag_t *	**segments**	Input	Array of segments tags to assign stock.

UF_ROUTE_assign_stock_style (view source)

Defined in: uf_route.h

Overview
Sets the stock style of the given stock objects.

Environment
Internal and External

History
Original release was in V15.0.

Required License(s)
(routing_base or routing_advanced)

```
int UF_ROUTE_assign_stock_style
(
    int new_style,
    int num_stocks,
    tag_p_t stock_tags
)
```

int	**new_style**	Input	Style to assign
int	**num_stocks**	Input	Count of stock objects being passed in
tag_p_t	**stock_tags**	Input	Array of stock objects

UF_ROUTE_bend_report_ask_number_of_bends (view source)

Defined in: uf_route.h

Overview
This function returns the number of bend segments in the given stock bend info.

Environment
Internal and External

Required License(s)
(routing_base or routing_advanced)

```
int UF_ROUTE_bend_report_ask_number_of_bends
(
    UF_ROUTE_bend_segment_info_p_t bend_segs,
    int* num_bends
)
```

UF_ROUTE_bend_segment_info_p_t	**bend_segs**	Input	Stock bend info
int*	**num_bends**	Output	Number of bends

UF_ROUTE_bend_report_free_mil98_report (view source)

Defined in: uf_route.h

Overview
This function frees a mil98 bend report structure that is returned from a call to UF_ROUTE_bend_report_generate_mil98_report.

Environment
Internal and External

Required License(s)
(routing_base or routing_advanced)

```
int UF_ROUTE_bend_report_free_mil98_report
(
    UF_ROUTE_bend_report_mil98_p_t mil98_data
)
```

UF_ROUTE_bend_report_mil98_p_t	**mil98_data**	Input	MIL98 bend report

UF_ROUTE_bend_report_free_segment_info (view source)

Defined in: uf_route.h

Overview
This function frees a bend segment info structure that is returned from a call to UF_ROUTE_bend_report_get_segment_info.

Environment
Internal and External

Required License(s)
(routing_base or routing_advanced)

```
int UF_ROUTE_bend_report_free_segment_info
(
    UF_ROUTE_bend_segment_info_p_t bend_segs
)
```

UF_ROUTE_bend_segment_info_p_t	**bend_segs**	Input	Stock bend info.

UF_ROUTE_bend_report_free_xyz_report (view source)

Defined in: uf_route.h

Overview
This function frees a xyz bend report structure that is returned from a call to UF_ROUTE_bend_report_generate_xyz_report.

Environment
Internal and External

Required License(s)
(routing_base or routing_advanced)

int UF_ROUTE_bend_report_free_xyz_report
(
UF_ROUTE_bend_report_xyz_p_t xyz_data
)

UF_ROUTE_bend_report_xyz_p_t	**xyz_data**	Input	XYZ bend report

---

UF_ROUTE_bend_report_free_ybc_report (view source)

Defined in: uf_route.h

Overview
This function frees a ybc bend report structure that is returned from a call to UF_ROUTE_bend_report_generate_ybc_report.

Environment
Internal and External

Required License(s)
(routing_base or routing_advanced)

int UF_ROUTE_bend_report_free_ybc_report
(
UF_ROUTE_bend_report_ybc_p_t ybc_data
)

UF_ROUTE_bend_report_ybc_p_t	**ybc_data**	Input	YBC bend report

---

UF_ROUTE_bend_report_generate_mil98_report (view source)

Defined in: uf_route.h

Overview
This function takes the data returned from a call to UF_ROUTE_bend_report_get_segment_info, and returns a data structure containing the bend report information in the MIL98 format.
Call UF_ROUTE_bend_report_free_mil98_report to free up the returned data.

Environment
Internal and External

Required License(s)
(routing_base or routing_advanced)

```
int UF_ROUTE_bend_report_generate_mil98_report
(
    UF_ROUTE_bend_segment_info_p_t bend_segs,
    UF_ROUTE_bend_report_mil98_p_t * mil98_data
)
```

UF_ROUTE_bend_segment_info_p_t	**bend_segs**	Input	Stock bend info
UF_ROUTE_bend_report_mil98_p_t *	**mil98_data**	Output to UF_*free*	MIL98 bend report

---

UF_ROUTE_bend_report_generate_xyz_report (view source)

Defined in: uf_route.h

Overview
This function takes the data returned from a call to UF_ROUTE_bend_report_get_segment_info, and returns a data structure containing the bend report information in the XYZ format.
Call UF_ROUTE_bend_report_free_xyz_report to free up the returned data.

Environment
Internal and External

Required License(s)
(routing_base or routing_advanced)

```
int UF_ROUTE_bend_report_generate_xyz_report
(
    UF_ROUTE_bend_segment_info_p_t bend_segs,
    UF_ROUTE_bend_report_xyz_p_t * xyz_data
)
```

UF_ROUTE_bend_segment_info_p_t	**bend_segs**	Input	Stock bend info.
UF_ROUTE_bend_report_xyz_p_t *	**xyz_data**	Output to UF_*free*	XYZ bend report

---

UF_ROUTE_bend_report_generate_ybc_report (view source)

Defined in: uf_route.h

Overview
This function takes the data returned from a call to UF_ROUTE_bend_report_get_segment_info, and returns a data structure containing the bend report information in the YBC format.

Call UF_ROUTE_bend_report_free_ybc_report to free up the returned data.

Environment
Internal and External

Required License(s)
(routing_base or routing_advanced)

```
int UF_ROUTE_bend_report_generate_ybc_report
(
    UF_ROUTE_bend_segment_info_p_t bend_segs,
    UF_ROUTE_bend_report_ybc_p_t * ybc_data
)
```

UF_ROUTE_bend_segment_info_p_t	**bend_segs**	Input	Stock bend info.
UF_ROUTE_bend_report_ybc_p_t *	**ybc_data**	Output to UF_*free*	YBC bend report

---

UF_ROUTE_bend_report_get_segment_info (view source)

Defined in: uf_route.h

Overview
This function initializes a bend segment info structure with the
bend data from a given solid body, segment or stock tag.
Call UF_ROUTE_bend_report_free_segment_info to free up the bend_segs structure.

Environment
Internal and External

Required License(s)
(routing_base or routing_advanced)

```
int UF_ROUTE_bend_report_get_segment_info
(
    tag_t pipe_tag,
    UF_ROUTE_bend_segment_info_p_t * bend_segs
)
```

tag_t	**pipe_tag**	Input	Segment, solid body, or stock tag
UF_ROUTE_bend_segment_info_p_t *	**bend_segs**	Output to UF_*free*	Stock bend info.

UF_ROUTE_bend_report_reverse_direction (view source)

Defined in: uf_route.h

Overview

This function reverses the order of bends in a stock bend info structure. Call UF_ROUTE_bend_report_free_segment_info to free up the reversed_bend_segs structure.

Environment

Internal and External

Required License(s)

(routing_base or routing_advanced)

```
int UF_ROUTE_bend_report_reverse_direction
(
    UF_ROUTE_bend_segment_info_p_t bend_segs,
    UF_ROUTE_bend_segment_info_p_t * reversed_bend_segs
)
```

UF_ROUTE_bend_segment_info_p_t	**bend_segs**	Input	Stock bend info
UF_ROUTE_bend_segment_info_p_t *	**reversed_bend_segs**	Output to UF_*free*	Stock bend info in reverse order

---

UF_ROUTE_calc_abs_minmax_box (view source)

Defined in: uf_route.h

Overview

Return the absolute min-max box that contains all the entities in the view.

Environment

Internal and External

Required License(s)

(routing_base or routing_advanced)

```
int UF_ROUTE_calc_abs_minmax_box
(
    tag_t dwg_view,
    double box [ 6 ]
)
```

tag_t	**dwg_view**	Input	View tag
double	**box [6]**	Output	Absolute coordinates of the min_x, max_x, min_y, max_y, min_z, max_z

---

UF_ROUTE_compute_stock_length (view source)

Defined in: uf_route.h

Overview
Computes the length of the stock object.

Environment
Internal and External

History
Original release was in V15.0.

Required License(s)
routing_base

```
int UF_ROUTE_compute_stock_length
(
    tag_t stock,
    double * total_path_length
)
```

tag_t	**stock**	Input	Tag of stock object
double *	**total_path_length**	Output	Length of stock

UF_ROUTE_connect_port (view source)

Defined in: uf_route.h

Overview
Attempts to connect a port to other available ports.

Environment
Internal and External

History
Original release was in V15.0.

Required License(s)
routing_base

```
int UF_ROUTE_connect_port
(
    tag_t port,
    tag_t * connection
)
```

tag_t	**port**	Input	Tag of port
tag_t *	**connection**	Output	Tag of connection (or NULL_TAG)

UF_ROUTE_convert_to_stock_as_components (view source)

Defined in: uf_route.h

Overview

This function is used to convert the work part from a old-style stock part (stock in the work part) to a new-style stock as components part (stock created in components).

The part specified is the part to convert, if a NULL_TAG is passed in then the work part is converted. The convert_subcomponents argument causes the conversion function to be called recursively on all subcomponents of the part being converted. The permanent_stock argument causes the all stock components that were created by the conversion to have the STOCK_COMPONENT_NAME user exit applied to each one, and the name of the stock component changed. The reuse_stock argument causes all stock components to have the the UF_ROUTE_reuse_stock_part call applied to each of them.

Environment

Internal and External

History

Released in V19.0

Required License(s)

(routing_base or routing_advanced)

```
int UF_ROUTE_convert_to_stock_as_components
(
    tag_t part,
    logical convert_subcomponents,
    logical permanent_stock,
    logical reuse_stock
)
```

tag_t	**part**	Input	Part to convert
logical	**convert_subcomponents**	Input	TRUE - convert all sub-assemblies of part, FALSE - convert only the given part
logical	**permanent_stock**	Input	TRUE - call STOCK_COMPONENT_NAME plugin on each stock FALSE - only call STOCK_COMPONENT_TEMP_NAME plugin
logical	**reuse_stock**	Input	TRUE - call STOCK_COMPONENT_LOOKUP plugin on each stock, FALSE - dont attempt to reuse stocks

---

UF_ROUTE_create_anchor_from_pnt (view source)

Defined in: uf_route.h

Overview

Creates a new anchor associative to the position of an existing point.

Environment
Internal and External

Required License(s)
(routing_base or routing_advanced)

```
int UF_ROUTE_create_anchor_from_pnt
(
    tag_t object_in_part,
    tag_t ref_point,
    tag_t * anchor
)
```

tag_t	**object_in_part**	Input	Object in part to create new anchor NULL_TAG = work part
tag_t	**ref_point**	Input	Tag of point object
tag_t *	**anchor**	Output	Tag of new anchor

UF_ROUTE_create_anchor_from_pos (view source)

Defined in: uf_route.h

Overview
Creates an anchor given a set of coordinates.

Environment
Internal and External

Required License(s)
(routing_base or routing_advanced)

```
int UF_ROUTE_create_anchor_from_pos
(
    tag_t object_in_part,
    double point_pos [ 3 ] ,
    tag_t * anchor
)
```

tag_t	**object_in_part**	Input	Tag of object in part to create new object NULL_TAG = work part
double	**point_pos [3]**	Input	Absolute position to create anchor
tag_t *	**anchor**	Output	Tag of created anchor

UF_ROUTE_create_bend_by_radius (view source)

Defined in: uf_route.h

Overview

Assigns a Bend corner with the given radius to the input object. If the input object is the RCP, corner, or Segment associated with an existing bend corner, the corner and associated Segment is updated with the new radius. If the assignment is to an RCP or Corner of a Miter corner, the old corner is removed and a new Bend corner is created.

Environment

Internal and External

See Also

Please refer to the example

History

Original release was in V13.0.

Required License(s)

routing_base

```
int UF_ROUTE_create_bend_by_radius
(
    tag_t obj_id,
    double radius,
    tag_t * corner,
    tag_t * seg
)
```

tag_t	**obj_id**	Input	Object identifier of an existing RCP, or a Bend segment or a Corner.
double	**radius**	Input	Radius of the Bend Corner.
tag_t *	**corner**	Output	Object identifier of the newly created Corner.
tag_t *	**seg**	Output	Object identifier of the Bend corner segment.

UF_ROUTE_create_bend_by_ratio (view source)

Defined in: uf_route.h

Overview

Assigns a Bend corner. The radius for the bend is determined by multiplying the given ratio times the diameter of the stock. The stock's diameter is determined from a characteristic assigned to the Stock Data object associated with the stock. The characteristic title should be that returned by routine UF_ROUTE_ask_app_view_diameter.

If the input object is the RCP, corner, or Segment associated with an existing bend corner, the corner and associated Segment is updated with the new radius. If the assignment is to an RCP or Corner of a Miter corner, the old corner is removed and a new Bend corner is created.

Environment
Internal and External

See Also
Please refer to the example

History
Original release was in V13.0.

Required License(s)
routing_base

```
int UF_ROUTE_create_bend_by_ratio
(
    tag_t obj_id,
    double ratio,
    tag_t * corner,
    tag_t * seg
)
```

tag_t	**obj_id**	Input	Object identifier of an existing RCP, or a Bend segment or a Corner.
double	**ratio**	Input	Ratio of stock diameter to bend radius assigned to the Bend corner.
tag_t *	**corner**	Output	Object identifier of the newly created Corner.
tag_t *	**seg**	Output	Object identifier of the Bend corner segment.

UF_ROUTE_create_bend_by_table (view source)

Defined in: uf_route.h

Overview
This function will create a bend corner at the given junction. The parameters of the corner are gotten from the given table name and the (largest) diameter stock at the junction. If a corner already exists, it will be removed/ modified to the new type.

Environment
Internal and External

History
Released in V18.0

Required License(s)
routing_base

```
int UF_ROUTE_create_bend_by_table
(
    tag_t obj_id,
    char * table,
    tag_t * corner,
    tag_t * seg
)
```

tag_t	**obj_id**	Input	Object identifier of an existing RCP, or a Bend segment or a Corner.
char *	**table**	Input	Table name to pull radius from.
tag_t *	**corner**	Output	Object identifier of the newly created Corner.
tag_t *	**seg**	Output	Object identifier of the Bend corner segment.

UF_ROUTE_create_built_in_path (view source)

Defined in: uf_route.h

Overview
Create a new built-in path in a routing part.

Environment
Internal and External

History
New in V17

Required License(s)
routing_base

```
    int UF_ROUTE_create_built_in_path
    (
        tag_t part,
        int num_objs,
        tag_t * objs,
        char * name,
        tag_t * bip_tag
    )
```

tag_t	**part**	Input	Tag of part containing curves. When a NULL_TAG is passed, built-in path will be created in current work part
int	**num_objs**	Input	Number of curves in path
tag_t *	**objs**	Input	Array of curve tags
char *	**name**	Input	Built-in path name. This can be NULL.
tag_t *	**bip_tag**	Output	Tag of built-in path created

UF_ROUTE_create_cross_section (view source)

Defined in: uf_route.h

Overview

Creates a Routing Cross Section object

A Cross Section object defines the set of "profile" curves that are swept along a Routing path to represent the "stock", i.e., pipe, wire, tube, etc. These curves should be defined to be in the XY plane, centered about the origin. A copy of the curves is transformed to the start of the set of segments for a stock and then swept along the segments to model the stock.

Environment

Internal and External

History

Original release was in V18.0.

Required License(s)

routing_base

```
int UF_ROUTE_create_cross_section
(
    tag_t object_in_part,
    int style,
    tag_t exprs [ ] ,
    int num_curves,
    tag_t curves [ ] ,
    tag_t * cross
)
```

tag_t	**object_in_part**	Input	Tag of an existing object which defines which part the cross section will be created in
int	**style**	Input	Routing stock style - one of UF_ROUTE_STYLE_SIMPLE UF_ROUTE_STYLE_DETAIL
tag_t	**exprs [ ]**	Input	Tags of 2 expressions which define the offsets to be applied to the curves when sweeping the curves along the path. Positive offset is "away" from the origin. Negative offset is "toward" the origin.
int	**num_curves**	Input	The number of curves in the "profile" or "cross section"
tag_t	**curves [ ]**	Input	Tags of the curves
tag_t *	**cross**	Output	Tag of the created cross section object

UF_ROUTE_create_iso_drawing (view source)

Defined in: uf_route.h

Overview
Create an Isometric Drawing for the selected part

Environment
Internal and External

History
Originally released in V16.0

Required License(s)
routing_base

```
int UF_ROUTE_create_iso_drawing
(
    tag_t part_tag
)
```

tag_t	**part_tag**	Input	Tag of the part to be represented in the isometric drawing.

UF_ROUTE_create_miter_corner (view source)

Defined in: uf_route.h

Overview
Assigns a Miter corner to the input object. The assignment can involve the removal of an existing Corner and the creation of a new Corner.

Environment
Internal and External

History
Original release was in V13.0.

Required License(s)
routing_base

```
int UF_ROUTE_create_miter_corner
(
    tag_t obj_id,
    tag_t * corner
)
```

tag_t	**obj_id**	Input	Object identifier of an existing RCP, or Bend segment or a Miter Corner.
tag_t *	**corner**	Output	Object identifier of the newly created Corner.

UF_ROUTE_create_multiport_from_position (view source)

Defined in: uf_route.h

Overview

Function Name: UF_ROUTE_mig_create_multi_port

Function Description:
Create a multiport in the specified part.

Environment

Internal

History

Released in V20

Required License(s)

routing_base

```
int UF_ROUTE_create_multiport_from_position
(
    tag_t part,
    double position [ 3 ] ,
    logical align_flag,
    double align_vector [ 3 ] ,
    logical fixture_port,
    char * term_id,
    tag_t * port_tag
)
```

tag_t	**part**	Input	Tag of part to create port. When a NULL_TAG is passed, port will be created in current work part
double	**position [3]**	Input	Position of port in absolute csys
logical	**align_flag**	Input	Port alignment flag: TRUE = Port uses an alignment vector. FALSE = Port does not use an alignment vector.
double	**align_vector [3]**	Input	Alignment vector of port
logical	**fixture_port**	Input	Is the port a fixture port? TRUE = Port is a fixture port. FALSE = Port is not a fixture port.
char *	**term_id**	Input	Terminal Id
tag_t *	**port_tag**	Output	Tag of created port

UF_ROUTE_create_port_at_segend (view source)

Defined in: uf_route.h

Overview

Creates a port at the specified end of a segment. The position of the port is associated with the end RCP. The alignment direction of the port is associated with the slope of the segment at the end.

Environment
Internal and External

History
Original release was in V15.0.

Required License(s)
routing_base

```
int UF_ROUTE_create_port_at_segend
(
    tag_t segment,
    int segend,
    logical rotate_flag,
    tag_t * port
)
```

tag_t	**segment**	Input	Tag of segment
int	**segend**	Input	End index (0 or 1)
logical	**rotate_flag**	Input	Does the port have a rotation vector?
tag_t *	**port**	Output	Tag of created port

UF_ROUTE_create_port_lock (view source)

Defined in: uf_route.h

Overview
This function locks two components together using mating conditions. The given port occurrence is a port that is connected to a port that is part of another component. The component of the passed in port is the from (or child) of the mating condition. The lock rotation flag indicates whether or not the child component will be able to rotate.

If an error occurs, UF_ROUTE_err_invalid_port_mate will be returned.

Environment
Internal and External

History
Released in V18.0

Required License(s)
routing_base

```
int UF_ROUTE_create_port_lock
(
        tag_t from_port_occ,
        logical lock_rotation
    )
```

tag_t	**from_port_occ**	Input	From (child) port occurrence tag.
logical	**lock_rotation**	Input	Lock rotation.

UF_ROUTE_create_rcp_arc_center (view source)

Defined in: uf_route.h

Overview

Creates a new RCP at the center of the given arc or circular edge. This RCP is derived from the given curve and moves if the curve is moved. If a previously created RCP exists at this location, it is returned.

Environment

Internal and External

Required License(s)

(routing_base or routing_advanced)

```
    int UF_ROUTE_create_rcp_arc_center
    (
        tag_t arc,
        tag_t * new_rcp
    )
```

tag_t	**arc**	Input	Object identifier of an existing arc or circular edge.
tag_t *	**new_rcp**	Output	Object identifier of the newly created RCP. In case an RCP exists at this location then the object identifier of this RCP is returned.

UF_ROUTE_create_rcp_at_port (view source)

Defined in: uf_route.h

Overview

Creates an RCP whose position is associatively defined by an existing port.

Environment

Internal and External

Required License(s)

(routing_base or routing_advanced)

```
int UF_ROUTE_create_rcp_at_port
(
    tag_t port,
    tag_t * new_rcp,
    logical check_existing_rcp,
    logical * found_existing_rcp
)
```

tag_t	**port**	Input	Tag of reference port
tag_t *	**new_rcp**	Output	tag of created/found RCP
logical	**check_existing_rcp**	Input	TRUE = use existing RCP at specified position FALSE = always create a new RCP

logical *	**found_existing_rcp**	Output	TRUE = existing RCP found and used
			FALSE = new RCP was created

UF_ROUTE_create_rcp_by_wcs_off (view source)

Defined in: uf_route.h

Overview
Creates (or finds) and RCP at the position derived by specifying an existing RCP or port plus a wcs offset from that port.

Environment
Internal and External

Required License(s)
(routing_base or routing_advanced)

```
int UF_ROUTE_create_rcp_by_wcs_off
(
    tag_t object,
    double offset [ 3 ] ,
    tag_t * new_rcp,
    logical check_existing_rcp,
    logical * found_existing_rcp
)
```

tag_t	**object**	Input	Reference RCP or port(in the work part)
double	**offset [3]**	Input	Offset in absolute WCS
tag_t *	**new_rcp**	Output	Tag of new or found RCP
logical	**check_existing_rcp**	Input	TRUE = use existing RCP at specified position FALSE = always create a new RCP

logical *	**found_existing_rcp**	Output	TRUE = existing RCP found and used
			FALSE = new RCP was created

UF_ROUTE_create_rcp_by_work_pos (view source)

Defined in: uf_route.h

Overview
Creates (or finds) an RCP at a work position in the WCS.

Environment
Internal and External

See Also
UF_ROUTE_create_rcp_position

Required License(s)
(routing_base or routing_advanced)

```
int UF_ROUTE_create_rcp_by_work_pos
(
    double work_pos [ 3 ] ,
    tag_t * new_rcp,
    logical check_existing_rcp,
    logical * found_existing_rcp
)
```

double	**work_pos [3]**	Input	Position in absolute work part coords
tag_t *	**new_rcp**	Output	Tag of created RCP
logical	**check_existing_rcp**	Input	TRUE = use existing RCP at specified position FALSE = always create a new RCP

logical *	**found_existing_rcp**	Output	TRUE = existing RCP found and used
			FALSE = new RCP was created

UF_ROUTE_create_rcp_curve_parm (view source)

Defined in: uf_route.h

Overview
Creates a new RCP at a point corresponding to the given parameter value along the given curve. If a previously created RCP exists at this location, it is returned.

This routine should be used to create the RCPs at the ends of any Segment which you create using UF_ROUTE_create_seg_on_curve. This is to insure that if the curve is transformed or moved, both the Segment and its RCPs also move with the curve.

Environment
Internal and External

See Also
UF_ROUTE_create_seg_on_curve
Refer to the example

Required License(s)

(routing_base or routing_advanced)

```
int UF_ROUTE_create_rcp_curve_parm
(
    tag_t curve,
    double parm,
    tag_t * new_rcp
)
```

tag_t	**curve**	Input	Object identifier of an existing curve.
double	**parm**	Input	Normalized curve parameter at which to create the RCP. The value should be between 0 and 1, inclusive.
tag_t *	**new_rcp**	Output	Object identifier of the newly created RCP. The RCP location is derived from the given curve and parameter and the RCP's position updates if the curve is moved. In case an RCP exists at this location then the object identifier of this RCP is returned.

UF_ROUTE_create_rcp_on_rcp (view source)

Defined in: uf_route.h

Overview

Creates an RCP in the work part whose position is associative to an occurrence of an RCP.

Environment

Internal and External

Required License(s)

(routing_base or routing_advanced)

```
int UF_ROUTE_create_rcp_on_rcp
(
    tag_t occ_rcp,
    tag_t * new_rcp,
    logical check_existing_rcp,
    logical * found_existing_rcp
)
```

tag_t	**occ_rcp**	Input	Tag of reference rcp occurrence
tag_t *	**new_rcp**	Output	Tag of created/found RCP
logical	**check_existing_rcp**	Input	TRUE = use existing RCP at specified position FALSE = always create a new RCP

logical *	**found_existing_rcp**	Output	TRUE = existing RCP found and used
			FALSE = new RCP was created

UF_ROUTE_create_rcp_point (view source)

Defined in: uf_route.h

Overview
Creates a new RCP attached to the existing point if no previous RCP exists at this location. Else it returns the previously created RCP.

Environment
Internal and External

See Also
Refer to the example

Required License(s)
(routing_base or routing_advanced)

```
int UF_ROUTE_create_rcp_point
(
    tag_t point,
    tag_t * new_rcp
)
```

tag_t	**point**	Input	Object identifier of an existing point.
tag_t *	**new_rcp**	Output	Object identifier of the newly created RCP at this point. The RCP location is derived from this point at each update. In case an RCP exists at this location then the object identifier of this RCP is returned.

UF_ROUTE_create_rcp_position (view source)

Defined in: uf_route.h

Overview
Creates a new RCP at the given location if no previous RCP exists. If an RCP already exists it returns the previously created RCP.

Environment
Internal and External

See Also
UF_ROUTE_create_rcp_by_work_pos
Refer to the example

Required License(s)
(routing_base or routing_advanced)

```
int UF_ROUTE_create_rcp_position
(
    double pos [ 3 ] ,
    tag_t * new_rcp
)
```

double	**pos [3]**	Input	Location of RCP in absolute display coords.
tag_t *	**new_rcp**	Output	Object identifier of the newly created RCP at this location. In case an RCP exists at this location then the object identifier of this RCP is returned

---

UF_ROUTE_create_round_cross_section (view source)

Defined in: uf_route.h

Overview

Creates a Round Routing Cross Section object

This is a simplified version of the UF_ROUTE_create_cross_section routine which will create a cross section consisting of a single curve (a circle of the specified diameter).

Environment

Internal and External

History

Original release was in V18.0.

Required License(s)

routing_base

```
int UF_ROUTE_create_round_cross_section
(
    tag_t object_in_part,
    int style,
    double diameter,
    char * offsets [ 2 ] ,
    tag_t * cross
)
```

tag_t	**object_in_part**	Input	Tag of an existing object which defines which part the cross section will be created in
int	**style**	Input	Routing stock style - one of UF_ROUTE_STYLE_SIMPLE UF_ROUTE_STYLE_DETAIL
double	**diameter**	Input	The diameter of the circle created
char *	**offsets [2]**	Input	Array of 2 strings which define the offsets to be applied to the curves when sweeping the curves along the path. Positive offset is "away"

			from the origin. Negative offset is "toward" the origin. Example, ".25" or "-.35". These are used to create expressions and as such may be any string that is valid for the right hand side of an expression, e.g., "radius (3 / 16)".
tag_t *	**cross**	Output	Tag of the created cross section object

---

UF_ROUTE_create_seg_on_curve (view source)

Defined in: uf_route.h

Overview

Creates a segment between two RCPs on the curve and follows its profile. The RCPs supplied to this routine should be created with the UF_ROUTE_create_rcp_curve_parm.

Environment

Internal and External

See Also

UF_ROUTE_create_rcp_curve_parm
Please refer to the example

Required License(s)

(routing_base or routing_advanced)

```
int UF_ROUTE_create_seg_on_curve
(
    tag_t curve,
    tag_t rcp1,
    tag_t rcp2,
    tag_t * new_segment
)
```

tag_t	**curve**	Input	Object Identifier of the curve to follow while creating the segment.
tag_t	**rcp1**	Input	Object Identifier of the start RCP on the curve.
tag_t	**rcp2**	Input	Object Identifier of the end RCP on the curve.
tag_t *	**new_segment**	Output	Object identifier of newly created segment.

UF_ROUTE_create_seg_thru_rcps (view source)

Defined in: uf_route.h

Overview

Creates a segment between two existing RCPs and creates DCM3 segment if dcm3 is active

Environment
Internal and External

See Also
Please refer to the example

Required License(s)
(routing_base or routing_advanced)

```
int UF_ROUTE_create_seg_thru_rcps
(
    tag_t rcp1,
    tag_t rcp2,
    tag_t * new_segment
)
```

tag_t	**rcp1**	Input	Object Identifier of the start RCP.
tag_t	**rcp2**	Input	Object Identifier of the end RCP.
tag_t *	**new_segment**	Output	Object identifier of newly created segment.

UF_ROUTE_delete_characteristics (view source)

Defined in: uf_route.h

Overview
Deletes a list of characteristics from a given routing object.

Environment
Internal and External

History
Original release was in V13.0.

Required License(s)
routing_base

```
int UF_ROUTE_delete_characteristics
(
    tag_t obj_id,
    int charx_count,
    UF_ROUTE_charx_p_t list
)
```

tag_t	**obj_id**	Input	Object identifier of the routing object whose characteristics needs to be deleted.
int	**charx_count**	Input	Number of characteristics to be deleted.
UF_ROUTE_charx_p_t	**list**	Input	List of characteristics to be deleted.

UF_ROUTE_delete_port_lock (view source)

Defined in: uf_route.h

Overview
This function deletes a lock between two components. The passed in object is the from (or child) port or part occurrence.

If an error occurs, UF_ROUTE_err_invalid_port_mate will be returned.

Environment
Internal and External

History
Released in V18.0

Required License(s)
routing_base

```
int UF_ROUTE_delete_port_lock
(
    tag_t from_occ
)
```

tag_t	**from_occ**	Input	child port occ, or part occ

UF_ROUTE_disconnect_port (view source)

Defined in: uf_route.h

Overview
Disconnects the port from any connection which it may include.

Environment
Internal and External

History
Original release was in V15.0.

Required License(s)
routing_base

```
int UF_ROUTE_disconnect_port
(
    tag_t port
)
```

tag_t	**port**	Input	Port to disconnect

UF_ROUTE_enter_custom_app (view source)

Defined in: uf_route_ugopenint.h

Overview
Enters a custom Routing application.

Environment
Internal

See Also
Please refer to the example

History
Original release was in V14.0.

Required License(s)
routing_base

```
int UF_ROUTE_enter_custom_app
(
    void
)
```

UF_ROUTE_exit_custom_app (view source)

Defined in: uf_route_ugopenint.h

Overview
Exits a custom Routing application.

Environment
Internal

See Also
Please refer to the example

History
Original release was in V14.0.

Required License(s)
routing_base

```
int UF_ROUTE_exit_custom_app
(
    void
)
```

UF_ROUTE_find_part_in_path (view source)

Defined in: uf_route.h

Overview

Finds the part described by the base name using the path appropriate
to the current path. When using interactive Routing, the path is
set based on the application view.

Environment

Internal and External

See Also

UF_ROUTE_set_part_search_path
UF_ROUTE_ask_part_search_path

History

Original release was in V15.0.

Required License(s)

routing_base

```
int UF_ROUTE_find_part_in_path
(
    char * part_name,
    char * * path
)
```

char *	**part_name**	Input	Base name of part to find
char * *	**path**	Output to UF_*free*	Fully qualified name of part file. This must be freed by calling UF_free.

UF_ROUTE_find_path (view source)

Defined in: uf_route.h

Overview

Finds a path between two segments and returns an array of tags consisting
of the segments and part occurrences that make up the path. The array of
tags returned is in order from begin to end. In order to find a correct path,
all the segments and any part occurences in between begin and end must be at
the work part level. This method will not traverse sub assemblies.
This method is used to create a path between start and end connectors.
The start and end connectors must be placed by selecting an RCP for 'Place Part'.
And that RCP must still exist for UF_ROUTE_find_path to find a path.
If the connectors are placed on a point or the RCP is removed with 'Simplify Path',
UF_ROUTE_find_path will not find a path.

Environment

Internal and External

Required License(s)

(routing_base or routing_advanced)

```
int UF_ROUTE_find_path
(
    tag_t begin,
    tag_t end,
    int * path_size,
    tag_t * * path_data
)
```

tag_t	**begin**	Input	The tag of the segment at the beginning of the path
tag_t	**end**	Input	The tag of the segment at the end of the path
int *	**path_size**	Output	The size of the path; the number of tags in path_data
tag_t * *	**path_data**	Output to UF_*free*	The array of tags that represent the path. Use UF_free to deallocate memory when no longer needed

---

UF_ROUTE_find_port_charx (view source)

Defined in: uf_route.h

Overview
Finds a Routing characteristic of the given title for the supplied port occurrence. If the given port occurrence does not have the given characteristic, the port's part occurrence is queried for the characteristic.

Environment
Internal and External

History
Original release was in V14.0.

Required License(s)
routing_base

```
int UF_ROUTE_find_port_charx
(
    char * charx_name,
    int type,
    tag_t port,
    UF_ROUTE_charx_p_t charx
)
```

char *	**charx_name**	Input	Name of the characteristic to be found
int	**type**	Input	Type of the characteristic to be found, e.g., UF_ROUTE_CHARX_TYPE_INT
tag_t	**port**	Input	Tag of the port occurrence to be queried
UF_ROUTE_charx_p_t	**charx**	Output	Address of charx structure (allocated by the caller) which will be filled in with the value

of the characteristic.

UF_ROUTE_find_terminal_charx (view source)

Defined in: uf_route.h

Overview

Find the characteristic of specified type and name
for the given terminal port. The characteristic is
looked for in the following order:
1) input terminal tag
2) the terminal's prototype if the input terminal is an occurrence.
3) the terminal's multiport
4) the terminal's part occurrence

Environment

Internal and External

History

Released in V17.0

Required License(s)

routing_base

```
int UF_ROUTE_find_terminal_charx
(
    char * charx_name,
    int charx_type,
    tag_t terminal,
    UF_ROUTE_charx_p_t charx
)
```

char *	**charx_name**	Input	, the name of the charx to search for
int	**charx_type**	Input	, the type of the charx
tag_t	**terminal**	Input	, the tag of the terminal port being asked
UF_ROUTE_charx_p_t	**charx**	Output	, the desired charx

UF_ROUTE_find_terminal_port (view source)

Defined in: uf_route.h

Overview

Finds the terminal port on the given multiport

Return

TRUE if a terminal or virtual port is found, else FALSE

Environment

Internal and External

History

Originally released in V16.0

Required License(s)

gateway

```
logical UF_ROUTE_find_terminal_port
(
    tag_t multi,
    char * id,
    tag_t * tag
)
```

tag_t	**multi**	Input	Multiport to search
char *	**id**	Input	ID of the terminal port
tag_t *	**tag**	Output	Pointer to tag of a terminal port, if found.

UF_ROUTE_find_title_in_charx (view source)

Defined in: uf_route.h

Overview

Finds the index of the charx whose title matches the title passed in. If the index is -1 the title was not found in the charx.

Environment

Internal and External

History

New in V17

Required License(s)

routing_base

```
int UF_ROUTE_find_title_in_charx
(
    int num_charx,
    UF_ROUTE_charx_p_t charx,
    char* title,
    int* index
)
```

int	**num_charx**	Input	number of charx
UF_ROUTE_charx_p_t	**charx**	Input	array of charx to search
char*	**title**	Input	title to search for
int*	**index**	Output	Index into charx that matches title

UF_ROUTE_free_array_of_tag_lists (view source)

Defined in: uf_route.h

Overview
This routine will free the memory associated with a variable length array of pointers to UF_ROUTE_tag_list_t structures. Returns an error code if any occurs, during the operation.

Environment
Internal and External

History
New in V17

Required License(s)
(routing_base or routing_advanced)

```
int UF_ROUTE_free_array_of_tag_lists
(
    int num_tag_lists,
    UF_ROUTE_tag_list_p_t * array_of_tag_lists
)
```

int	**num_tag_lists**	Input	Number of tag_lists in the array to be freed
UF_ROUTE_tag_list_p_t *	**array_of_tag_lists**	Input	Array of tag_lists to be freed

UF_ROUTE_free_charx_array (view source)

Defined in: uf_route.h

Overview
This function frees the allocated charx array. Must use after every call to UF_ROUTE_ask_characteristics

Environment
Internal and External

Required License(s)
routing_base

```
int UF_ROUTE_free_charx_array
(
    int num_charx,
    UF_ROUTE_charx_p_t charx_list
)
```

int	**num_charx**	Input	Number of Charx values
UF_ROUTE_charx_p_t	**charx_list**	Input	Array of the characteristics

---

UF_ROUTE_free_match_results (view source)

Defined in: uf_route.h

Overview
Frees the memory associated with the result of a part library match. These values should not have been changed since being returned from the match function (UF_ROUTE_match_charx_in_plib).

Environment
Internal and External

See Also
UF_ROUTE_match_charx_in_plib

History
Original release was in V14.0.

Required License(s)
routing_base

```
int UF_ROUTE_free_match_results
(
    int num_matches,
    UF_ROUTE_part_lib_part_p_t matches
)
```

int	**num_matches**	Input	Number of matches to be freed
UF_ROUTE_part_lib_part_p_t	**matches**	Input	Array of matches to be freed

---

UF_ROUTE_free_places (view source)

Defined in: uf_route.h

Overview
This routine should be used to free the data returned by the UF_ROUTE_solve_places routine.

Environment
Internal and External

Required License(s)
routing_base

```
int UF_ROUTE_free_places
(
    int num_places,
    UF_ROUTE_place_solution_p_t * places
)
```

int	**num_places**	Input	number of "places" entries
UF_ROUTE_place_solution_p_t *	**places**	Input	array of placement objects

---

UF_ROUTE_free_user_prefs_data (view source)

Defined in: uf_route.h

Overview
Frees all data associated with a user preferences query.

Environment
Internal and External

History
Original release was in V15.0.

Required License(s)
routing_base

```
int UF_ROUTE_free_user_prefs_data
(
    int n_prefs,
    UF_ROUTE_user_preference_p_t prefs
)
```

int	**n_prefs**	Input	Number of preferences in data
UF_ROUTE_user_preference_p_t	**prefs**	Input	User preference data to free.

UF_ROUTE_get_next_connections (view source)

Defined in: uf_route.h

Overview
UF_ROUTE_get_next_connections

DESCRIPTION:
Determine next set of connected objects to traverse from a given rcp or port occurrence (curr_conn), excluding objects based on current part occurrence or segment (curr_obj).

Environment
Internal and External

History
Released in NX

Required License(s)
(routing_base or routing_advanced)

```
int UF_ROUTE_get_next_connections
(
    tag_t curr_conn,
    tag_t curr_obj,
    int * num_conns,
    tag_t * * next_conns,
    tag_t * * next_objs
)
```

tag_t	**curr_conn**	Input	object to traverse (rcp or port occurrence)
tag_t	**curr_obj**	Input	part occurrence, or segment to exclude in traversal (may be NULL_TAG). This is the object traversed in order to get to the current traversal object.
int *	**num_conns**	Output	number of connected objects
tag_t * *	**next_conns**	Output to UF_*free*	array of connected objects. Will NOT contain any NULL_TAGs. Use UF_free() to free.
tag_t * *	**next_objs**	Output to UF_*free*	array of segments or part occurrences. Can have NULL_TAGs. These are the objects to traverse in order to get to the next set of connections. Use UF_free() to free.

UF_ROUTE_init_custom_app (view source)

Defined in: uf_route_ugopenint.h

Overview
Initializes a Routing custom application.

Environment
Internal

See Also
Please refer to the example

History
Original release was in V14.0.

Required License(s)
routing_base

```
int UF_ROUTE_init_custom_app
(
    void
)
```

UF_ROUTE_is_part_anchor (view source)

Defined in: uf_route.h

Overview
Determine if the supplied object is a Routing anchor.

Environment
Internal and External

History
Original release was in V19.0.

Required License(s)
gateway

```
int UF_ROUTE_is_part_anchor
(
    tag_t object,
    logical * is_anchor
)
```

tag_t	**object**	Input	Object to query.
logical *	**is_anchor**	Output	True if the object is a Routing part anchor; false otherwise

UF_ROUTE_is_part_fabrication (view source)

Defined in: uf_route.h

Overview
Determines if the given part is a Routing fabrication part.

Environment
Internal and External

See Also
UF_ROUTE_ask_part_part_type

History
Original release was in V14.0.

Required License(s)
(routing_base or routing_advanced)

```
int UF_ROUTE_is_part_fabrication
(
    tag_t fab_part,
    logical * fab
)
```

tag_t	**fab_part**	Input	The tag of part or part occurrence to be inquired.
logical *	**fab**	Output	Logical set to TRUE if the part is a Routing fabrication part and FALSE otherwise.

UF_ROUTE_is_part_occ_route_part (view source)

Defined in: uf_route.h

Overview

Returns True if the given part occurrence is a part occurrence of a Routing part. A part is considered a Routing part if a Routing object, for example, an RCP, or Port, or Segment, etc., exists in the part.

Return

Return code:
True = The object is a routing part occ that contains one or more routing objects.
False = The object occurrence is not a routing part.

Environment

Internal and External

History

Original release was in V14.0.

Required License(s)

gateway

```
logical UF_ROUTE_is_part_occ_route_part
(
    tag_t obj_id
)
```

tag_t	**obj_id**	Input	Object identifier of the routing part occurrence object.

UF_ROUTE_is_port_connected (view source)

Defined in: uf_route.h

Overview

Ask the if port is connected.

Environment

Internal and External

Required License(s)

gateway

```
int UF_ROUTE_is_port_connected
(
    tag_t port_tag,
    logical * is_connected
)
```

tag_t	**port_tag**	Input	Tag of the port examined.
logical *	**is_connected**	Output	true if connected else false

---

UF_ROUTE_is_port_fixture_port (view source)

Defined in: uf_route.h

Overview
Determine if the supplied port is a fixture port. Fixture ports differ from fitting ports in that they do not subdivide the segment upon which the part containing the port is placed. They are used to model the "connection" point of "fixture" type parts such as clamps or other support fixtures.

Environment
Internal and External

History
Original release was in V17.0.

Required License(s)
routing_base

```
int UF_ROUTE_is_port_fixture_port
(
    tag_t port,
    logical * is_fixture
)
```

tag_t	**port**	Input	Port to query.
logical *	**is_fixture**	Output	True if the port is a fixture port and false otherwise

---

UF_ROUTE_is_port_multi (view source)

Defined in: uf_route.h

Overview
Query if a port is a multiport.

Return
TRUE if the port is a multiport
FALSE otherwise

Environment
Internal and External

See Also
UF_ROUTE_is_port_terminal

History
Originally released in V16.0

Required License(s)
gateway

```
logical UF_ROUTE_is_port_multi
(
    tag_t port
)
```

tag_t	**port**	Input	tag of port to query

UF_ROUTE_is_port_terminal (view source)

Defined in: uf_route.h

Overview
Query if a port is a terminal port.

Return
TRUE if the port is terminal
FALSE otherwise

Environment
Internal and External

See Also
UF_ROUTE_is_port_multi

History
Originally released in V16.0

Required License(s)
gateway

```
logical UF_ROUTE_is_port_terminal
(
    tag_t port
)
```

tag_t	**port**	Input	tag of port to query

UF_ROUTE_is_rcp_bend_seg_rcp (view source)

Defined in: uf_route.h

Overview
Inquires whether the given RCP is an RCP on the end of a bend segment. This is not the same as the bend RCP.

Return
Return code:
TRUE = RCP is on a bend segment
FALSE = RCP is not on a bend segment

Environment
Internal and External

Required License(s)
gateway

```
logical UF_ROUTE_is_rcp_bend_seg_rcp
(
    tag_t candidate,
    tag_p_t corner
)
```

tag_t	**candidate**	Input	Candidate RCP
tag_p_t	**corner**	Output	Corner object or NULL_TAG

UF_ROUTE_is_rcp_miter_corner (view source)

Defined in: uf_route.h

Overview
Inquires whether the given RCP has a miter corner assigned to it.

Return
Return code:
TRUE = RCP has miter assigned
FALSE = RCP has no miter assigned

Environment
Internal and External

Required License(s)
gateway

```
logical UF_ROUTE_is_rcp_miter_corner
(
    tag_t rcp
)
```

tag_t	**rcp**	Input	Tag of RCP

UF_ROUTE_is_segment (view source)

Defined in: uf_route.h

Overview
UF_ROUTE_is_segment

DESCRIPTION:
Used to determine if an NX entity is a routing segment.

Environment
Internal and External

History
Released in V20

Required License(s)
gateway

```
int UF_ROUTE_is_segment
(
    tag_t object,
    logical * is_segment
)
```

tag_t	**object**	Input	entity to check
logical *	**is_segment**	Output	Is entity a routing segment?

UF_ROUTE_is_segment_inside_part (view source)

Defined in: uf_route.h

Overview
Inquires whether the segment is within the extent of the given part occurrence.

Environment
Internal and External

History
Original release was in V15.0.

Required License(s)
gateway

```
logical UF_ROUTE_is_segment_inside_part
(
    tag_t segment,
    tag_t part_occ
)
```

tag_t	**segment**	Input	Tag of segment
tag_t	**part_occ**	Input	Part occurrence

UF_ROUTE_is_stock_equal (view source)

Defined in: uf_route.h

Overview
Tests whether two stock objects are equivalent. Two stock objects are equivalent if they reference the same stock data, the same anchor, and all characteristic values are equal.

Return
Return code:
TRUE = Stock objects are equivalent
FALSE = Stock objects are not equivalent

Environment
Internal and External

History
Original release was in V15.0.

Required License(s)
gateway

```
logical UF_ROUTE_is_stock_equal
(
    tag_t stock1,
    tag_t stock2
)
```

tag_t	**stock1**	Input	Tag of stock object
tag_t	**stock2**	Input	Tag of stock object

UF_ROUTE_is_stock_interior (view source)

Defined in: uf_route.h

Overview
Function Name: UF_ROUTE_is_stock_interior

Function Description: This function queries a stock to see if it is "inside" any part in the current assembly. The stock is only interior if it's segment(s) are interior to a part.

Environment

Internal and External

History
Released in V20

Required License(s)
routing_base

```
int UF_ROUTE_is_stock_interior
(
    tag_t stock,
    logical * is_interior
)
```

tag_t	**stock**	Input	the tag of the stock to query
logical *	**is_interior**	Output	TRUE if interior, else FALSE

UF_ROUTE_is_terminal_segment (view source)

Defined in: uf_route.h

Overview
Ask a passed segment if it is a terminal segment.

Environment
Internal and External

History
Released in V17.0

Required License(s)
gateway

```
int UF_ROUTE_is_terminal_segment
(
    tag_t segment,
    logical * is_term
)
```

tag_t	**segment**	Input	, the segment to query
logical *	**is_term**	Output	, true if it is terminal

UF_ROUTE_is_wire_on_segment (view source)

Defined in: uf_route.h

Overview

Returns true if the given wire traverses through the given segment.

Environment
Internal and External

History
Released in V17.0

Required License(s)
(routing_base or routing_advanced)

```
int UF_ROUTE_is_wire_on_segment
(
    tag_t wire,
    tag_t segment,
    logical * on_seg
)
```

tag_t	**wire**	Input	, wire
tag_t	**segment**	Input	, seg
logical *	**on_seg**	Output	, true if wire traverses segment

UF_ROUTE_load_app_view (view source)

Defined in: uf_route.h

Overview
Loads an application view structure from a data file. It does not set the Application View. Multiple Application Views may be loaded but only one of them can be made current. Call UF_ROUTE_set_current_app_view to set this Application View in the session. Since memory is allocated in this function, UF_ROUTE_unload_app_view must be called to free memory. Do NOT use UF_free to free this memory.

If only a simple file name is specified in the filename variable, the data file is looked for in:

UGII_ROUTING_APP_VIEW_DIR (An environment variable)
UGII_BASE_DIR/ugrouting/

Environment
Internal and External

See Also
UF_ROUTE_set_current_app_view
UF_ROUTE_unload_app_view

Please refer to the example

Required License(s)
routing_base

```
int UF_ROUTE_load_app_view
(
    char * filename,
    UF_ROUTE_application_view_p_t * app_view
)
```

char *	**filename**	Input	Name of the application view definition file.
UF_ROUTE_application_view_p_t *	**app_view**	Output to UF_*free*	Filled application view data. This must be freed by calling UF_ROUTE_unload_app_view.

UF_ROUTE_load_app_view_list (view source)

Defined in: uf_route.h

Overview
Load the list of application view descriptions.

Environment
Internal and External

Required License(s)
routing_base

```
int UF_ROUTE_load_app_view_list
(
    int * num_app_views,
    UF_ROUTE_app_view_desc_p_t * app_views
)
```

int *	**num_app_views**	Output	num_app_views - Number of application views
UF_ROUTE_app_view_desc_p_t *	**app_views**	Output to UF_*free*	app_views - Allocated array of application view descriptions (Array of Name/File pair structures). This array should be freed by calling UF_free.

UF_ROUTE_load_part_by_charx (view source)

Defined in: uf_route.h

Overview
Loads the given part, or part family member if member name is among the list of characteristics, into the current session. The part is loaded but NOT made the current work part. The returned part tag

may subsequently be used by the Assembly functions to add this part as a component to the assembly for later "placement" within the routing.

If the part is already loaded the routine finds the part. If member name is among the list of characteristics, the routine will establish it if it is an instance of the part.

Environment

Internal and External

History

Original release was in V14.0.

Required License(s)

routing_base

```
int UF_ROUTE_load_part_by_charx
(
    int num_charx,
    UF_ROUTE_charx_t charx [ ] ,
    tag_t * part
)
```

int	**num_charx**	Input	Number of characteristics.
UF_ROUTE_charx_t	**charx [ ]**	Input	Array of characteristics associated with a part
tag_t *	**part**	Output	Part tag of the loaded part or NULL_TAG if it was not loaded.

UF_ROUTE_load_part_by_name (view source)

Defined in: uf_route.h

Overview

Loads the given part, or part family member if member name is not NULL, into the current session. The part is loaded but NOT made the current work part. The returned part tag may subsequently be used by the Assembly functions to add this part as a component to the assembly for later "placement" within the routing.

Environment

Internal and External

See Also

UF_ROUTE_set_part_in_stock
For example Please refer to the example

History

Original release was in V13.0.

Required License(s)

routing_base

```
int UF_ROUTE_load_part_by_name
(
    char * part_name,
    char * member_name,
    tag_t * part
)
```

char *	**part_name**	Input	Name of the part to load. This may be the name of a simple part file or the name of a Part Family part.
char *	**member_name**	Input	Part Family member name if the part specified by part_name is a Part Family, else NULL.
tag_t *	**part**	Output	Part tag of the loaded part or NULL_TAG if it was not loaded.

UF_ROUTE_load_stock_by_charx (view source)

Defined in: uf_route.h

Overview

This function uses the supplied characteristics (charx) to locate or retrieve the corresponding stock data. The structure input to this function (specifying the stock charx) must contain a PART_NAME charx and, if the part is a PART FAMILY part, the MEMBER_NAME.

The output of a call to UF_ROUTE_match_charx_in_plib may be used to load the UF_ROUTE_part_lib_part_p_t structure.

If stock matching the given charx already exists within the work part, the tag of the stock data, anchor, and cross section objects matching the charx, anchor name, and style given as input will be returned.

If there is no stock data object in the work part matching the supplied charx, the PART_NAME (and optionally MEMBER_NAME) charx is used to locate the stock part.

The stock data information from this stock part is then imported (retrieved) into the current work part and the tag of the stock data object is returned. Also returned are the tag of the anchor and cross section objects which match the given anchor name and stock style.

In either situation, the stock data, anchor, and cross section objects may then be used to assign stock of this type to segments within the routing.

Environment

Internal and External

Required License(s)

routing_base

```
int UF_ROUTE_load_stock_by_charx
(
    UF_ROUTE_part_lib_part_p_t stock,
    char * anchor_name,
    int stock_style,
    tag_t * stock_data_tag,
    tag_t * anchor_tag,
    tag_t * cross_tag
)
```

UF_ROUTE_part_lib_part_p_t	**stock**	Input	Pointer to a UF_ROUTE_part_lib_part_p_t structure filled in with the PART_NAME, MEMBER_NAME, and other characteristics corresponding to the stock to be used.
char *	**anchor_name**	Input	the name of the anchor in the stock data to be used. May be NULL.
int	**stock_style**	Input	the stock style integer which determines which cross section of the stock data to use.One of: UF_ROUTE_STYLE_NONE UF_ROUTE_STYLE_SIMPLE UF_ROUTE_STYLE_DETAIL
tag_t *	**stock_data_tag**	Output	the tag of the stock data object
tag_t *	**anchor_tag**	Output	the tag of the anchor object
tag_t *	**cross_tag**	Output	the tag of the cross section object

---

UF_ROUTE_load_stock_data (view source)

Defined in: uf_route.h

Overview

Loads the stock data into the current part. The stock data tag can be used to assign stocks to segments using UF_ROUTE_assign_stock. Once the stock has been "loaded", several calls to UF_ROUTE_assign_stock may be made (for various segments) without the need to "load" another.

The Assembly Search Directory list, specified via the load_options.def file or interactively through the File -->Options-->Load Options dialog, is used to locate the part file for the stock.

Environment

Internal and External

See Also

UF_ROUTE_assign_stock
Please refer to the example

History

Original release was in V13.0.

Required License(s)

routing_base

```
int UF_ROUTE_load_stock_data
(
    char * part_name,
    char * member_name,
    int stock_style,
    tag_t * stock_data_tag,
    tag_t * anchor_tag,
    tag_t * cross_tag
)
```

char *	**part_name**	Input	Name of the stock part family.
char *	**member_name**	Input	Name of the member part in the stock part family or NULL if part_name is not a Part Family.
int	**stock_style**	Input	Stock style can have value of UF_ROUTE_STYLE_NONE UF_ROUTE_STYLE_SIMPLE or UF_ROUTE_STYLE_DETAIL
tag_t *	**stock_data_tag**	Output	Object Identifier of the loaded stock_data.
tag_t *	**anchor_tag**	Output	Object Identifier of the anchor of the stock.
tag_t *	**cross_tag**	Output	Object Identifier of the cross_section of the stock.

UF_ROUTE_match_charx_in_plib (view source)

Defined in: uf_route.h

Overview

Matches part entries in the part library based on the given set of characteristics. If start is not NULL, it should be the name of the node in the part library hierarchy at which to start the search.

Environment

Internal and External

See Also

UF_ROUTE_free_match_results

History

Original release was in V14.0.

Required License(s)

gateway

```
int UF_ROUTE_match_charx_in_plib
(
    char * start,
    int num_criteria,
    UF_ROUTE_charx_p_t criteria,
    int * num_matches,
    UF_ROUTE_part_lib_part_p_t * matches
)
```

char *	**start**	Input	Name of the start node in the part library view tree.
int	**num_criteria**	Input	Number of characteristics being matched
UF_ROUTE_charx_p_t	**criteria**	Input	Array of match characteristics.
int *	**num_matches**	Output	Number of matches found.
UF_ROUTE_part_lib_part_p_t *	**matches**	Output to UF_*free*	Array of matches. To be freed using UF_ROUTE_free_match_results.

UF_ROUTE_merge_rcps (view source)

Defined in: uf_route.h

Overview

Function to merge duplicate RCPs occurring at the same location.
The function will attempt to merge the coincident RCPs according to
Merge Rules. It will output the count and tags of RCPs remaining after the
merging. Any of the input objects that are not RCPs will be added to
the set of remaining objects.
All mergeable RCPs will be merged to the highest priority RCP determined
according to the merge rules - unless a preferred RCP is explicitly specified.
The function will not check to make sure that the RCPs to be merged lie at
the same location - it is the caller's responsibility to ensure that!. After
merging, the input list of RCP tags may contain tags of objects that no longer
exist - caller should free that list immediately after this function returns.
Returns a non-zero error code if any occurs, during the operation.
Note : Caller should perform a Model Update using UF_MODL_update() after
merging a set of RCPs to ensure data model consistency.

MERGE RULES

# The term Merging refers to the transfer of all dependencies (links) from one
object to another and the subsequent deletion of the obsoleted object.
# RCPs to be merged will be prioritized according to their derive methods and have
priority values defined as below
o Port (4) - i.e. derived from a port occurrence
o Point (3) - i.e. derived from smart points
o RCP (2) - i.e. derived from an existing RCP
o Absolute (1) - i.e. derived from absolute coordinates
# A Port RCP will have the highest priority and the Absolute derive method will
have the least.
# All duplicate RCPs occurring at a given location will be merged with the
highest priority RCP among them.
# Bend, Miter and Cope Corner RCPs cannot be merged together or with any other RCP type.
# Bend segment RCPs cannot be merged together or with any other RCP type.

```
--------------------------------------------------------------------------------
| | PORT | POINT | RCP | ABSOLUTE |
|--------------------------------------------------------------------------------
```

| PORT | o Merge | o Merge | o Merge | o Merge |
	o Retain Port	o Retain Port	o Retain Port	o Retain Port
POINT	o Merge	o Merge	o Merge	o Merge
	o Retain Port	o Retain Point	o Retain Point	o Retain Point
RCP	o Merge	o Merge	o Merge	o Merge
	o Retain Port	o Retain Point	o Retain RCP	o Retain RCP
ABSOLUTE	o Merge	o Merge	o Merge	o Merge
	o Retain Port	o Retain Point	o Retain RCP	o Retain Abs.
---|

Environment
Internal and External

History
New in V17

Required License(s)
(routing_base or routing_advanced)

```
int UF_ROUTE_merge_rcps
(
    int num_rcps,
    tag_t * rcps,
    tag_t preferred_rcp,
    int * num_remaining,
    tag_t * * remaining
)
```

int	**num_rcps**	Input	Length of the array of coincident RCPs to be merged.
tag_t *	**rcps**	Input	Array of coincident RCP tags to be merged. Should be Prototype tags.
tag_t	**preferred_rcp**	Input	If specified, all RCPs will be merged to this RCP. If a NULL_TAG, the routine will determine what gets retained according to its merge rules. No merge will occur if this object is not an RCP.
int *	**num_remaining**	Output	Number of RCPs remaining
tag_t * *	**remaining**	Output to UF_*free*	RCPs remaining after merge operations, including those that couldn't be merged Free using UF_free

UF_ROUTE_register_custom_app (view source)

Defined in: uf_route_ugopenint.h

Overview
Registers a Routing custom application. This routine should be called just after UF_MB_register_application().

Environment
Internal

See Also
Please refer to the example

History
Original release was in V14.0.

Required License(s)
routing_base

int UF_ROUTE_register_custom_app
(
    int app_id
)

int	**app_id**	Input	Application ID for this application. This is returned from UF_MB_register_application.

UF_ROUTE_remove_corner (view source)

Defined in: uf_route.h

Overview
Removes the Corner, be it a Bend or a Miter.

Environment
Internal and External

History
Original release was in V13.0.

Required License(s)
routing_base

int UF_ROUTE_remove_corner
(
    tag_t corner
)

tag_t	**corner**	Input	Object identifier of the Corner, Bend or Miter.

UF_ROUTE_remove_seg_from_stock (view source)

Defined in: uf_route.h

Overview

Removes the given segment from all of the given stock objects.

Environment
Internal and External

History
Original release was in V15.0.

Required License(s)
(routing_base or routing_advanced)

```
int UF_ROUTE_remove_seg_from_stock
(
    tag_t seg,
    int num_stock,
    tag_t * stock
)
```

tag_t	**seg**	Input	Tag of segment
int	**num_stock**	Input	Count of stock objects
tag_t *	**stock**	Input	Array of stock objects

UF_ROUTE_remove_stock (view source)

Defined in: uf_route.h

Overview
Removes the stock from the given segments. The segments need not belong to the same path. An update of routing objects is performed after removal of the stock by this routine.

Environment
Internal and External

History
Original release was in V13.0.

Required License(s)
routing_base

```
int UF_ROUTE_remove_stock
(
    int num_segs,
    tag_t * segments
)
```

int	**num_segs**	Input	Number of segment objects.
tag_t *	**segments**	Input	An array of segment objects.

UF_ROUTE_remove_terminal_ports (view source)

Defined in: uf_route.h

Overview
Remove terminal ports from the multiport

Environment
Internal and External

See Also
UF_ROUTE_remove_virtual_ports

History
Originally released in V16.0

Required License(s)
routing_base

```
int UF_ROUTE_remove_terminal_ports
(
    tag_t multi,
    int num_terms,
    tag_t * terms
)
```

tag_t	**multi**	Input	multiport
int	**num_terms**	Input	Number of terminal ports
tag_t *	**terms**	Input	Array of terminal port tags

UF_ROUTE_remove_virtual_ports (view source)

Defined in: uf_route.h

Overview
Remove virtual ports from the multiport

Environment
Internal and External

See Also
UF_ROUTE_remove_terminal_ports

History
Originally released in V16.0

Required License(s)
routing_base

int UF_ROUTE_remove_virtual_ports
(
    tag_t multi,
    int num_terms,
    char * * terms
)

tag_t	**multi**	Input	multiport
int	**num_terms**	Input	Number of virtual ports
char * *	**terms**	Input	Array of virtual ports unique ids

UF_ROUTE_reuse_stock_part (view source)

Defined in: uf_route.h

Overview

This function is used to replace a stock component with an equivalent stock component. This functions main purpose is for component reuse with stock parts.

The function first calls the user exit for finding a part for reuse, and
then replaces all instances in the work part, of that part with the returned part.
If no part is returned from the user exit then nothing is done.

The input to this function is the tag of the stock object.

Environment

Internal and External

History

Released in V19.0

Required License(s)

routing_base

int UF_ROUTE_reuse_stock_part
(
    tag_t stock
)

tag_t	**stock**	Input	Tag of the stock object.

UF_ROUTE_RUN_ask_from_items (view source)

Defined in: uf_route_run.h

Overview

Retrieves the from items of a run. All from items are extracted ports.
Caller should free the array of from items

Environment
Internal and External

See Also
UF_free()

History
Released in NX3.0

Required License(s)
gateway

```
int UF_ROUTE_RUN_ask_from_items
(
    tag_t run,
    int * n_from_items,
    tag_t * * from_items
)
```

tag_t	**run**	Input	Run object to inquire
int *	**n_from_items**	Output	number of from items
tag_t * *	**from_items**	Output to UF_*free*	array of from items. All from items are extracted ports. Must be freed with UF_free()

UF_ROUTE_RUN_ask_member_items (view source)

Defined in: uf_route_run.h

Overview
Retrieves the member items of a run. All member items are either extracted ports or route segments. Caller should free the array of member items.

Environment
Internal and External

See Also
UF_free()

History
Released in NX3.0

Required License(s)
gateway

```
int UF_ROUTE_RUN_ask_member_items
(
    tag_t run,
    int * n_member_items,
    tag_t * * member_items
)
```

tag_t	**run**	Input	Run object to inquire
int *	**n_member_items**	Output	number of member items
tag_t * *	**member_items**	Output to UF_*free*	array of member items. All member items are either extracted ports or route segments Must be freed with UF_free()

UF_ROUTE_RUN_ask_run_id_and_type (view source)

Defined in: uf_route_run.h

Overview

Retrieves the run id and run type of a given run. Caller must free the memory.

Environment

Internal and External

See Also

UF_free()

History

Released in NX3.0

Required License(s)

gateway

```
int UF_ROUTE_RUN_ask_run_id_and_type
(
    tag_t run,
    char * * run_id,
    char * * run_type
)
```

tag_t	**run**	Input	Run object to inquire
char * *	**run_id**	Output to UF_*free*	run_id of the run object, must be freed with UF_free()
char * *	**run_type**	Output to UF_*free*	run_type of run, must be freed with UF_free

---

UF_ROUTE_RUN_ask_runs_in_part (view source)

Defined in: uf_route_run.h

Overview

Retrieves all run objects in given part. Caller should free the run array.
If input part is NULL_TAG, current work part is assumed.

Environment
Internal and External

See Also
UF_free()

History
Released in NX3.0

Required License(s)
gateway

```
int UF_ROUTE_RUN_ask_runs_in_part
(
    tag_t part,
    int* n_runs,
    tag_t* * runs
)
```

tag_t	**part**	Input	Part to ask the run objects, if NULL_TAG, current work part is assumed
int*	**n_runs**	Output	number of runs in the given part
tag_t* *	**runs**	Output to UF_*free*	array of runs. Must be freed with UF_free()

UF_ROUTE_RUN_ask_to_items (view source)

Defined in: uf_route_run.h

Overview
Retrieves the to items of a run. All to items are extracted ports.
Caller should free the array of to items

Environment
Internal and External

See Also
UF_free()

History
Released in NX3.0

Required License(s)
gateway

```
int UF_ROUTE_RUN_ask_to_items
(
    tag_t run,
    int * n_to_items,
    tag_t * * to_items
)
```

tag_t	**run**	Input	Run object to inquire
int *	**n_to_items**	Output	number of to items
tag_t * *	**to_items**	Output to UF_*free*	array of to items. All to items are extracted ports. Must be freed with UF_free()

UF_ROUTE_RUN_edit_run (view source)

Defined in: uf_route_run.h

Overview

Edit a given run. From and To items must be extracted ports or rcps.
Member items must be extracted ports or route segments.

Environment

Internal and External

History

Released in NX3.0.2.3

Required License(s)

routing_base

```
int UF_ROUTE_RUN_edit_run
(
    tag_t run,
    char* run_id,
    char* run_type,
    int n_from_items,
    tag_t* from_items,
    int n_to_items,
    tag_t* to_items,
    int n_member_items,
    tag_t* member_items
)
```

tag_t	**run**	Input	Run object to edit
char*	**run_id**	Input	Run id of the run after edit
char*	**run_type**	Input	Run type of the run after edit
int	**n_from_items**	Input	Number of From items
tag_t*	**from_items**	Input	Array of From items
int	**n_to_items**	Input	Number of To items
tag_t*	**to_items**	Input	Array of To items
int	**n_member_items**	Input	Number of Member items

tag_t*	**member_items**	Input	Array of Member items

UF_ROUTE_RUN_set_run_id (view source)

Defined in: uf_route_run.h

Overview
Set the run id of given run.

Environment
Internal and External

History
Released in NX3.0.2.3

Required License(s)
routing_base

```
int UF_ROUTE_RUN_set_run_id
(
    tag_t run,
    char* run_id
)
```

tag_t	**run**	Input	Run to change the run id of
char*	**run_id**	Input	New run id of the run

UF_ROUTE_RUN_set_run_type (view source)

Defined in: uf_route_run.h

Overview
Set the run type of given run.

Environment
Internal and External

History
Released in NX3.0.2.3

Required License(s)
routing_base

```
int UF_ROUTE_RUN_set_run_type
(
    tag_t run,
    char* run_type
)
```

tag_t	**run**	Input	Existing valid run
char*	**run_type**	Input	New run type of the run

UF_ROUTE_set_built_in_path_objs (view source)

Defined in: uf_route.h

Overview
Modify the curves in a built-in path

Environment
Internal and External

History
New in V17

Required License(s)
routing_base

```
int UF_ROUTE_set_built_in_path_objs
(
    tag_t bip,
    int num_objs,
    tag_t * objs
)
```

tag_t	**bip**	Input	Tag of built-in path
int	**num_objs**	Input	Number of built-in path curves
tag_t *	**objs**	Input	Array of curve tags

UF_ROUTE_set_characteristics (view source)

Defined in: uf_route.h

Overview
Sets the characteristics to the given routing object.

Environment
Internal and External

See Also
Please refer to the example

History
Original release was in V13.0.

Required License(s)

routing_base

```
int UF_ROUTE_set_characteristics
(
    tag_t obj_id,
    int charx_count,
    UF_ROUTE_charx_p_t list
)
```

tag_t	**obj_id**	Input	Object identifier of the routing object.
int	**charx_count**	Input	Count of the characteristics.
UF_ROUTE_charx_p_t	**list**	Input	List of all characteristics.

UF_ROUTE_set_charx_env (view source)

Defined in: uf_route.h

Overview
Replaces the current characteristic environment with the supplied set of characteristics.

Environment
Internal and External

History
Original release was in V14.0.

Required License(s)
routing_base

```
int UF_ROUTE_set_charx_env
(
    int num_charx,
    UF_ROUTE_charx_t charx [ ]
)
```

int	**num_charx**	Input	Number of characteristics supplied
UF_ROUTE_charx_t	**charx [ ]**	Input	Array of characteristics supplied.

UF_ROUTE_set_current_app_view (view source)

Defined in: uf_route.h

Overview
Sets the current Application View for the current session. This function must be called after loading an Application View using

UF_ROUTE_load_app_view.

Environment
Internal and External

See Also
UF_ROUTE_load_app_view
Please refer to the example

History
Original release was in V13.0.

Required License(s)
routing_base

int UF_ROUTE_set_current_app_view
(
UF_ROUTE_application_view_t * app_view
)

UF_ROUTE_application_view_t *	**app_view**	Input	Sets the current Application View and remains active till set again. If the current Application View is unloaded reset the current Application View to NULL.

UF_ROUTE_set_part_in_stock (view source)

Defined in: uf_route.h

Overview
This function "sets" the given part occurrence into the routing. This routine determines which Segments the part occurrence lies upon and matches any Ports from the part to these Segments. The Segments are subdivided at these points and any stock along the Segments is connected to the Ports of the part. The part should be placed at the proper position and orientation using UF_ASSEM calls.

Environment
Internal and External

See Also
Please refer to the example

History
Original release was in V13.0.

Required License(s)
routing_base

int UF_ROUTE_set_part_in_stock
(
tag_t occ
)

tag_t	**occ**	Input	Occurrence id of the part in the current work part.

UF_ROUTE_set_part_search_path (view source)

Defined in: uf_route.h

Overview
Sets the tag of a search path.

Environment
Internal and External

See Also
UF_DIRPATH_create_from_env
UF_DIRPATH_create_from_dirs
UF_DIRPATH_append
UF_DIRPATH_append_from_dirs
UF_DIRPATH_append_from_env

History
Original release was in V15.0.

Required License(s)
routing_base

```
int UF_ROUTE_set_part_search_path
(
    tag_t dirpath
)
```

tag_t	**dirpath**	Input	tag of search path

UF_ROUTE_set_port_back_extension (view source)

Defined in: uf_route.h

Overview
This function sets the back extension value for a port.

Environment
Internal and External

History
Released in V18.0

Required License(s)
routing_base

```
int UF_ROUTE_set_port_back_extension
(
    tag_t port,
    double ext
)
```

tag_t	**port**	Input	, the port
double	**ext**	Input	, the port extension

UF_ROUTE_set_port_back_extension_obj (view source)

Defined in: uf_route.h

Overview
This function sets the back extension object for a port.
The object must be a UF_scalar_type.

Environment
Internal and External

History
Released in V18.0

Required License(s)
routing_base

```
int UF_ROUTE_set_port_back_extension_obj
(
    tag_t port,
    tag_t ext
)
```

tag_t	**port**	Input	, the port
tag_t	**ext**	Input	, the port back extension object

UF_ROUTE_set_port_clock_increment (view source)

Defined in: uf_route.h

Overview
This function sets the clock angle increment value for a port.

Environment
Internal and External

History
Released in V18.0

Required License(s)

routing_base

```
int UF_ROUTE_set_port_clock_increment
(
    tag_t port,
    double increment
)
```

tag_t	**port**	Input	The port to modify.
double	**increment**	Input	The increment of the clock angle.

UF_ROUTE_set_port_engagement (view source)

Defined in: uf_route.h

Overview
This function sets the engagement value for a port.

Environment
Internal and External

History
Released in V18.0

Required License(s)
routing_base

```
int UF_ROUTE_set_port_engagement
(
    tag_t port,
    double eng
)
```

tag_t	**port**	Input	, the port
double	**eng**	Input	, the port engagement

UF_ROUTE_set_port_engagement_obj (view source)

Defined in: uf_route.h

Overview
This function sets the engagement object for a port.
The object must be a UF_scalar_type.

Environment
Internal and External

History
Released in V18.0

Required License(s)
routing_base

```
int UF_ROUTE_set_port_engagement_obj
(
    tag_t port,
    tag_t eng
)
```

tag_t	**port**	Input	, the port
tag_t	**eng**	Input	, the port engagement object

UF_ROUTE_set_port_extension (view source)

Defined in: uf_route.h

Overview
This function sets the extension value for a port.

Environment
Internal and External

History
Released in V18.0

Required License(s)
routing_base

```
int UF_ROUTE_set_port_extension
(
    tag_t port,
    double ext
)
```

tag_t	**port**	Input	, the port
double	**ext**	Input	, the port extension

UF_ROUTE_set_port_extension_obj (view source)

Defined in: uf_route.h

Overview
This function sets the extension object for a port.
The object must be a UF_scalar_type.

Environment
Internal and External

History
Released in V18.0

Required License(s)
routing_base

```
int UF_ROUTE_set_port_extension_obj
(
    tag_t port,
    tag_t ext
)
```

tag_t	**port**	Input	, the port
tag_t	**ext**	Input	, the port extension object

UF_ROUTE_set_port_id (view source)

Defined in: uf_route.h

Overview
UF_ROUTE_set_port_id

DESCRIPTION:
Used to set the unique id charx of given port.
Return value : 0 ==> OK, != 0 ==> Error
-2 ==> Illegal object type
-5 ==> Unique id charx of port assignment operation failure.

Environment
Internal and External

History
Released in NX

Required License(s)
routing_base

```
int UF_ROUTE_set_port_id
(
    tag_t obj_id,
    char * port_id
)
```

tag_t	**obj_id**	Input	port object
char *	**port_id**	Input	unique id charx of port to be set

UF_ROUTE_set_port_lock_rotation_flag (view source)

Defined in: uf_route.h

Overview

This function is used to set the lock rotation flag of a port lock.
If the passed in flag is true, and the ports involved in the lock
both have rotation vectors, then the child component will not be
able to rotate. If the flag is false, then the child component
of the lock will be able to rotate freely.

If an error occurs, UF_ROUTE_err_invalid_port_mate will be returned.

Environment

Internal and External

History

Released in V18.0

Required License(s)

routing_base

```
int UF_ROUTE_set_port_lock_rotation_flag
(
    tag_t port_occ,
    logical rotation_locked
)
```

tag_t	**port_occ**	Input	The FROM or TO port occurrence
logical	**rotation_locked**	Input	Lock rotation flag.

UF_ROUTE_set_port_rot_by_point (view source)

Defined in: uf_route.h

Overview

Sets the rotation vector of a port by passing a point at which the
rotation vector should point.

Environment

Internal and External

History

Original release was in V15.0.

Required License(s)

routing_base

```
int UF_ROUTE_set_port_rot_by_point
(
    double pnt_pos [ 3 ] ,
    tag_t port_tag
```

)

double	**pnt_pos [3]**	Input	Position for rotation
tag_t	**port_tag**	Input	Tag of port

UF_ROUTE_set_stock_part_name (view source)

Defined in: uf_route.h

Overview
This function is used to change the name of a stock component part.

Environment
Internal and External

History
Released in V19.0

Required License(s)
routing_base

```
int UF_ROUTE_set_stock_part_name
(
    tag_t stock,
    char* part_name
)
```

tag_t	**stock**	Input	Tag of the stock object
char*	**part_name**	Input	New name of the stock component

UF_ROUTE_set_stock_style (view source)

Defined in: uf_route.h

Overview
Allows stock style to be assigned to the stock of segments.

Environment
Internal and External

History
Original release was in V14.0.

Required License(s)
(routing_base or routing_advanced)

```
int UF_ROUTE_set_stock_style
(
    int new_style,
    int num_stocks,
    tag_t * stock_tags
)
```

int	**new_style**	Input	New setting for the stock style. May be set to: UF_ROUTE_STYLE_NONE, UF_ROUTE_STYLE_SIMPLE, UF_ROUTE_STYLE_DETAIL
int	**num_stocks**	Input	Number of stock objects
tag_t *	**stock_tags**	Input	Array of stock tags for which style has to be modified.

UF_ROUTE_set_user_preferences (view source)

Defined in: uf_route.h

Overview
Sets the values of the user preferences.

Environment
Internal and External

History
Original release was in V14.0.

Required License(s)
routing_base

```
int UF_ROUTE_set_user_preferences
(
    int n_prefs,
    UF_ROUTE_user_preference_p_t prefs
)
```

int	**n_prefs**	Input	Number of preferences to modify/add.
UF_ROUTE_user_preference_p_t	**prefs**	Input / Output	Array of preference structures.

UF_ROUTE_simplify_rcps (view source)

Defined in: uf_route.h

Overview
Of the given RCPs, remove any that are unnecessary, i.e. they are placed in a straight line with the adjacent RCPs.

Environment
Internal and External

Required License(s)
(routing_base or routing_advanced)

```
int UF_ROUTE_simplify_rcps
(
    int count,
    tag_t rcps [ ]
)
```

int	**count**	Input	Count of RCPs
tag_t	**rcps [ ]**	Input	Array of RCPs

UF_ROUTE_simplify_segments (view source)

Defined in: uf_route.h

Overview
Attempts to combine segments which are colinear and whose intermediate rcp's do not branch.

Environment
Internal and External

History
Original release was in V15.0.

Required License(s)
(routing_base or routing_advanced)

```
int UF_ROUTE_simplify_segments
(
    int count,
    tag_t segments [ ] ,
    int * num_new_segments,
    tag_t * * new_segments
)
```

int	**count**	Input	Count of segments passed
tag_t	**segments [ ]**	Input	Array of segments
int *	**num_new_segments**	Output	Count of segments returned
tag_t * *	**new_segments**	Output to UF_*free*	Array of segments created. Use UF_free to deallocate memory when no longer required.

UF_ROUTE_solve_places (view source)

Defined in: uf_route.h

Overview

Find all the possible ways to position a routing part using
the given placement object. The initial call to this
routine should have num_places==0 and places == NULL. This
routine can then be called multiple times (with different placement
objects) to generate additional solutions.

Environment

Internal and External

Required License(s)

routing_base

```
int UF_ROUTE_solve_places
(
    tag_t placer,
    tag_t part_occ,
    int * num_places,
    UF_ROUTE_place_solution_p_t * * places
)
```

tag_t	**placer**	Input	Tag of the "placement" object. Should be a RCP, arc or circular edge, line, port, or another Routing part occurrence.
tag_t	**part_occ**	Input	Part occurrence to solve
int *	**num_places**	Output	Updated with new number of placements found
UF_ROUTE_place_solution_p_t * *	**places**	Output to UF_*free*	Function_to_free = UF_ROUTE_free_places Additional solutions appended to current list (Initial call should have places == NULL). The "places" structure should be freed with UF_ROUTE_free_places

---

UF_ROUTE_stock_ask_name (view source)

Defined in: uf_route.h

Overview

UF_ROUTE_stock_ask_name

DESCRIPTION:
Return stock name

Return value : 0 ==> OK, != 0 ==> Error

Environment

Internal and External

History

Released in NX

Required License(s)

gateway

```
int UF_ROUTE_stock_ask_name
(
    tag_t stock,
    char* * name
)
```

tag_t	**stock**	Input	stock tag
char* *	**name**	Output to UF_*free*	Stock name

UF_ROUTE_transform_objects (view source)

Defined in: uf_route.h

Overview

Applies either Move or Copy transformations to the given objects.
The operation is determined by the value of the operation flag.
The transformation matrix from the input is used for transforming
the objects. Returns an error code if any error occurs during the
transformation operation.

Transformation of objects that are smart has no effect. Call UF_SO_is_so
to determine if an object is smart. A segment is considered smart if its
end RCPs are smart.

Environment

Internal and External

History

New in V17

Required License(s)

(routing_base or routing_advanced)

```
int UF_ROUTE_transform_objects
(
    tag_t * tags,
    int num_tags,
    double transform [ 4 ] [ 4 ] ,
    logical copy_operation,
    tag_t * * copy_tags
)
```

tag_t *	**tags**	Input	Array of object tags. In an Assembly context, the tags corresponding to each Object Occurrence to be transformed. Objects may be only of types UF_route_part_type_type or UF_route_control_point_type.
int	**num_tags**	Input	The number of the above tags
double	**transform [4] [4]**	Input	The transformation matrix to use. The structure of the transform: transform[0][0],[0][1],[0][2] - X Axis Rotation vectors [1][0],[1][1],[1][2] - Y Axis Rotation vectors [2][0],[2][1],[2][2] - Z Axis Rotation vectors transform[0][3],[1][3],[2][3] - Translation vector transform[3][3] - Scale
logical	**copy_operation**	Input	TRUE : Copy operation FALSE: Move operation
tag_t * *	**copy_tags**	Output to UF_*free*	The corresponding array of copied tags. There will be a one to one correspondence between tags of objects in the input tags list and the copy_tags array. Will be NULL for a Move operation. For a Copy operation, free using UF_free.

UF_ROUTE_unload_app_view (view source)

Defined in: uf_route.h

Overview

Frees data allocated during loading of the Application View using UF_ROUTE_load_app_view. Once the Application View has been freed then the current Application View must be set to NULL using UF_ROUTE_set_current_app_view.

Environment

Internal and External

See Also

Please refer to the example

Required License(s)

routing_base

```
int UF_ROUTE_unload_app_view
(
    UF_ROUTE_application_view_p_t app_view
)
```

UF_ROUTE_application_view_p_t	**app_view**	Input	Application view data to be freed.

UF_ROUTE_unset_shadow_for_view (view source)

Defined in: uf_route.h

Overview
Unexplode the components in a view.

Environment
Internal and External

Required License(s)
(routing_base or routing_advanced)

```
int UF_ROUTE_unset_shadow_for_view
(
    tag_t view
)
```

tag_t	**view**	Input	Tag of the view to unexplode

UF_ROUTE_update_charx_env (view source)

Defined in: uf_route.h

Overview
Updates the current characteristic environment with the given characteristics by updating existing values in the environment and adding new values.

Environment
Internal and External

Required License(s)
routing_base

```
int UF_ROUTE_update_charx_env
(
    int num_charx,
    UF_ROUTE_charx_t charx [ ]
)
```

int	**num_charx**	Input	Number of characteristics supplied.
UF_ROUTE_charx_t	**charx [ ]**	Input	Array of characteristics

---