UF_VEC2_add (view source)

Defined in: uf_vec.h

Overview
Performs a two dimensional vector addition and returns the vector sum in vec_sum[2].

Return
void

Environment
Internal and External

Required License(s)
gateway

```
void UF_VEC2_add
(
    const double vec1 [ 2 ] ,
    const double vec2 [ 2 ] ,
    double vec_sum [ 2 ]
)
```

const double	**vec1 [2]**	Input	Vector #1
const double	**vec2 [2]**	Input	Vector #2
double	**vec_sum [2]**	Output	The vector sum of vectors one and two vec_sum = vec1 + vec2

UF_VEC2_affine_comb (view source)

Defined in: uf_vec.h

Overview
Performs a vector affine combination which consists of adding an unscaled vector to a scaled vector. The first vector you input is vec[2], which is unscaled. The second vector you input is the vec_to_scale[2], which is scaled by the input argument scale. The resultant vector is output to vec_comb[2].

Return
Void.

Environment
Internal and External

Required License(s)
gateway

```
void UF_VEC2_affine_comb
(
```

```
    const double vec [ 2 ] ,
    double scale,
    const double vec_to_scale [ 2 ] ,
    double vec_comb [ 2 ]
)
```

const double	**vec [2]**	Input	Unscaled vector
double	**scale**	Input	Scale to apply to vec_to_scale[2]
const double	**vec_to_scale [2]**	Input	The second two dimensional vector which is scaled.
double	**vec_comb [2]**	Output	Vector sum of unscaled vector and scaled vector where vec_comb = vec + (scale vec_to_scale)

UF_VEC2_ask_perpendicular (view source)

Defined in: uf_vec.h

Overview
Returns a 2D vector that is perpendicular to the input vector

Return
void.

Environment
Internal and External

Required License(s)
gateway

```
void UF_VEC2_ask_perpendicular
(
    const double vec1 [ 2 ] ,
    double vec_perp [ 2 ]
)
```

const double	**vec1 [2]**	Input	2D vector
double	**vec_perp [2]**	Output	2D vector

UF_VEC2_components (view source)

Defined in: uf_vec.h

Overview
Calculates the scale factors of a linear combination of two vectors that form a third vector. The vector combination is equal to the sum of the two scaled vectors (i.e., vec_comb = scale1 vec1 + scale2 + vec2).

Return

0 = Scale factors can be calculated
1 = Input vectors are parallel

Environment
Internal and External

Required License(s)
gateway

```
int UF_VEC2_components
(
    const double vec1 [ 2 ] ,
    const double vec2 [ 2 ] ,
    const double vec_comb [ 2 ] ,
    double tolerance,
    double * scale1,
    double * scale2
)
```

const double	**vec1 [2]**	Input	First vector of linear combination
const double	**vec2 [2]**	Input	Second vector of linear combination
const double	**vec_comb [2]**	Input	Linear combination of vec1 and vec2
double	**tolerance**	Input	Tolerance value to use for checking whether vec1 and vec2 are parallel
double *	**scale1**	Output	Scale factor of vec1 in linear combination
double *	**scale2**	Output	Scale factor of vec2 in linear combination

---

UF_VEC2_convex_comb (view source)

Defined in: uf_vec.h

Overview
Calculates the position of a point between the two end points of a line segment. The point on the line segment is defined by:
pnt_on_seg = (parameter pnt1) + ((1.0 - parameter) pnt2).

Environment
Internal and External

Return
Void.

Required License(s)
gateway

```
void UF_VEC2_convex_comb
(
    double parameter,
    const double pnt1 [ 2 ] ,
```

```
    const double pnt2 [ 2 ] ,
    double pnt_on_seg [ 2 ]
)
```

double	**parameter**	Input	Parameter of point to calculate
const double	**pnt1 [2]**	Input	First end point of line segment
const double	**pnt2 [2]**	Input	Second end point of line segment
double	**pnt_on_seg [2]**	Output	Point on line segment

UF_VEC2_copy (view source)

Defined in: uf_vec.h

Overview
Copies the vector coordinates from a source vector to a destination vector (vec_dst = vec_src).

Return
void.

Environment
Internal and External

Required License(s)
gateway

```
void UF_VEC2_copy
(
    const double vec_src [ 2 ] ,
    double vec_dst [ 2 ]
)
```

const double	**vec_src [2]**	Input	Source vector
double	**vec_dst [2]**	Output	Destination vector

UF_VEC2_cross (view source)

Defined in: uf_vec.h

Overview
Calculates the cross product of two vectors.

Return
void.

Environment

Internal and External

Required License(s)
gateway

void UF_VEC2_cross
(
const double vec1 [2] ,
const double vec2 [2] ,
double * cross_product
)

const double	**vec1 [2]**	Input	Vector 1
const double	**vec2 [2]**	Input	Vector 2
double *	**cross_product**	Output	The cross product of vec1 and vec2 cross_product = vec1 x vec2

UF_VEC2_distance (view source)

Defined in: uf_vec.h

Overview
Calculates the distance between two points.

Return
void.

Environment
Internal and External

Required License(s)
gateway

void UF_VEC2_distance
(
const double pnt1 [2] ,
const double pnt2 [2] ,
double * distance
)

const double	**pnt1 [2]**	Input	Point 1
const double	**pnt2 [2]**	Input	Point 2
double *	**distance**	Output	The distance between pnt1 and pnt2 (distance = ||pnt1 - pnt2||)

UF_VEC2_dot (view source)

Defined in: uf_vec.h

Overview
Calculates the dot product of vec1 and vec2.

Return
void.

Environment
Internal and External

Required License(s)
gateway

```
void UF_VEC2_dot
(
    const double vec1 [ 2 ] ,
    const double vec2 [ 2 ] ,
    double * dot_product
)
```

const double	**vec1 [2]**	Input	Vector 1
const double	**vec2 [2]**	Input	Vector 2
double *	**dot_product**	Output	The dot product of vec1 and vec2 dot_product = vec1 (dot) vec2

UF_VEC2_is_equal (view source)

Defined in: uf_vec.h

Overview
Determines if two vectors are equal within the specified tolerance.

Return
void.

Environment
Internal and External

Required License(s)
gateway

```
void UF_VEC2_is_equal
(
    const double vec1 [ 2 ] ,
    const double vec2 [ 2 ] ,
    double tolerance,
    int * is_equal
)
```

const double	**vec1 [2]**	Input	Vector 1
const double	**vec2 [2]**	Input	Vector 2
double	**tolerance**	Input	Tolerance value to use for checking
int *	**is_equal**	Output	0 = Vectors are not equal 1 = Vectors are equal

---

UF_VEC2_is_parallel (view source)

Defined in: uf_vec.h

Overview

Determine if vectors are parallel within an input tolerance. If the sine of the angle between vec1 and vec2 is 0 then TRUE is returned. Otherwise FALSE is returned.

Return

void.

Environment

Internal and External

Required License(s)

gateway

```
void UF_VEC2_is_parallel
(
    const double vec1 [ 2 ] ,
    const double vec2 [ 2 ] ,
    double tolerance,
    int * is_parallel
)
```

const double	**vec1 [2]**	Input	2D vector
const double	**vec2 [2]**	Input	2D vector
double	**tolerance**	Input	tolerance
int *	**is_parallel**	Output	= 0 Vectors are not parallel = 1 Vectors are parallel

---

UF_VEC2_is_perpendicular (view source)

Defined in: uf_vec.h

Overview

Determine if vectors are perpendicular within an input tolerance. If the cosine of the angle between vec1 and vec2 is 0 then TRUE is returned. Otherwise FALSE is returned.

Return
void.

Environment
Internal and External

Required License(s)
gateway

```
void UF_VEC2_is_perpendicular
(
    const double vec1 [ 2 ] ,
    const double vec2 [ 2 ] ,
    double tolerance,
    int * is_perp
)
```

const double	**vec1 [2]**	Input	2D vector
const double	**vec2 [2]**	Input	2D vector
double	**tolerance**	Input	tolerance
int *	**is_perp**	Output	= 0 Vectors are not perpendicular = 1 Vectors are perpendicular

---

UF_VEC2_is_zero (view source)

Defined in: uf_vec.h

Overview
Determines if a vector is zero within the specified tolerance.

Return
void.

Environment
Internal and External

Required License(s)
gateway

```
void UF_VEC2_is_zero
(
    const double vec [ 2 ] ,
    double tolerance,
    int * is_zero
)
```

const double	**vec [2]**	Input	Vector to test
double	**tolerance**	Input	Tolerance value to use for checking
int *	**is_zero**	Output	0 = Vectors is not zero 1 = Vectors is zero

---

UF_VEC2_linear_comb (view source)

Defined in: uf_vec.h

Overview
Calculates the vector linear combination of two vectors with the specified scale values.

Return
void.

Environment
Internal and External

Required License(s)
gateway

```
void UF_VEC2_linear_comb
(
    double scale1,
    const double vec1 [ 2 ] ,
    double scale2,
    const double vec2 [ 2 ] ,
    double vec_comb [ 2 ]
)
```

double	**scale1**	Input	Scale value for vector 1
const double	**vec1 [2]**	Input	Vector 1
double	**scale2**	Input	scale for vector 2
const double	**vec2 [2]**	Input	Vector 2
double	**vec_comb [2]**	Output	Vector linear combination vec_comb = (scale vec1) + (scale2 vec2)

UF_VEC2_mag (view source)

Defined in: uf_vec.h

Overview

Calculates the magnitude of a vector.

Return
void.

Environment
Internal and External

Required License(s)
gateway

```
void UF_VEC2_mag
(
    const double vec [ 2 ] ,
    double * magnitude
)
```

const double	**vec [2]**	Input	Vector whose magnitude is required
double *	**magnitude**	Output	Magnitude of vector magnitude = ||vec||

UF_VEC2_midpt (view source)

Defined in: uf_vec.h

Overview
Calculates the coordinates of the mid-point on a line segment.

Return
void.

Environment
Internal and External

Required License(s)
gateway

```
void UF_VEC2_midpt
(
    const double pnt1 [ 2 ] ,
    const double pnt2 [ 2 ] ,
    double mid_pnt [ 2 ]
)
```

const double	**pnt1 [2]**	Input	End Point #1 of line segment
const double	**pnt2 [2]**	Input	End Point #2 of line segment
double	**mid_pnt [2]**	Output	Mid-point of line segment mid_pnt = (0.5 pnt1) + (0.5 pnt2)

UF_VEC2_negate (view source)

Defined in: uf_vec.h

Overview
Calculates the negative of a vector.

Return
void.

Environment
Internal and External

Required License(s)
gateway

```
void UF_VEC2_negate
(
    const double vec [ 2 ] ,
    double negated_vec [ 2 ]
)
```

const double	**vec [2]**	Input	Vector to negate
double	**negated_vec [2]**	Output	Negated vector negated_vec = (-1.0) x vec

---

UF_VEC2_rotate (view source)

Defined in: uf_vec.h

Overview
Rotates a 2D vector about a line perpendicular to the plane of the vector through the vector origin.

Return
void.

Environment
Internal and External

Required License(s)
gateway

```
void UF_VEC2_rotate
(
    const double vec [ 2 ] ,
    double angle,
    double rotated_vec [ 2 ]
)
```

const double	**vec [2]**	Input	Vector to rotate

double	**angle**	Input	Angle to rotate through (in radians)
double	**rotated_vec [2]**	Output	Rotated vector

UF_VEC2_scale (view source)

Defined in: uf_vec.h

Overview
Scales the coordinates of a vector.

Return
void.

Environment
Internal and External

Required License(s)
gateway

```
void UF_VEC2_scale
(
    double scale,
    const double vec [ 2 ] ,
    double scaled_vec [ 2 ]
)
```

double	**scale**	Input	Scale factor
const double	**vec [2]**	Input	Vector to scale
double	**scaled_vec [2]**	Output	scaled vector scaled_vec = (scale x vec)

UF_VEC2_sub (view source)

Defined in: uf_vec.h

Overview
Subtracts one 2D vector from another.

Return
void.

Environment
Internal and External

Required License(s)
gateway

```
void UF_VEC2_sub
(
    const double vec1 [ 2 ] ,
    const double vec2 [ 2 ] ,
    double vec_diff [ 2 ]
)
```

const double	**vec1 [2]**	Input	Vector to subtract from
const double	**vec2 [2]**	Input	Vector to subtract
double	**vec_diff [2]**	Output	Vector difference vec_diff = vec1 - vec2

UF_VEC2_unitize (view source)

Defined in: uf_vec.h

Overview
Unitizes a 2D vector.

Return
0 = Success (unit vector can be calculated)
1 = input vector is zero

Environment
Internal and External

Required License(s)
gateway

```
int UF_VEC2_unitize
(
    const double vec [ 2 ] ,
    double tolerance,
    double * magnitude,
    double unit_vec [ 2 ]
)
```

const double	**vec [2]**	Input	Vector to unitize
double	**tolerance**	Input	Tolerance value to use for checking
double *	**magnitude**	Output	Vector magnitude = |vec|
double	**unit_vec [2]**	Output	Unitized vector = vec/|vec|

UF_VEC2_vec3 (view source)

Defined in: uf_vec.h

Overview
Converts a 2D vector to a 3D vector. Sets the Z-coordinate to zero.

Return
void.

Environment
Internal and External

Required License(s)
gateway

```
void UF_VEC2_vec3
(
    const double vec_2D [ 2 ] ,
    double vec_3D [ 3 ]
)
```

const double	**vec_2D [2]**	Input	2D vector to convert to 3D
double	**vec_3D [3]**	Output	3D vector

UF_VEC3_add (view source)

Defined in: uf_vec.h

Overview
Performs a three dimensional vector addition and returns the vector sum in vec_sum[3].

Return
void.

Environment
Internal and External

Required License(s)
gateway

```
void UF_VEC3_add
(
    const double vec1 [ 3 ] ,
    const double vec2 [ 3 ] ,
    double vec_sum [ 3 ]
)
```

const double	**vec1 [3]**	Input	The first three dimensional vector
const double	**vec2 [3]**	Input	The second three dimensional vector
double	**vec_sum [3]**	Output	The vector sum of vectors one and two vec_sum = vec1 + vec2

UF_VEC3_affine_comb (view source)

Defined in: uf_vec.h

Overview

Performs a vector affine combination which consists of adding an unscaled vector to a scaled vector. The first vector you input is vec[3], which is unscaled. The second vector you input is the vec_to_scale[3] which is scaled by the input argument scale. The resultant vector is output to vec_comb[3].

Return

void.

Environment

Internal and External

Required License(s)

gateway

```
void UF_VEC3_affine_comb
(
    const double vec [ 3 ] ,
    double scale,
    const double vec_to_scale [ 3 ] ,
    double vec_comb [ 3 ]
)
```

const double	**vec [3]**	Input	Unscaled vector
double	**scale**	Input	Scale to apply to vec_to_scale[3]
const double	**vec_to_scale [3]**	Input	The second three dimensional vector which is scaled.
double	**vec_comb [3]**	Output	Vector sum of unscaled vector and scaled vector where vec_comb = vec + (scale vec_to_scale)

UF_VEC3_angle_between (view source)

Defined in: uf_vec.h

Overview

Calculates the angle between two vectors using a third vector to determine the direction. The third vector is one that is perpendicular to both the vec_from and vec_to vectors. For example, if the two vectors lie in the x-y plane, then the third vector would be parallel to the z-axis.

Return

void.

Environment
Internal and External

Required License(s)
gateway

```
void UF_VEC3_angle_between
(
    const double vec_from [ 3 ] ,
    const double vec_to [ 3 ] ,
    const double vec_ccw [ 3 ] ,
    double * angle
)
```

const double	**vec_from [3]**	Input	Vector to calculate angle from
const double	**vec_to [3]**	Input	Vector to calculate angle to
const double	**vec_ccw [3]**	Input	Vector to define counter-clockwise orientation
double *	**angle**	Output	Angle between vec_from and vec_to using vec_ccw to determine positive orientation. The angle is in radians and 0.0 <= (angle) < 2 PI.

UF_VEC3_ask_perpendicular (view source)

Defined in: uf_vec.h

Overview
Returns a 3D vector that is perpendicular to the input vector

Return
void.

Environment
Internal and External

Required License(s)
gateway

```
void UF_VEC3_ask_perpendicular
(
    const double vec1 [ 3 ] ,
    double vec_perp [ 3 ]
)
```

const double	**vec1 [3]**	Input	3D vector
double	**vec_perp [3]**	Output	3D vector perpendicular to the first vector.

UF_VEC3_convex_comb (view source)

Defined in: uf_vec.h

Overview
Calculates the position of a point between the two end points of a line segment. The point on the line segment is defined by:
pnt_on_seg = (parameter pnt1) + ((1.0 - parameter) pnt2).

Return
void.

Environment
Internal and External

Required License(s)
gateway

```
void UF_VEC3_convex_comb
(
    double parameter,
    const double pnt1 [ 3 ] ,
    const double pnt2 [ 3 ] ,
    double pnt_on_seg [ 3 ]
)
```

double	**parameter**	Input	Parameter of point to calculate
const double	**pnt1 [3]**	Input	First end point of line segment
const double	**pnt2 [3]**	Input	Second end point of line segment
double	**pnt_on_seg [3]**	Output	Point on line segment

UF_VEC3_copy (view source)

Defined in: uf_vec.h

Overview
Copies the vector coordinates from a source vector to a destination vector (vec_dst = vec_src).

Return
void.

Environment
Internal and External

Required License(s)
gateway

```
void UF_VEC3_copy
(
    const double vec_src [ 3 ] ,
    double vec_dst [ 3 ]
)
```

const double	**vec_src [3]**	Input	Source vector
double	**vec_dst [3]**	Output	Destination vector

UF_VEC3_cross (view source)

Defined in: uf_vec.h

Overview
Calculates the cross product of two vectors.

Return
void.

Environment
Internal and External

Required License(s)
gateway

```
void UF_VEC3_cross
(
    const double vec1 [ 3 ] ,
    const double vec2 [ 3 ] ,
    double cross_product [ 3 ]
)
```

const double	**vec1 [3]**	Input	Vector 1
const double	**vec2 [3]**	Input	Vector 2
double	**cross_product [3]**	Output	The cross product of vec1 and vec2 cross_product = vec1 vec2

UF_VEC3_distance (view source)

Defined in: uf_vec.h

Overview
Calculates the distance between two points.

Return
void.

Environment

Internal and External

Required License(s)
gateway

```
void UF_VEC3_distance
(
    const double pnt1 [ 3 ] ,
    const double pnt2 [ 3 ] ,
    double * distance
)
```

const double	**pnt1 [3]**	Input	Point 1
const double	**pnt2 [3]**	Input	Point 2
double *	**distance**	Output	The distance between pnt1 and pnt2 (distance = ||pnt1 - pnt2||)

UF_VEC3_distance_to_plane (view source)

Defined in: uf_vec.h

Overview
Calculates the normal distance from a point to a plane.

Return
0 = Success (distance can be calculated)
1 = The plane normal is zero

Environment
Internal and External

Required License(s)
gateway

```
int UF_VEC3_distance_to_plane
(
    const double pnt1 [ 3 ] ,
    const double pnt_on_plane [ 3 ] ,
    const double plane_normal [ 3 ] ,
    double tolerance,
    double * distance
)
```

const double	**pnt1 [3]**	Input	Point to calculate distance from
const double	**pnt_on_plane [3]**	Input	Point located on the plane
const double	**plane_normal [3]**	Input	Plane normal
double	**tolerance**	Input	Tolerance value to use for checking

double *	**distance**	Output	The normal distance from the point to the plane

---

UF_VEC3_dot (view source)

Defined in: uf_vec.h

Overview
Calculates the dot product of vec1 and vec2.

Return
void.

Environment
Internal and External

Required License(s)
gateway

```
void UF_VEC3_dot
(
    const double vec1 [ 3 ] ,
    const double vec2 [ 3 ] ,
    double * dot_product
)
```

const double	**vec1 [3]**	Input	Vector 1
const double	**vec2 [3]**	Input	Vector 2
double *	**dot_product**	Output	The dot product of vec1 and vec2 dot_product = vec1 (dot) vec2

UF_VEC3_is_equal (view source)

Defined in: uf_vec.h

Overview
Determines if two vectors are equal within the specified tolerance.

Return
void.

Environment
Internal and External

Required License(s)
gateway

```
void UF_VEC3_is_equal
(
    const double vec1 [ 3 ] ,
    const double vec2 [ 3 ] ,
    double tolerance,
    int * is_equal
)
```

const double	**vec1 [3]**	Input	Vector 1
const double	**vec2 [3]**	Input	Vector 2
double	**tolerance**	Input	Tolerance value to use for checking
int *	**is_equal**	Output	0 = Vectors are not equal 1 = Vectors are equal

---

UF_VEC3_is_parallel (view source)

Defined in: uf_vec.h

Overview

Determine if vectors are parallel within an input tolerance. If the sine of the angle between the vec1 and vec2 is less than tolerance then TRUE is returned. Otherwise a FALSE will be returned. To get an angle tolerance of x degrees the expected tolerance would be sin(xDEGRA).

Return

void.

Environment

Internal and External

Required License(s)

gateway

```
void UF_VEC3_is_parallel
(
    const double vec1 [ 3 ] ,
    const double vec2 [ 3 ] ,
    double tolerance,
    int * is_parallel
)
```

const double	**vec1 [3]**	Input	3D vector
const double	**vec2 [3]**	Input	3D vector
double	**tolerance**	Input	tolerance
int *	**is_parallel**	Output	= 0 Vectors are not parallel = 1 Vectors are parallel

---

UF_VEC3_is_perpendicular (view source)

Defined in: uf_vec.h

Overview
Determine if vectors are perpendicular an input tolerance. If the cosine of the angle between vec1 and vec2 is less than the tolerance, then a TRUE is returned. Otherwise FALSE is returned. To check perpendicularity with x degrees from 90, the expected tolerance would be cos((90-x)DEGRA).

Return
void.

Environment
Internal and External

Required License(s)
gateway

```
void UF_VEC3_is_perpendicular
(
    const double vec1 [ 3 ] ,
    const double vec2 [ 3 ] ,
    double tolerance,
    int * is_perp
)
```

const double	**vec1 [3]**	Input	3D vector
const double	**vec2 [3]**	Input	3D vector
double	**tolerance**	Input	tolerance
int *	**is_perp**	Output	= 0 Vectors are not perpendicular = 1 Vectors are perpendicular

---

UF_VEC3_is_zero (view source)

Defined in: uf_vec.h

Overview
Determines if a vector is zero within the specified tolerance.

Return
void.

Environment
Internal and External

Required License(s)
gateway

```
void UF_VEC3_is_zero
(
    const double vec [ 3 ] ,
    double tolerance,
    int * is_zero
)
```

const double	**vec [3]**	Input	Vector to test
double	**tolerance**	Input	Tolerance value to use for checking
int *	**is_zero**	Output	0 = Vectors is not zero 1 = Vectors is zero

---

UF_VEC3_linear_comb (view source)

Defined in: uf_vec.h

Overview
Calculates the vector linear combination of two vectors with the specified scale values.

Return
void.

Environment
Internal and External

Required License(s)
gateway

```
void UF_VEC3_linear_comb
(
    double scale1,
    const double vec1 [ 3 ] ,
    double scale2,
    const double vec2 [ 3 ] ,
    double vec_comb [ 3 ]
)
```

double	**scale1**	Input	Scale value for vector 1
const double	**vec1 [3]**	Input	Vector 1
double	**scale2**	Input	scale for vector 2
const double	**vec2 [3]**	Input	Vector 2
double	**vec_comb [3]**	Output	Vector linear combination vec_comb = (scale vec1) + (scale2 vec2)

UF_VEC3_mag (view source)

Defined in: uf_vec.h

Overview
Calculates the magnitude of a vector.

Return
void.

Environment
Internal and External

Required License(s)
gateway

```
void UF_VEC3_mag
(
    const double vec [ 3 ] ,
    double * magnitude
)
```

const double	**vec [3]**	Input	Vector whose magnitude is required
double *	**magnitude**	Output	Magnitude of vector magnitude = ||vec||

UF_VEC3_midpt (view source)

Defined in: uf_vec.h

Overview
Calculates the coordinates of the mid-point on a line segment.

Return
void.

Environment
Internal and External

Required License(s)
gateway

```
void UF_VEC3_midpt
(
    const double pnt1 [ 3 ] ,
    const double pnt2 [ 3 ] ,
    double mid_pnt [ 3 ]
)
```

const double	**pnt1 [3]**	Input	End Point #1 of line segment
const double	**pnt2 [3]**	Input	End Point #2 of line segment

double	**mid_pnt [3]**	Output	Mid-point of line segment mid_pnt = (0.5 pnt1) + (0.5 pnt2)

UF_VEC3_negate (view source)

Defined in: uf_vec.h

Overview
Calculates the negative of a vector.

Return
void.

Environment
Internal and External

Required License(s)
gateway

```
void UF_VEC3_negate
(
    const double vec [ 3 ] ,
    double negated_vec [ 3 ]
)
```

const double	**vec [3]**	Input	Vector to negate
double	**negated_vec [3]**	Output	Negated vector negated_vec = (-1.0) vec

---

UF_VEC3_scale (view source)

Defined in: uf_vec.h

Overview
Scales the coordinates of a vector.

Return
void.

Environment
Internal and External

Required License(s)
gateway

```
void UF_VEC3_scale
(
    double scale,
    const double vec [ 3 ] ,
```

```
    double scaled_vec [ 3 ]
)
```

double	**scale**	Input	Scale factor
const double	**vec [3]**	Input	Vector to scale
double	**scaled_vec [3]**	Output	scaled vector scaled_vec = (scale x vec)

UF_VEC3_sub (view source)

Defined in: uf_vec.h

Overview
Subtracts one vector from another.

Return
void.

Environment
Internal and External

Required License(s)
gateway

```
void UF_VEC3_sub
(
    const double vec1 [ 3 ] ,
    const double vec2 [ 3 ] ,
    double vec_diff [ 3 ]
)
```

const double	**vec1 [3]**	Input	Vector to subtract from
const double	**vec2 [3]**	Input	Vector to subtract
double	**vec_diff [3]**	Output	Vector difference vec_diff = vec1 - vec2

UF_VEC3_triple (view source)

Defined in: uf_vec.h

Overview
Calculates the triple scalar product of three vectors.

Return
void.

Environment

Internal and External

Required License(s)
gateway

void UF_VEC3_triple
(
const double vec1 [3] ,
const double vec2 [3] ,
const double vec3 [3] ,
double * triple_product
)

const double	**vec1 [3]**	Input	Vector #1
const double	**vec2 [3]**	Input	Vector #2
const double	**vec3 [3]**	Input	Vector #3
double *	**triple_product**	Output	The triple scalar product triple_product = vec1 (dot) (vec2 x vec3)

UF_VEC3_unitize (view source)

Defined in: uf_vec.h

Overview
Unitizes a vector.

Return
0 = Success (unit vector can be calculated)
1 = input vector is zero

Environment
Internal and External

Required License(s)
gateway

int UF_VEC3_unitize
(
const double vec [3] ,
double tolerance,
double * magnitude,
double unit_vec [3]
)

const double	**vec [3]**	Input	Vector to unitize
double	**tolerance**	Input	Tolerance value to use for checking
double *	**magnitude**	Output	Vector magnitude = ||vec||

double	**unit_vec [3]**	Output	Unitized vector = vec/||vec||

UF_VEC3_vec2 (view source)

Defined in: uf_vec.h

Overview
Converts a 3D vector to a 2D vector. Strips the Z-coordinate of the 3D vector.

Return
void.

Environment
Internal and External

Required License(s)
gateway

```
void UF_VEC3_vec2
(
    const double vec_3D [ 3 ] ,
    double vec_2D [ 2 ]
)
```

const double	**vec_3D [3]**	Input	3D vector to convert to 2D
double	**vec_2D [2]**	Output	2D vector

UF_VEC3_vec4 (view source)

Defined in: uf_vec.h

Overview
Converts a 3D vector to a 4D homogeneous vector with a weight of 1.0.

Return
void.

Environment
Internal and External

Required License(s)
gateway

```
void UF_VEC3_vec4
(
    const double vec_3D [ 3 ] ,
    double vec_4D [ 4 ]
```

```
)
```

const double	**vec_3D [3]**	Input	3D vector to convert to 4D
double	**vec_4D [4]**	Output	4D vector

UF_VEC3_vec4_homogen (view source)

Defined in: uf_vec.h

Overview
Converts a 3D vector to a 4D homogeneous vector with the given weight. The 3D coordinates are multiplied by the specified weight. If the 3D coordinates are (x,y,z) and the weight = h, then the 4D coordinates would be (hx,hy,hz,h).

Return
void.

Environment
Internal and External

Required License(s)
gateway

```
void UF_VEC3_vec4_homogen
(
    const double vec_3D [ 3 ] ,
    double weight,
    double vec_4D [ 4 ]
)
```

const double	**vec_3D [3]**	Input	3D vector to convert
double	**weight**	Input	Weight to be used
double	**vec_4D [4]**	Output	4D homogeneous vector

UF_VEC4_copy (view source)

Defined in: uf_vec.h

Overview
Copies the vector coordinates from a source vector to a destination vector (vec_dst = vec_src).

Return
void.

Environment

Internal and External

Required License(s)
gateway

```
void UF_VEC4_copy
(
    const double vec_src [ 4 ] ,
    double vec_dst [ 4 ]
)
```

const double	**vec_src [4]**	Input	Source vector
double	**vec_dst [4]**	Output	Destination vector

UF_VEC4_is_equal (view source)

Defined in: uf_vec.h

Overview
Determines if two vectors are equal within the specified tolerance.

Return
void.

Environment
Internal and External

Required License(s)
gateway

```
void UF_VEC4_is_equal
(
    const double vec1 [ 4 ] ,
    const double vec2 [ 4 ] ,
    double tolerance,
    int * is_equal
)
```

const double	**vec1 [4]**	Input	Vector #1
const double	**vec2 [4]**	Input	Vector #2
double	**tolerance**	Input	Tolerance value to use for checking
int *	**is_equal**	Output	0 = Vectors are not equal 1 = Vectors are equal

---

UF_VEC4_is_zero (view source)

Defined in: uf_vec.h

Overview
Determines if a vector is zero within the specified tolerance.

Return
void.

Environment
Internal and External

Required License(s)
gateway

```
void UF_VEC4_is_zero
(
    const double vec [ 4 ] ,
    double tolerance,
    int * is_zero
)
```

const double	**vec [4]**	Input	Vector to test
double	**tolerance**	Input	Tolerance value to use for checking
int *	**is_zero**	Output	0 = Vector is not zero 1 = Vector is zero

---

UF_VEC4_scale (view source)

Defined in: uf_vec.h

Overview
Scales the coordinates of a vector.

Return
void.

Environment
Internal and External

Required License(s)
gateway

```
void UF_VEC4_scale
(
    double scale,
    const double vec [ 4 ] ,
    double scaled_vec [ 4 ]
)
```

double	**scale**	Input	Scale factor
const double	**vec [4]**	Input	Vector to scale
double	**scaled_vec [4]**	Output	scaled vector scaled_vec = (scale x vec)

UF_VEC4_vec3 (view source)

Defined in: uf_vec.h

Overview
Converts a 4D vector to a 3D vector by stripping the weight of the 4D vector.

Return
void.

Environment
Internal and External

Required License(s)
gateway

```
void UF_VEC4_vec3
(
    const double vec_4D [ 4 ] ,
    double vec_3D [ 3 ]
)
```

const double	**vec_4D [4]**	Input	4D vector to convert to 3D
double	**vec_3D [3]**	Output	3D vector

UF_VEC4_vec3_homogen (view source)

Defined in: uf_vec.h

Overview
Converts a 4D homogeneous vector to a 3D vector by dividing the 4D coordinates by the weight.

Return
void.

Environment
Internal and External

Required License(s)
gateway

```
void UF_VEC4_vec3_homogen
(
    const double vec_4D [ 4 ] ,
    double vec_3D [ 3 ]
)
```

const double	**vec_4D [4]**	Input	4D vector to convert
double	**vec_3D [3]**	Output	3D vector
