



QualNet 6.1

API Reference Guide

September 2012



Scalable Network Technologies, Inc.

6100 Center Drive, Suite 1250
Los Angeles, CA 90045

Phone: 310-338-3318
Fax: 310-338-7213

<http://www.scalable-networks.com>

Copyright Information

© 2012 Scalable Network Technologies, Inc. All rights reserved.

QualNet and EXata are registered trademarks of Scalable Network Technologies, Inc.

All other trademarks and trade names used are property of their respective companies.

Scalable Network Technologies, Inc.
6100 Center Drive, Suite 1250
Los Angeles, CA 90045
Phone: 310-338-3318
Fax: 310-338-7213

<http://www.scalable-networks.com>

QualNet 6.1 API Reference

QualNet API

[3D_MATH](#)

This file describes data structures and functions used to model 3D weather patterns in conjunction with the Weather package.

[ANTENNA](#)

This file describes data structures and functions used by antenna models.

[ANTENNA_GLOBAL](#)

This file describes additional data structures and functions used by antenna models.

[API](#)

This file enumerates the basic message/events exchanged during the simulation process and the various layer functions (initialize, finalize, and event handling functions) and other miscellaneous routines and data structure definitions.

[APP_UTIL](#)

This file describes Application Layer utility functions.

[APPLICATION_LAYER](#)

This file describes data structures and functions used by the Application Layer.

[BUFFER](#)

This file describes data structures and functions to implement buffers.

[CIRCULAR-BUFFER](#)

This file describes data structures and functions used for circular buffer implementation.

[CLOCK](#)

This file describes data structures and functions used for time-related operations.

[COORDINATES](#)

This file describes data structures and functions used for coordinates-related operations.

[ERROR](#)

This file defines data structures and functions used in error-handling.

[EXTERNAL](#)

This file defines the generic interface to external modules.

[EXTERNAL_SOCKET](#)

This file describes utilities for managing socket connections to external programs.

[EXTERNAL UTILITIES](#)

This file describes utilities for external interfaces.

[FILEIO](#)

This file describes data structures and functions used for reading from input files and printing to output files.

[GUI](#)

This file describes data structures and functions for interfacing with the QualNet GUI and the other graphical tools.

[IP](#)

This file contains data structures and prototypes of functions used by IP.

[IPv6](#)

Data structures and parameters used in network layer are defined here.

[LIST](#)

This file describes the data structures and functions used in the implementation of lists.

[MAC_LAYER](#)

This file describes data structures and functions used by the MAC Layer.

[MAIN](#)

This file contains some common definitions.

[MAPPING](#)

This file describes data structures and functions for mapping between node pointers, node identifiers, and node addresses.

[MEMORY](#)

This file describes the memory management data structures and functions.

[MESSAGE](#)

This file describes the message structure used to implement events and functions for message operations.

[MOBILITY](#)

This file describes data structures and functions used by mobility models.

[MUTEX](#)

This file describes objects for use in creating critical regions (synchronized access) for global variables or data structures that have to be shared between threads.

[NETWORK_LAYER](#)

This file describes the data structures and functions used by the Network Layer.

[NODE](#)

This file defines the Node data structure and some generic operations on nodes.

[PARALLEL](#)

This file describes data structures and functions used for parallel programming.

[PARTITION](#)

This file contains declarations of some functions for partition threads.

[PHYSICAL_LAYER](#)

This file describes data structures and functions used by the Physical Layer. Most of this functionality is enabled/used in the Wireless library.

[PROPAGATION](#)

This file describes data structures and functions used by propagation models.

[QUEUES](#)

This file describes the member functions of the queue base class.

[RANDOM_NUMBERS](#)

This file describes functions to generate pseudo-random number streams.

[SCHEDULERS](#)

This file describes the member functions of the scheduler base class.

[SLIDING-WINDOW](#)

This file describes data structures and functions to implement a sliding window.

[TRACE](#)

This file describes data structures and functions used for packet tracing.

[TRANSPORT_LAYER](#)

This file describes data structures and functions used by the Transport Layer.

USER

This file describes data structures and functions used by the User Layer.

WALLCLOCK

This file describes methods of the WallClock class whose primary use is to keep track of the amount of real time that has passed during the simulation.

QualNet 6.1 API Reference

3D_MATH

This file describes data structures and functions used to model 3D weather patterns in conjunction with the Weather package.

Constant / Data Structure Summary

Type	Name
STRUCT	Vector3 This is used to hold 3D points and vectors. This will eventually be added upon to create a robust class with operator overloading. For now we just need an x, y, z.
STRUCT	Triangle3 This struture will hold information for one triangle.

Function / Macro Summary

Return Type	Summary
Vector3	MATH_CrossProduct (Vector3 vector1, Vector3 vector2) Returns a perpendicular vector from 2 given vectors by taking the cross product.
Vector3	MATH_Vector (Vector3 point1, Vector3 point2) Returns a vector between 2 points
double	MATH_Magnitude (Vector3 vector) Returns the magnitude of a normal (or any other vector)
Vector3	MATH_Normalize (Vector3 vector) Returns a normalized vector (of exactly length 1)
Vector3	MATH_Normal (Vector3[] triangle)

	Returns the direction the polygon is facing MATH_PlaneDistance (Vector3 vector, Vector3 point)
double	Returns the distance the plane is from the origin (0, 0, 0). It takes the normal to the plane, along with ANY point that lies on the plane (any corner) MATH_IntersectedPlane (Vector3[] polygon, Vector3[] line, Vector3& normal, double& originDistance)
BOOL	Takes a triangle (plane) and line and returns true if they intersected
double	MATH_DotProduct (Vector3 vector1, Vector3 vector2) Returns the dot product between 2 vectors.
double	MATH_AngleBetweenVectors (Vector3 vector1, Vector3 vector2) This returns the angle between 2 vectors
Vector3	MATH_IntersectionPoint (Vector3 normal, Vector3[] line, double distance) Returns an intersection point of a polygon and a line (assuming intersects the plane)
BOOL	MATH_InsidePolygon (Vector3 intersection, Vector3[] polygon, int verticeCount) Returns true if the intersection point is inside of the polygon
BOOL	MATH_IntersectedPolygon (Vector3[] polygon, Vector3[] line, int verticeCount) Tests collision between a line and polygon
double	MATH_Distance (Vector3 point1, Vector3 point2) Returns the distance between 2 3D points
BOOL	MATH_LineIntersects (Vector3[] line1, Vector3[] line2) Checks whether two lines intersect each other or not.
Vector3	MATH_ReturnLineToLineIntersectionPoint (Vector3[] line1, Vector3[] line2) Returns the point of intersection between two lines.
BOOL	MATH_IsPointOnLine (Vector3 point, Vector3[] line)

	Returns the whether the given point lies on Line or not.
void	<p>MATH_ConvertXYToLatLong(double x1, double y1, double latitude, double longitude)</p> <p>Converts given cartesian coordinates to Latitide and Longitude</p>

Constant / Data Structure Detail

Structure	Vector3
	<p>This is used to hold 3D points and vectors. This will eventually be added upon to create a robust class with operator overloading. For now we just need an x, y, z.</p>
Structure	<p>Triangle3</p> <p>This struture will hold information for one triangle.</p>

Function / Macro Detail

Function / Macro	Format
MATH_CrossProduct	<p>Vector3 MATH_CrossProduct (Vector3 vector1, Vector3 vector2)</p> <p>Parameters:</p> <ul style="list-style-type: none"> vector1 - the first vector vector2 - the second vector <p>Returns:</p> <ul style="list-style-type: none"> Vector3 - the cross product
MATH_Vector	<p>Vector3 MATH_Vector (Vector3 point1, Vector3 point2)</p> <p>Parameters:</p> <ul style="list-style-type: none"> point1 - the first point point2 - the second point <p>Returns:</p> <ul style="list-style-type: none"> Vector3 - a vector between the two points

MATH_Magnitude	<p>Returns the magnitude of a normal (or any other vector)</p> <p>double MATH_Magnitude (Vector3 vector)</p> <p>Parameters:</p> <ul style="list-style-type: none"> vector - a vector <p>Returns:</p> <ul style="list-style-type: none"> double - the magnitude of the vector
MATH_Normalize	<p>Returns a normalized vector (of exactly length 1)</p> <p>Vector3 MATH_Normalize (Vector3 vector)</p> <p>Parameters:</p> <ul style="list-style-type: none"> vector - a vector <p>Returns:</p> <ul style="list-style-type: none"> Vector3 - a normalized vector
MATH_Normal	<p>Returns the direction the polygon is facing</p> <p>Vector3 MATH_Normal (Vector3[] triangle)</p> <p>Parameters:</p> <ul style="list-style-type: none"> triangle - an array of vectors representing a polygon <p>Returns:</p> <ul style="list-style-type: none"> Vector3 - the direction vector
MATH_PlaneDistance	<p>Returns the distance the plane is from the origin (0, 0, 0). It takes the normal to the plane, along with ANY point that lies on the plane (any corner)</p> <p>double MATH_PlaneDistance (Vector3 vector, Vector3 point)</p> <p>Parameters:</p> <ul style="list-style-type: none"> vector - a vector point - a point <p>Returns:</p> <ul style="list-style-type: none"> double - the plane's distance from the origin (0,0,0)
MATH_IntersectedPlane	<p>Takes a triangle (plane) and line and returns true if they intersected</p> <p>BOOL MATH_IntersectedPlane (Vector3[] polygon, Vector3[] line, Vector3& normal, double& originDistance)</p> <p>Parameters:</p> <ul style="list-style-type: none"> polygon - a polygon line - a line normal - a normalized vector originDistance - the distance

	<p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - True if they intersect
MATH_DotProduct	<p>double MATH_DotProduct (Vector3 vector1, Vector3 vector2)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>vector1</code> - the first vector • <code>vector2</code> - the second vector <p>Returns:</p> <ul style="list-style-type: none"> • <code>double</code> - the dot product of the two vectors
MATH_AngleBetweenVectors	<p>This returns the angle between 2 vectors</p> <p>double MATH_AngleBetweenVectors (Vector3 vector1, Vector3 vector2)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>vector1</code> - the first vector • <code>vector2</code> - the second vector <p>Returns:</p> <ul style="list-style-type: none"> • <code>double</code> - None
MATH_IntersectionPoint	<p>Returns an intersection point of a polygon and a line (assuming intersects the plane)</p> <p>Vector3 MATH_IntersectionPoint (Vector3 normal, Vector3[] line, double distance)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>normal</code> - a polygon • <code>line</code> - a line • <code>distance</code> - the distance between? <p>Returns:</p> <ul style="list-style-type: none"> • <code>Vector3</code> - None
MATH_InsidePolygon	<p>Returns true if the intersection point is inside of the polygon</p> <p>BOOL MATH_InsidePolygon (Vector3 intersection, Vector3[] polygon, int verticeCount)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>intersection</code> - an intersection point • <code>polygon</code> - a polygon • <code>verticeCount</code> - number of points in polygon <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - True if the intersection point is in the polygon

MATH_IntersectedPolygon Tests collision between a line and polygon	<p>BOOL MATH_IntersectedPolygon (Vector3[] polygon, Vector3[] line, int verticeCount)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • polygon - a polygon • line - a line • verticeCount - number of points in polygon <p>Returns:</p> <ul style="list-style-type: none"> • BOOL - True if the polygon and line intersect
MATH_Distance Returns the distance between 2 3D points	<p>double MATH_Distance (Vector3 point1, Vector3 point2)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • point1 - the first point • point2 - the second point <p>Returns:</p> <ul style="list-style-type: none"> • double - the distance between the two points
MATH_LineIntersects Checks whether two lines intersect each other or not.	<p>BOOL MATH_LineIntersects (Vector3[] line1, Vector3[] line2)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • line1 - the first line • line2 - the second line <p>Returns:</p> <ul style="list-style-type: none"> • BOOL - True if the lines intersect
MATH_ReturnLineToLineIntersectionPoint Returns the point of intersection between two lines.	<p>Vector3 MATH_ReturnLineToLineIntersectionPoint (Vector3[] line1, Vector3[] line2)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • line1 - the first line • line2 - the second line <p>Returns:</p> <ul style="list-style-type: none"> • Vector3 - the intersection point
MATH_IsPointOnLine	<p>BOOL MATH_IsPointOnLine (Vector3 point, Vector3[] line)</p> <p>Parameters:</p>

<p>Returns the whether the given point lies on Line or not.</p>	<ul style="list-style-type: none"> • <code>point</code> - the point which we are checking. • <code>line</code> - the line on which the point might lie. <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - TRUE if the point lies on line
<p>MATH_ConvertXYToLatLong</p> <p>Converts given cartesian coordinates to Latitide and Longitude</p>	<pre>void MATH_ConvertXYToLatLong (double x1, double y1, double latitude, double longitude)</pre> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>x1</code> - Specifies X value on X-Axis • <code>y1</code> - Specifies Y value on Y-Axis • <code>latitude</code> - Will store the converted latitude value • <code>longitude</code> - Will store the converted longitude value <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL



Contact [QualNet Support](#) for questions pertaining to the QualNet API Reference. This document is confidential and proprietary.
It may not be reproduced or distributed without the expressed written consent of [Scalable Network Technologies](#).

QualNet® is a Registered Trademark of [Scalable Network Technologies](#).

Copyright © 2001-2012 [Scalable Network Technologies, Inc.](#). All rights reserved.

QualNet 6.1 API Reference

ANTENNA

This file describes data structures and functions used by antenna models.

Constant / Data Structure Summary

Type	Name
CONSTANT	ANTENNA_DEFAULT_HEIGHT
	Default height of the antenna
CONSTANT	ANTENNA_DEFAULT_GAIN_dBi
	Default gain of the antenna
CONSTANT	ANTENNA_DEFAULT_EFFICIENCY
	Default efficiency of the antenna
CONSTANT	ANTENNA_DEFAULT_MISMATCH_LOSS_db
	Default mismatch loss of the antenna
CONSTANT	ANTENNA_DEFAULT_CONNECTION_LOSS_db
	Default connection loss of the antenna
CONSTANT	ANTENNA_DEFAULT_CABLE_LOSS_db
	Default cable loss of the antenna
CONSTANT	ANTENNA_LOWEST_GAIN_dBi
	Default minimum gain of the antenna
CONSTANT	ANTENNA_DEFAULT_PATTERN
	Default Pattern
CONSTANT	ANTENNA_OMNIDIRECTIONAL_PATTERN

	OMNIDIRECTIONAL PATTERN
CONSTANT	ANTENNA_PATTERN_NOT_SET
	Const for Pattern of antenna not set
CONSTANT	AZIMUTH_INDEX
	Const for azimuth index of antenna Pattern
CONSTANT	ELEVATION_INDEX
	Const for elevation index of antenna Pattern
CONSTANT	MAX_ANTENNA_NUM_LINES
	Const for the line number in the antennaModelInput
CONSTANT	AZIMUTH_ELEVATION_INDEX
	Const for the memory allocation of azimuth and elevation gain array.
CONSTANT	NSMA_PATTERN_START_LINE_NUMBER
	Const represents the basic pattern starting point in NSMA file
CONSTANT	NSMA_MAX_STARTLINE
	Const represents the Revised pattern max line number where the revised NSMA pattern can start.

Function / Macro Summary

Return Type	Summary
void	<p>ANTENNA_Init(Node* node, int phyIndex, const NodeInput* nodeInput)</p> <p>Initialize antennas.</p>
void	<p>ANTENNA_ReadPatterns(Node* node, int phyIndex, const NodeInput* antennaInput, int* numPatterns, int* steerablePatternSetRepeatSectorAngle, float*** pattern_dB, BOOL azimuthPlane)</p>

ANTENNA

	Read in the azimuth pattern file. ANTENNA_ReadNsmaPatterns (Node* node, int phyIndex)
void	Read in the NSMA pattern file. ANTENNA_ReadRevisedNsmaPatterns (Node* node, int phyIndex)
void	Read in the Revised NSMA pattern file. ANTENNA_Read3DAsciiPatterns (Node* node, int phyIndex)
void	Used to read ASCII 3D pattern file. ANTENNA_Read2DAsciiPatterns (Node* node, int phyIndex)
void	Used to read ASCII 2D pattern file. ANTENNA_OmniDirectionalInit (Node* node, const NodeInput* nodeInput, int phyIndex, const AntennaModelGlobal* antennaModel)
void	Initialize omnidirectional antenna from the antenna model file. ANTENNA_OmniDirectionalInitFromConfigFile (Node* node, int phyIndex, const NodeInput* nodeInput)
void	Initialize omnidirectional antenna from the default.config file. ANTENNA_InitFromConfigFile (Node* node, int phyIndex, const NodeInput* nodeInput)
BOOL	Initialize antenna from the default.config file. ANTENNA_IsInOmnidirectionalMode (Node* node, int phyIndex)
int	Is antenna in omnidirectional mode. ANTENNA_ReturnPatternIndex (Node* node, int phyIndex)
float	Return nodes current pattern index. ANTENNA_ReturnHeight (Node* node, int phyIndex)
double	Return nodes antenna height. ANTENNA_ReturnSystemLossIndB (Node* node, int phyIndex)

	<p>Return system loss in dB.</p>
float	ANTENNA_GainForThisDirection (Node* node, int phyIndex, Orientation DOA)
	<p>Return gain for this direction in dB.</p>
float	ANTENNA_GainForThisDirectionWithPatternIndex (Node* node, int phyIndex, int patternIndex, Orientation DOA)
	<p>Return gain for this direction for the specified pattern in dB.</p>
float	ANTENNA_GainForThisSignal (Node* node, int phyIndex, PropRxInfo* propRxInfo)
	<p>Return gain in dB.</p>
float	ANTENNA_DefaultGainForThisSignal (Node* node, int phyIndex, PropRxInfo* propRxInfo)
	<p>Return default gain in dB.</p>
void	ANTENNA_LockAntennaDirection (Node* node, int phyIndex)
	<p>Lock antenna to current direction.</p>
void	ANTENNA_UnlockAntennaDirection (Node* node, int phyIndex)
	<p>Unlock antenna.</p>
BOOL	ANTENNA_DirectionIsLocked (Node* node, int phyIndex)
	<p>Return if direction antenna is locked.</p>
BOOL	ANTENNA_IsLocked (Node* node, int phyIndex)
	<p>Return if antenna is locked.</p>
void	ANTENNA_SetToDefaultMode (Node* node, int phyIndex)
	<p>Set default antenna mode (usually omni).</p>
void	ANTENNA_SetToBestGainConfigurationForThisSignal (Node* node, int phyIndex, PropRxInfo* propRxInfo)
	<p>Set antenna for best gain using the Rx info.</p>
void	ANTENNA_SetBestConfigurationForAzimuth (Node* node, int phyIndex, double azimuth)
	<p>Set antenna for best gain using the azimuth.</p>

void	ANTENNA_GetSteeringAngle (Node* node, int phyIndex, Orientation* angle)
	Get steering angle of the antenna.
void	ANTENNA_SetSteeringAngle (Node* node, int phyIndex, Orientation angle)
	Set the steering angle of the antenna
void	ANTENNA_ReturnAsciiPatternFile (Node* node, int phyIndex, const NodeInput* antennaModelInput)
	Read in the ASCII pattern .
void	ANTENNA_ReturnNsmaPatternFile (Node* node, int phyIndex, const NodeInput* antennaModelInput, AntennaPatterns* antennaPatterns)
	Read in the NSMA pattern .
void	ANTENNA_ReturnTraditionalPatternFile (Node* node, int phyIndex, const NodeInput* antennaModelInput)
	Used to read Qualnet Traditional pattern file
NodeInput *	ANTENNA_MakeAntennaModelInput (Node* node, char* buf)
	Reads the antenna configuration parameters into the NodeInput structure.

Constant / Data Structure Detail

Constant	
Constant	ANTENNA_DEFAULT_HEIGHT 1.5 Default height of the antenna
Constant	ANTENNA_DEFAULT_GAIN_dBi 0.0 Default gain of the antenna
Constant	ANTENNA_DEFAULT_EFFICIENCY 0.8 Default efficiency of the antenna
Constant	ANTENNA_DEFAULT_MISMATCH_LOSS_dB 0.3

	Default mismatch loss of the antenna
Constant	ANTENNA_DEFAULT_CONNECTION_LOSS_dB 0.2 Default connection loss of the antenna
Constant	ANTENNA_DEFAULT_CABLE LOSS_dB 0.0 Default cable loss of the antenna
Constant	ANTENNA_LOWEST_GAIN_dBi -10000.0 Default minimum gain of the antenna
Constant	ANTENNA_DEFAULT_PATTERN 0 Default Pattern
Constant	ANTENNA_OMNIDIRECTIONAL_PATTERN -1 OMNIDIRECTIONAL PATTERN
Constant	ANTENNA_PATTERN_NOT_SET -2 Const for Pattern of antenna not set
Constant	AZIMUTH_INDEX 0 Const for azimuth index of antenna Pattern
Constant	ELEVATION_INDEX 1 Const for elevation index of antenna Pattern
Constant	MAX_ANTENNA_NUM_LINES 30 Const for the line number in the antennaModelInput
Constant	AZIMUTH_ELEVATION_INDEX 2

	Const for the memory allocation of azimuth and elevation gain array.
Constant	NSMA_PATTERN_START_LINE_NUMBER 10 Const represents the basic pattern starting point in NSMA file
Constant	NSMA_MAX_STARTLINE 41 Const represents the Revised pattern max line number where the revised NSMA pattern can start.

Function / Macro Detail

Function / Macro	Format
ANTENNA_Init Initialize antennas.	void ANTENNA_Init (Node* node, int phyIndex, const NodeInput* nodeInput) Parameters: <ul style="list-style-type: none"> • node - node being initialized. • phyIndex - interface for which physical to be • nodeInput - structure containing contents of input Returns: <ul style="list-style-type: none"> • void - NULL
ANTENNA_ReadPatterns Read in the azimuth pattern file.	void ANTENNA_ReadPatterns (Node* node, int phyIndex, const NodeInput* antennaInput, int* numPatterns, int* steerablePatternSetRepeatSectorAngle, float*** pattern_dB, BOOL azimuthPlane) Parameters: <ul style="list-style-type: none"> • node - node being used. • phyIndex - interface for which physical to be • antennaInput - structure containing contents of • numPatterns - contains the number of patterns • steerablePatternSetRepeatSectorAngle - contains • pattern_dB - array used to store the gain values • azimuthPlane - shows whether the file is azimuth Returns: <ul style="list-style-type: none"> • void - NULL

ANTENNA_ReadNsmaPatterns	<p>Read in the NSMA pattern file.</p> <p>void ANTENNA_ReadNsmaPatterns (Node* node, int phyIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node being used. • phyIndex - interface for which physical <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
ANTENNA_ReadRevisedNsmaPatterns	<p>Read in the Revised NSMA pattern file.</p> <p>void ANTENNA_ReadRevisedNsmaPatterns (Node* node, int phyIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node being used. • phyIndex - interface for which physical <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
ANTENNA_Read3DAsciiPatterns	<p>Used to read ASCII 3D pattern file.</p> <p>void ANTENNA_Read3DAsciiPatterns (Node* node, int phyIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node being used. • phyIndex - interface for which physical <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
ANTENNA_Read2DAsciiPatterns	<p>Used to read ASCII 2D pattern file.</p> <p>void ANTENNA_Read2DAsciiPatterns (Node* node, int phyIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node being used. • phyIndex - interface for which physical <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
ANTENNA_OmniDirectionalInit	<p>Initialize omnidirectional antenna from the antenna model file.</p> <p>void ANTENNA_OmniDirectionalInit (Node* node, const NodeInput* nodeInput, int phyIndex, const AntennaModelGlobal* antennaModel)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node being initialized.

	<ul style="list-style-type: none"> • <code>nodeInput</code> - pointer to node input • <code>phyIndex</code> - interface for which physical to be • <code>antennaModel</code> - pointer to AntennaModelGlobal <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
ANTENNA_OmniDirectionalInitFromConfigFile	<p>Initialize omnidirectional antenna from the default.config file.</p> <p>void ANTENNA_OmniDirectionalInitFromConfigFile (Node* node, int phyIndex, const NodeInput* nodeInput)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node being initialized. • <code>phyIndex</code> - interface for which physical to be • <code>nodeInput</code> - structure containing contents of input <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
ANTENNA_InitFromConfigFile	<p>Initialize antenna from the default.config file.</p> <p>void ANTENNA_InitFromConfigFile (Node* node, int phyIndex, const NodeInput* nodeInput)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node being initialized. • <code>phyIndex</code> - interface for which physical to be • <code>nodeInput</code> - structure containing contents of input <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
ANTENNA_IsInOmnidirectionalMode	<p>Is antenna in omnidirectional mode.</p> <p>BOOL ANTENNA_IsInOmnidirectionalMode (Node* node, int phyIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node being used • <code>phyIndex</code> - interface for which physical to be used <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - returns TRUE if antenna is in omnidirectional mode
ANTENNA_ReturnPatternIndex	<p>Return nodes current pattern index.</p> <p>int ANTENNA_ReturnPatternIndex (Node* node, int phyIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node being used

	<ul style="list-style-type: none"> • <code>phyIndex</code> - interface for which physical to use <p>Returns:</p> <ul style="list-style-type: none"> • <code>int</code> - returns pattern index
ANTENNA_ReturnHeight	<p>float ANTENNA_ReturnHeight (Node* node, int phyIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node being used • <code>phyIndex</code> - interface for which physical to be used <p>Returns:</p> <ul style="list-style-type: none"> • <code>float</code> - height in meters
ANTENNA_ReturnSystemLossIndB	<p>double ANTENNA_ReturnSystemLossIndB (Node* node, int phyIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node being used • <code>phyIndex</code> - interface for which physical to be used <p>Returns:</p> <ul style="list-style-type: none"> • <code>double</code> - loss in dB
ANTENNA_GainForThisDirection	<p>float ANTENNA_GainForThisDirection (Node* node, int phyIndex, Orientation DOA)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node being used • <code>phyIndex</code> - interface for which physical to be used • <code>DOA</code> - direction of antenna <p>Returns:</p> <ul style="list-style-type: none"> • <code>float</code> - gain in dB
ANTENNA_GainForThisDirectionWithPatternIndex	<p>float ANTENNA_GainForThisDirectionWithPatternIndex (Node* node, int phyIndex, int patternIndex, Orientation DOA)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node being used • <code>phyIndex</code> - interface for which physical to be used • <code>patternIndex</code> - pattern index to use

	<ul style="list-style-type: none"> • DOA - direction of antenna <p>Returns:</p> <ul style="list-style-type: none"> • float - gain in dB
ANTENNA_GainForThisSignal	<p>float ANTENNA_GainForThisSignal (Node* node, int phyIndex, PropRxInfo* propRxInfo)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node being used • phyIndex - interface for which physical to be used • propRxInfo - receiver propagation info <p>Returns:</p> <ul style="list-style-type: none"> • float - gain in dB
ANTENNA_DefaultGainForThisSignal	<p>float ANTENNA_DefaultGainForThisSignal (Node* node, int phyIndex, PropRxInfo* propRxInfo)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node being used • phyIndex - interface for which physical to be used • propRxInfo - receiver propagation info <p>Returns:</p> <ul style="list-style-type: none"> • float - gain in dB
ANTENNA_LockAntennaDirection	<p>void ANTENNA_LockAntennaDirection (Node* node, int phyIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node being used • phyIndex - interface for which physical to be used <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
ANTENNA_UnlockAntennaDirection	<p>void ANTENNA_UnlockAntennaDirection (Node* node, int phyIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node being used • phyIndex - interface for which physical to be used

	<p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
ANTENNA_DirectionIsLocked	<p>BOOL ANTENNA_DirectionIsLocked (<code>Node* node, int phyIndex</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node being used • <code>phyIndex</code> - interface for which physical to be used <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - returns TRUE if the antenna direction is locked
ANTENNA_IsLocked	<p>BOOL ANTENNA_IsLocked (<code>Node* node, int phyIndex</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node being used • <code>phyIndex</code> - interface for which physical to be used <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - Returns TRUE if antenna is locked.
ANTENNA_SetToDefaultMode	<p>void ANTENNA_SetToDefaultMode (<code>Node* node, int phyIndex</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node being used • <code>phyIndex</code> - interface for which physical to be used <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
ANTENNA_SetToBestGainConfigurationForThisSignal	<p>void ANTENNA_SetToBestGainConfigurationForThisSignal (<code>Node* node, int phyIndex, PropRxInfo* propRxInfo</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node being used • <code>phyIndex</code> - interface for which physical to be used • <code>propRxInfo</code> - receiver propagation info <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL

ANTENNA_SetBestConfigurationForAzimuth	<p>void ANTENNA_SetBestConfigurationForAzimuth (Node* node, int phyIndex, double azimuth)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node being used • phyIndex - interface for which physical to be used • azimuth - the azimuth <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
ANTENNA_GetSteeringAngle	<p>void ANTENNA_GetSteeringAngle (Node* node, int phyIndex, Orientation* angle)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node being used • phyIndex - interface for which physical to be used • angle - For returning the angle <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
ANTENNA_SetSteeringAngle	<p>void ANTENNA_SetSteeringAngle (Node* node, int phyIndex, Orientation angle)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node being used • phyIndex - interface for which physical to be used • angle - Steering angle to be <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
ANTENNA_ReturnAsciiPatternFile	<p>void ANTENNA_ReturnAsciiPatternFile (Node* node, int phyIndex, const NodeInput* antennaModelInput)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node being used • phyIndex - interface for which physical • antennaModelInput - structure containing <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL

ANTENNA_ReturnNsmaPatternFile Read in the NSMA pattern .	<p>void ANTENNA_ReturnNsmaPatternFile (Node* node, int phyIndex, const NodeInput* antennaModelInput, AntennaPatterns* antennaPatterns)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node being used • phyIndex - interface for which • antennaModelInput - structure containing • antennaPatterns - Pointer to <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
Used to read Qualnet Traditional pattern file ANTENNA_ReturnTraditionalPatternFile	<p>void ANTENNA_ReturnTraditionalPatternFile (Node* node, int phyIndex, const NodeInput* antennaModelInput)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node being used • phyIndex - interface for which • antennaModelInput - structure containing <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
Reads the antenna configuration parameters into the NodeInput structure. ANTENNA_MakeAntennaModelInput	<p>NodeInput * ANTENNA_MakeAntennaModelInput (Node* node, char* buf)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node being used • buf - Path to input file. <p>Returns:</p> <ul style="list-style-type: none"> • NodeInput * - pointer to nodeInput structure



QualNet® is a Registered Trademark of [Scalable Network Technologies, Inc.](#)

Copyright © 2001-2012 [Scalable Network Technologies, Inc.](#). All rights reserved.

QualNet 6.1 API Reference

ANTENNA_GLOBAL

This file describes additional data structures and functions used by antenna models.

Constant / Data Structure Summary

Type	Name
CONSTANT	MAX_ANTENNA_MODELS Maximum number of models to allow.
CONSTANT	MAX_ANTENNA_PATTERNS Maximum number of antenna patterns to allow.
ENUMERATION	AntennaModelType Different types of antenna models supported.
ENUMERATION	AntennaPatternType Different types of antenna pattern types supported.
ENUMERATION	NSMAPatternVersion Different types of NSMA pattern versions supported.
ENUMERATION	AntennaGainUnit Different types of antenna gain units supported.
ENUMERATION	AntennaPatternUnit Different types of antenna pattern units supported.
STRUCT	struct_antenna_pattern_element Structure for antenna pattern elements
STRUCT	struct_antenna_pattern

	Structure for antenna pattern struct_antenna_Global_model
STRUCT	Structure for antenna model

Function / Macro Summary

Return Type	Summary
void	ANTENNA_GlobalAntennaModelPreInitialize (PartitionData* partitionData) Preinitialize the global antenna structs.
void	ANTENNA_GlobalAntennaPatternPreInitialize (PartitionData* partitionData) Preinitialize the global antenna structs.
AntennaPattern*	ANTENNA_GlobalModelAssignPattern (Node* node, int phyIndex, const NodeInput* antennaModelInput) used to assign global radiation pattern for each antenna.
void	ANTENNA_GlobalAntennaModelInit (Node* node, int phyIndex, const NodeInput* antennaModelInput) Reads the antenna configuration parameters into the global antenna model structure.
Void	ANTENNA_GlobalAntennaPatternInitFromFile (Node* node, int phyIndex, const char* antennaPatternName, BOOL steer) Init the antenna pattern structure for pattern name for the Old antenna model.
Void	ANTENNA_GlobalAntennaPatternInit (Node* node, int phyIndex, const NodeInput* antennaModelInput, const char* antennaPatternName) Init the antenna pattern structure for pattern name.
AntennaModelGlobal*	ANTENNA_GlobalAntennaModelAlloc (PartitionData* partitionData) Alloc a new model.
AntennaModelGlobal*	ANTENNA_GlobalAntennaModelGet (PartitionData* partitionData, const char* antennamodelName)

	<p>Return the model based on the name.</p>
AntennaPattern*	<p>ANTENNA_GlobalAntennaPatternGet(PartitionData* partitionData, const char* antennaPatternName)</p>
void	<p>Return the antenna pattern based on the name. ANTENNA_GeneratePatterName(Node* node, int phyIndex, const NodeInput* antennaModelInput, char* antennaPatternName)</p> <p>Generate the Pattern name base on Pattern type.</p>

Constant / Data Structure Detail

Constant	<p>MAX_ANTENNA_MODELS 50</p> <p>Maximum number of models to allow.</p>
Constant	<p>MAX_ANTENNA_PATTERNS 50</p> <p>Maximum number of antenna patterns to allow.</p>
Enumeration	<p>AntennaModelType</p> <p>Different types of antenna models supported.</p>
Enumeration	<p>AntennaPatternType</p> <p>Different types of antenna pattern types supported.</p>
Enumeration	<p>NSMAPatternVersion</p> <p>Different types of NSMA pattern versions supported.</p>
Enumeration	<p>AntennaGainUnit</p> <p>Different types of antenna gain units supported.</p>
Enumeration	<p>AntennaPatternUnit</p> <p>Different types of antenna pattern units supported.</p>

Structure	struct_antenna_pattern_element Structure for antenna pattern elements
Structure	struct_antenna_pattern Structure for antenna pattern
Structure	struct_antenna_Global_model Structure for antenna model

Function / Macro Detail

Function / Macro	Format
ANTENNA_GlobalAntennaModelPreInitialize Preinitialize the global antenna structs.	void ANTENNA_GlobalAntennaModelPreInitialize (PartitionData* partitionData) Parameters: <ul style="list-style-type: none">• partitionData - Pointer to partition data. Returns: <ul style="list-style-type: none">• void - NULL
ANTENNA_GlobalAntennaPatternPreInitialize Preinitialize the global antenna structs.	void ANTENNA_GlobalAntennaPatternPreInitialize (PartitionData* partitionData) Parameters: <ul style="list-style-type: none">• partitionData - Pointer to partition data. Returns: <ul style="list-style-type: none">• void - NULL
ANTENNA_GlobalModelAssignPattern used to assign global radiation pattern for each antenna.	AntennaPattern* ANTENNA_GlobalModelAssignPattern (Node* node, int phyIndex, const NodeInput* antennaModelInput) Parameters: <ul style="list-style-type: none">• node - node being used.• phyIndex - interface for which physical to be• antennaModelInput - structure containing Returns:

	<ul style="list-style-type: none"> • <code>AntennaPattern*</code> - Pointer to the global antenna pattern structure.
ANTENNA_GlobalAntennaModelInit	<p>void ANTENNA_GlobalAntennaModelInit (<code>Node* node, int phyIndex, const NodeInput* antennaModelInput</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node being used. • <code>phyIndex</code> - interface for which physical to be • <code>antennaModelInput</code> - structure containing <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
ANTENNA_GlobalAntennaPatternInitFromConfigFile	<p>Void ANTENNA_GlobalAntennaPatternInitFromConfigFile (<code>Node* node, int phyIndex, const char* antennaPatternName, BOOL steer</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node being used. • <code>phyIndex</code> - interface for which physical to be • <code>antennaPatternName</code> - antenna pattern name to be • <code>steer</code> - A boolean variable to differentiate which <p>Returns:</p> <ul style="list-style-type: none"> • <code>Void</code> - NULL
ANTENNA_GlobalAntennaPatternInit	<p>Void ANTENNA_GlobalAntennaPatternInit (<code>Node* node, int phyIndex, const NodeInput* antennaModelInput, const char* antennaPatternName</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node being used. • <code>phyIndex</code> - interface for which physical to be • <code>antennaModelInput</code> - structure containing • <code>antennaPatternName</code> - antenna pattern name to be <p>Returns:</p> <ul style="list-style-type: none"> • <code>Void</code> - NULL
ANTENNA_GlobalAntennaModelAlloc	<p><code>AntennaModelGlobal* ANTENNA_GlobalAntennaModelAlloc (PartitionData* partitionData)</code></p> <p>Parameters:</p>

ANTENNA_GLOBAL	<p>Alloc a new model.</p> <p>• <code>partitionData</code> - Pointer to partition data.</p> <p>Returns:</p> <ul style="list-style-type: none"> • <code>AntennaModelGlobal*</code> - Pointer to the global antenna model structure.
ANTENNA_GlobalAntennaModelGet	<p>Return the model based on the name.</p> <p><code>AntennaModelGlobal* ANTENNA_GlobalAntennaModelGet (PartitionData* partitionData, const char* antennamodelName)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>partitionData</code> - Pointer to partition data. • <code>antennamodelName</code> - contains the name of the <p>Returns:</p> <ul style="list-style-type: none"> • <code>AntennaModelGlobal*</code> - Pointer to the global antenna model structure.
ANTENNA_GlobalAntennaPatternGet	<p>Return the antenna pattern based on the name.</p> <p><code>AntennaPattern* ANTENNA_GlobalAntennaPatternGet (PartitionData* partitionData, const char* antennaPatternName)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>partitionData</code> - Pointer to partition data. • <code>antennaPatternName</code> - contains the name of the <p>Returns:</p> <ul style="list-style-type: none"> • <code>AntennaPattern*</code> - Pointer to the global antenna pattern structure.
ANTENNA_GeneratePatterName	<p>Generate the Pattern name base on Pattern type.</p> <p><code>void ANTENNA_GeneratePatterName (Node* node, int phyIndex, const NodeInput* antennaModelInput, char* antennaPatternName)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node being used. • <code>phyIndex</code> - interface for which physical to be • <code>antennaModelInput</code> - structure containing • <code>antennaPatternName</code> - antenna pattern name to be <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL



Contact [QualNet Support](#) for questions pertaining to the QualNet API Reference. This document is confidential and proprietary.
It may not be reproduced or distributed without the expressed written consent of [Scalable Network Technologies](#).

QualNet® is a Registered Trademark of [Scalable Network Technologies](#).

Copyright © 2001-2012 [Scalable Network Technologies, Inc.](#). All rights reserved.

QualNet 6.1 API Reference

API

This file enumerates the basic message/events exchanged during the simulation process and the various layer functions (initialize, finalize, and event handling functions) and other miscellaneous routines and data structure definitions.

Constant / Data Structure Summary

Type	Name
ENUMERATION	MESSAGE/EVENT
	Event/message types exchanged in the simulation
ENUMERATION	TransportType
	Transport type to check reliable, unreliable or TADIL network for Link16 or Link11
ENUMERATION	DestinationType
	Interface IP address type
STRUCT	PhyBatteryPower
	Used by App layer and Phy layer to exchange battery power
STRUCT	PacketNetworkToApp
	Network to application layer packet structure
STRUCT	NetworkToTransportInfo
	Network To Transport layer Information structure
STRUCT	PacketTransportNetwork
	Transport to network layer packet structure
STRUCT	TcpTimerPacket
	TCP timer packet

STRUCT	AppToUdpSend
	Additional information given to UDP from applications. This information is saved in the info field of a message.
STRUCT	ZigbeeAppInfo
	Structure used for zigbee GTS implementation
STRUCT	UdpToAppRecv
	Additional information given to applications from UDP. This information is saved in the info field of a message.
STRUCT	AppToRsvpSend
	send response structure from application layer
STRUCT	TransportToAppListenResult
	Report the result of application's listen request.
STRUCT	TransportToAppOpenResult
	Report the result of opening a connection.
STRUCT	TransportToAppDataSent
	Report the result of sending application data.
STRUCT	TransportToAppDataReceived
	Deliver data to application.
STRUCT	TransportToAppCloseResult
	Report the result of closing a connection.
STRUCT	AppToTcpListen
	Application announces willingness to accept connections on given port.
STRUCT	AppToTcpOpen
	Application attempts to establish a connection
STRUCT	AppToTcpSend

	Application wants to send some data over the connection AppToTcpClose
STRUCT	Application wants to release the connection AppToTcpConnSetup
	Application sets up connection at the local end Needed for NS TCP to fake connection setup AppQosToNetworkSend
STRUCT	Application uses this structure in its info field to perform the initialization of a new QoS connection with its QoS requirements. NetworkToAppQosConnectionStatus
	Q-OSPF uses this structure to report status of a session requested by the application for Quality of Service.

Function / Macro Summary

Return Type	Summary
void	CHANNEL_Initialize (Node* node, const NodeInput* nodeInput) Initialization function for channel
void	PHY_Init (Node* node, const NodeInput* nodeInput) Initialization function for physical layer
void	MAC_Initialize (Node* node, const NodeInput* nodeInput) Initialization function for the MAC layer
void	NETWORK_PreInit (Node* node, const NodeInput* nodeInput) Pre-Initialization function for Network layer
void	NETWORK_Initialize (Node* node, const NodeInput* nodeInput) Initialization function for Network layer

void	<u>TRANSPORT Initialize</u> (Node* node, const NodeInput* nodeInput)
	Initialization function for transport layer
void	<u>APP Initialize</u> (Node* node, const NodeInput* nodeInput)
	Initialization function for Application layer
void	<u>USER Initialize</u> (Node* node, const NodeInput* nodeInput)
	Initialization function for User layer
void	<u>APP InitializeApplications</u> (Node* firstNode, const NodeInput* nodeInput)
	Initialization function for applications in APPLICATION layer
void	<u>ATMLAYER2 Initialize</u> (Node* node, const NodeInput* nodeInput)
	Initialization function for the ATM Layer2.
void	<u>ADAPTATION Initialize</u> (Node* node, const NodeInput* nodeInput)
	Initialization function for Adaptation layer
void	<u>CHANNEL Finalize</u> (Node * node)
	To collect results of simulation at the end for channels
void	<u>PHY Finalize</u> (Node * node)
	To collect results of simulation at the end for the PHYSICAL layer
void	<u>MAC Finalize</u> (Node * node)
	To collect results of simulation at the end for the mac layers
void	<u>NETWORK Finalize</u> (Node * node)
	To collect results of simulation at the end for network layers
void	<u>TRANSPORT Finalize</u> (Node * node)
	To collect results of simulation at the end for transport layers
void	<u>APP Finalize</u> (Node * node)

	To collect results of simulation at the end for application layers
void	<u>USER_Finalize</u> (Node * node)
	To collect results of simulation at the end for user layers
void	<u>ATMLAYER2_Finalize</u> (Node * node)
	To collect results at the end of the simulation.
void	<u>ADAPTATION_Finalize</u> (Node * node)
	To collect results of simulation at the end for network layers
void	<u>CHANNEL_ProcessEvent</u> (Node* node, Message* msg)
	Processes the message/event of physical layer received by the node thus simulating the PHYSICAL layer behaviour
void	<u>PHY_ProcessEvent</u> (Node* node, Message* msg)
	Processes the message/event of physical layer received by the node thus simulating the PHYSICAL layer behaviour
void	<u>MAC_ProcessEvent</u> (Node* node, Message* msg)
	Processes the message/event of MAC layer received by the node thus simulating the MAC layer behaviour
void	<u>NETWORK_ProcessEvent</u> (Node* node, Message* msg)
	Processes the message/event received by the node thus simulating the NETWORK layer behaviour
void	<u>TRANSPORT_ProcessEvent</u> (Node* node, Message* msg)
	Processes the message/event received by the node thus simulating the TRANSPORT layer behaviour
void	<u>APP_ProcessEvent</u> (Node* node, Message* msg)
	Processes the message/event received by the node thus simulating the APPLICATION layer behaviour
void	<u>USER_ProcessEvent</u> (Node* node, Message* msg)
	Processes the message/event received by the node thus simulating the USER layer behaviour
void	<u>ATMLAYER2_ProcessEvent</u> (Node* node, Message* msg)

	Processes the message/event of ATM_LAYER2 layer received by the node thus simulating the ATM_LAYER2 layer behaviour <u>ADAPTATION_ProcessEvent</u> (Node* node, Message* msg)
void	Processes the message/event received by the node thus simulating the ADAPTATION layer behaviour <u>MAC_RunTimeStat</u> (Node* node)
void	To print runtime statistics for the MAC layer <u>NETWORK_RunTimeStat</u> (Node* node)
void	To print runtime statistics for the NETWORK layer <u>TRANSPORT_RunTimeStat</u> (Node* node)
void	To print runtime statistics for the TRANSPORT layer <u>APP_RunTimeStat</u> (Node* node)
	To print runtime statistics for the APPLICATION layer

Constant / Data Structure Detail

Enumeration	MESSAGE/EVENT Event/message types exchanged in the simulation
Enumeration	TransportType Transport type to check reliable, unreliable or TADIL network for Link16 or Link11
Enumeration	DestinationType Interface IP address type
Structure	PhyBatteryPower

	Used by App layer and Phy layer to exchange battery power
Structure	<p>PacketNetworkToApp</p> <p>Network to application layer packet structure</p>
Structure	<p>NetworkToTransportInfo</p> <p>Network To Transport layer Information structure</p>
Structure	<p>PacketTransportNetwork</p> <p>Transport to network layer packet structure</p>
Structure	<p>TcpTimerPacket</p> <p>TCP timer packet</p>
Structure	<p>AppToUdpSend</p> <p>Additional information given to UDP from applications. This information is saved in the info field of a message.</p>
Structure	<p>ZigbeeAppInfo</p> <p>Structure used for zigbee GTS implementation</p>
Structure	<p>UdpToAppRecv</p> <p>Additional information given to applications from UDP. This information is saved in the info field of a message.</p>
Structure	<p>AppToRsvpSend</p> <p>send response structure from application layer</p>
Structure	<p>TransportToAppListenResult</p> <p>Report the result of application's listen request.</p>
Structure	<p>TransportToAppOpenResult</p> <p>Report the result of opening a connection.</p>
Structure	TransportToAppDataSent

	<p>Report the result of sending application data.</p>
Structure	<p>TransportToAppDataReceived</p> <p>Deliver data to application.</p>
Structure	<p>TransportToAppCloseResult</p> <p>Report the result of closing a connection.</p>
Structure	<p>AppToTcpListen</p> <p>Application announces willingness to accept connections on given port.</p>
Structure	<p>AppToTcpOpen</p> <p>Application attempts to establish a connection</p>
Structure	<p>AppToTcpSend</p> <p>Application wants to send some data over the connection</p>
Structure	<p>AppToTcpClose</p> <p>Application wants to release the connection</p>
Structure	<p>AppToTcpConnSetup</p> <p>Application sets up connection at the local end Needed for NS TCP to fake connection setup</p>
Structure	<p>AppQosToNetworkSend</p> <p>Application uses this structure in its info field to perform the initialization of a new QoS connection with its QoS requirements.</p>
Structure	<p>NetworkToAppQosConnectionStatus</p> <p>Q-OSPF uses this structure to report status of a session requested by the application for Quality of Service.</p>

Function / Macro Detail

Function / Macro	Format
CHANNEL_Initialize Initialization function for channel	<p>void CHANNEL_Initialize (Node* node, const NodeInput* nodeInput)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node being initialized • nodeInput - structure containing all the <p>Returns:</p> <ul style="list-style-type: none"> • void - None
PHY_Init Initialization function for physical layer	<p>void PHY_Init (Node* node, const NodeInput* nodeInput)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node being initialized • nodeInput - structure containing config file details <p>Returns:</p> <ul style="list-style-type: none"> • void - None
MAC_Initialize Initialization function for the MAC layer	<p>void MAC_Initialize (Node* node, const NodeInput* nodeInput)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node being initialized • nodeInput - structure containing input file details <p>Returns:</p> <ul style="list-style-type: none"> • void - None
NETWORK_PreInit Pre-Initialization function for Network layer	<p>void NETWORK_PreInit (Node* node, const NodeInput* nodeInput)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node being initialized • nodeInput - structure containing input file details <p>Returns:</p> <ul style="list-style-type: none"> • void - None
NETWORK_Initialize	<p>void NETWORK_Initialize (Node* node, const NodeInput* nodeInput)</p> <p>Parameters:</p>

Initialization function for Network layer	<ul style="list-style-type: none"> • <code>node</code> - node being initialized • <code>nodeInput</code> - structure containing input file details <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
TRANSPORT_Initialize Initialization function for transport layer	void TRANSPORT_Initialize (Node* node, const NodeInput* nodeInput) Parameters: <ul style="list-style-type: none"> • <code>node</code> - node being initialized • <code>nodeInput</code> - structure containing input file details <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
APP_Initialize Initialization function for Application layer	void APP_Initialize (Node* node, const NodeInput* nodeInput) Parameters: <ul style="list-style-type: none"> • <code>node</code> - node being initialized • <code>nodeInput</code> - structure containing input file details <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
USER_Initialize Initialization function for User layer	void USER_Initialize (Node* node, const NodeInput* nodeInput) Parameters: <ul style="list-style-type: none"> • <code>node</code> - node being initialized • <code>nodeInput</code> - structure containing input file details <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
APP_InitializeApplications Initialization function for applications in APPLICATION layer	void APP_InitializeApplications (Node* firstNode, const NodeInput* nodeInput) Parameters: <ul style="list-style-type: none"> • <code>firstNode</code> - first node being initialized • <code>nodeInput</code> - structure containing input file details <p>Returns:</p>

	<p><code>void - None</code></p>
ATMLAYER2_Initialize	<p>Initialization function for the ATM Layer2.</p> <p><code>void ATMLAYER2_Initialize (Node* node, const NodeInput* nodeInput)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node being initialized • <code>nodeInput</code> - structure containing input file details <p>Returns:</p> <ul style="list-style-type: none"> • <code>void - None</code>
ADAPTATION_Initialize	<p>Initialization function for Adaptation layer</p> <p><code>void ADAPTATION_Initialize (Node* node, const NodeInput* nodeInput)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node being initialized • <code>nodeInput</code> - structure containing input file details <p>Returns:</p> <ul style="list-style-type: none"> • <code>void - None</code>
CHANNEL_Finalize	<p>To collect results of simulation at the end for channels</p> <p><code>void CHANNEL_Finalize (Node * node)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Node for which data is collected <p>Returns:</p> <ul style="list-style-type: none"> • <code>void - None</code>
PHY_Finalize	<p>To collect results of simulation at the end for the PHYSICAL layer</p> <p><code>void PHY_Finalize (Node * node)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Node for which finalization function is called <p>Returns:</p> <ul style="list-style-type: none"> • <code>void - None</code>
MAC_Finalize	<p>To collect results of simulation at the end for the mac layers</p> <p><code>void MAC_Finalize (Node * node)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Node for which finalization function is called <p>Returns:</p> <ul style="list-style-type: none"> • <code>void - None</code>

NETWORK_Finalize	<p>void NETWORK_Finalize (Node * node)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Node for which finalization function is called <p>Returns:</p> <ul style="list-style-type: none"> • void - None
TRANSPORT_Finalize	<p>void TRANSPORT_Finalize (Node * node)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Node for which finalization function is called <p>Returns:</p> <ul style="list-style-type: none"> • void - None
APP_Finalize	<p>void APP_Finalize (Node * node)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Node for which finalization function is called <p>Returns:</p> <ul style="list-style-type: none"> • void - None
USER_Finalize	<p>void USER_Finalize (Node * node)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Node for which finalization function is called <p>Returns:</p> <ul style="list-style-type: none"> • void - None
ATMLAYER2_Finalize	<p>void ATMLAYER2_Finalize (Node * node)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Node for which finalization function is called <p>Returns:</p> <ul style="list-style-type: none"> • void - None
ADAPTATION_Finalize	<p>void ADAPTATION_Finalize (Node * node)</p> <p>Parameters:</p>

<p>To collect results of simulation at the end for network layers</p>	<ul style="list-style-type: none"> • <code>node</code> - Node for which finalization function is called <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
<p>CHANNEL_ProcessEvent</p> <p>Processes the message/event of physical layer received by the node thus simulating the PHYSICAL layer behaviour</p>	<p><code>void CHANNEL_ProcessEvent (Node* node, Message* msg)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node which receives the message • <code>msg</code> - Received message structure <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
<p>PHY_ProcessEvent</p> <p>Processes the message/event of physical layer received by the node thus simulating the PHYSICAL layer behaviour</p>	<p><code>void PHY_ProcessEvent (Node* node, Message* msg)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node which receives the message • <code>msg</code> - Received message structure <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
<p>MAC_ProcessEvent</p> <p>Processes the message/event of MAC layer received by the node thus simulating the MAC layer behaviour</p>	<p><code>void MAC_ProcessEvent (Node* node, Message* msg)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node which receives the message • <code>msg</code> - Received message structure <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
<p>NETWORK_ProcessEvent</p> <p>Processes the message/event received by the node thus simulating the NETWORK layer behaviour</p>	<p><code>void NETWORK_ProcessEvent (Node* node, Message* msg)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node which receives the message • <code>msg</code> - Received message structure <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None

TRANSPORT_ProcessEvent	<p>Processes the message/event received by the node thus simulating the TRANSPORT layer behaviour</p> <p>void <code>TRANSPORT_ProcessEvent</code> (Node* node, Message* msg)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node which receives the message • msg - Received message structure <p>Returns:</p> <ul style="list-style-type: none"> • void - None
APP_ProcessEvent	<p>Processes the message/event received by the node thus simulating the APPLICATION layer behaviour</p> <p>void <code>APP_ProcessEvent</code> (Node* node, Message* msg)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node which receives the message • msg - Received message structure <p>Returns:</p> <ul style="list-style-type: none"> • void - None
USER_ProcessEvent	<p>Processes the message/event received by the node thus simulating the USER layer behaviour</p> <p>void <code>USER_ProcessEvent</code> (Node* node, Message* msg)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node which receives the message • msg - Received message structure <p>Returns:</p> <ul style="list-style-type: none"> • void - None
ATMLAYER2_ProcessEvent	<p>Processes the message/event of ATM_LAYER2 layer received by the node thus simulating the ATM_LAYER2 layer behaviour</p> <p>void <code>ATMLAYER2_ProcessEvent</code> (Node* node, Message* msg)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node which receives the message • msg - Received message structure <p>Returns:</p> <ul style="list-style-type: none"> • void - None
ADAPTATION_ProcessEvent	<p>Processes the message/event received by the node thus simulating the ADAPTATION layer behaviour</p> <p>void <code>ADAPTATION_ProcessEvent</code> (Node* node, Message* msg)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node which receives the message • msg - Received message structure

	<p>Returns:</p> <ul style="list-style-type: none"> • void - None
MAC_RunTimeStat	<p>void MAC_RunTimeStat (Node* node)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node for which statistics to be printed <p>Returns:</p> <ul style="list-style-type: none"> • void - None
NETWORK_RunTimeStat	<p>void NETWORK_RunTimeStat (Node* node)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node for which statistics to be printed <p>Returns:</p> <ul style="list-style-type: none"> • void - None
TRANSPORT_RunTimeStat	<p>void TRANSPORT_RunTimeStat (Node* node)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node for which statistics to be printed <p>Returns:</p> <ul style="list-style-type: none"> • void - None
APP_RunTimeStat	<p>void APP_RunTimeStat (Node* node)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node for which statistics to be printed <p>Returns:</p> <ul style="list-style-type: none"> • void - None



QualNet® is a Registered Trademark of [Scalable Network Technologies](#).

Copyright © 2001-2012 [Scalable Network Technologies, Inc.](#). All rights reserved.

QualNet 6.1 API Reference

APP_UTIL

This file describes Application Layer utility functions.

Function / Macro Summary

Return Type	Summary
MACRO	<p><u>APP_GetTimerType(x)</u></p> <p>Get the timerType for a received App Layer Timer.</p>
AppInfo*	<p><u>APP_RegisterNewApp</u>(Node* node, AppType appType, void * dataPtr)</p> <p>Insert a new application into the list of apps on this node.</p>
void	<p><u>APP_SetTimer</u>(Node* node, AppType appType, int connId, short sourcePort, int timerType, clocktype delay)</p> <p>Set a new App Layer Timer and send to self after delay.</p>
Message *	<p><u>APP_UdpSendNewData</u>(Node * node, AppType appType, NodeAddress sourceAddr, short sourcePort, NodeAddress destAddr, char * payload, int payloadSize, clocktype delay, TraceProtocolType traceProtocol)</p> <p>Allocate data and send to UDP.</p>
Message *	<p><u>APP_UdpSendNewDataWithPriority</u>(Node * node, AppType appType, NodeAddress sourceAddr, short sourcePort, NodeAddress destAddr, int outgoingInterface, char* payload, int payloadSize, TosType priority, clocktype delay, TraceProtocolType traceProtocol, BOOL isMdpEnabled, Int32 uniqueId, char* mdpDataObjectInfo, UInt16 mdpDataInfosize)</p> <p>Allocate data with specified priority and send to UDP.</p>
Message*	<p><u>APP_UdpSendNewDataWithPriority</u>(Node * node, AppType appType, NodeAddress sourceAddr, short sourcePort, NodeAddress destAddr, int outgoingInterface, char* payload, int payloadSize, TosType priority, clocktype delay, TraceProtocolType traceProtocol, BOOL isMdpEnabled, Int32 uniqueId)</p> <p>Allocate data with specified priority and send to UDP (For IPv6).</p>
void	<p><u>APP_UdpSendNewHeaderData</u>(Node * node, AppType appType, NodeAddress sourceAddr, short sourcePort, NodeAddress destAddr, char* header, int headerSize, char* payload, int payloadSize, clocktype delay, TraceProtocolType traceProtocol)</p> <p>Allocate header and data and send to UDP..</p>

void	<code>APP_UdpSendNewHeaderDataWithPriority(Node * node, AppType appType, NodeAddress sourceAddr, short sourcePort, NodeAddress destAddr, int outgoingInterface, char* header, int headerSize, char* payload, int payloadSize, TosType priority, clocktype delay, TraceProtocolType traceProtocol)</code>
Message *	<p>Allocate header and data with specified priority and send to UDP</p> <code>APP_UdpSendNewHeaderVirtualDataWithPriority(Node * node, AppType appType, NodeAddress sourceAddr, short sourcePort, NodeAddress destAddr, char* header, int headerSize, int payloadSize, TosType priority, clocktype delay, TraceProtocolType traceProtocol, BOOL isMdpEnabled, Int32 mdpUniqueId)</code>
void	<p>Allocate header + virtual data with specified priority and send to UDP</p> <code>APP_UdpSendNewHeaderVirtualDataWithPriority(Node * node, NodeAddress sourceAddr, short sourcePort, NodeAddress destAddr, short destinationPort, char* infoData, int infoSize, int infoType, int payloadSize, TosType priority, clocktype delay, TraceProtocolType traceProtocol, BOOL isMdpEnabled, Int32 mdpUniqueId)</code>
Message*	<p>Allocate header + virtual data with specified priority and send to UDP. Data is sent to a non-default destination port (port number may not have same value as the AppType).</p> <code>APP_UdpSendNewHeaderVirtualDataWithPriority(Node * node, Address sourceAddr, short sourcePort, Address destAddr, short destinationPort, char* infoData, int infoSize, int infoType, int payloadSize, TosType priority, clocktype delay, TraceProtocolType traceProtocol, char* appname, BOOL isMdpEnabled, Int32 mdpUniqueId)</code>
void	<p>(Overloaded for IPv6) Allocate header + virtual data with specified priority and send to UDP. Data is sent to a non-default destination port (port number may not have same value as the AppType).</p> <code>APP_UdpSendNewHeaderVirtualDataWithPriority(Node * node, Address sourceAddr, short sourcePort, Address destAddr, short destinationPort, char* header, int headerSize, int payloadSize, TosType priority, clocktype delay, TraceProtocolType traceProtocol, char* infoData, int infoSize, int infoType, BOOL isMdpEnabled, Int32 mdpUniqueId)</code>
void	<p>(Overloaded for IPv6) Allocate header + virtual data with specified priority and send to UDP. Data is sent to a non-default destination port (port number may not have same value as the AppType).</p> <code>APP_UdpSendNewHeaderVirtualDataWithPriority(Node * node, NodeAddress sourceAddr, short sourcePort, NodeAddress destAddr, short destinationPort, char* header, int headerSize, int payloadSize, TosType priority, clocktype delay, TraceProtocolType traceProtocol)</code>
void	<p>Allocate header + virtual data with specified priority and send to UDP. Data is sent to a non-default destination port (port number may not have same value as the AppType).</p> <code>APP_UdpSendNewHeaderVirtualDataWithPriority(Node * node, AppType appType, Address sourceAddr, short sourcePort, Address destAddr, char* header, int headerSize, int payloadSize, TosType priority, clocktype delay, char* mdpInfo, TraceProtocolType traceProtocol, Message* theMsgPtr)</code>
void	<p>(Overloaded for MDP) Allocate actual + virtual data with specified priority and send to UDP. Data is sent to a non-default destination port (port number may not have same value as the AppType).</p> <code>APP_UdpSendNewHeaderVirtualDataWithPriority(Node * node, Address sourceAddr, short sourcePort, Address destAddr, short destinationPort, char* header, int headerSize, int payloadSize, TosType priority, clocktype delay, TraceProtocolType traceProtocol)</code>

	(Overloaded for IPv6) Allocate header + virtual data with specified priority and send to UDP. Data is sent to a non-default destination port (port number may not have same value as the AppType).
Message *	APP_UdpSendNewHeaderVirtualDataWithPriority (Node * node, Address sourceAddr, short sourcePort, Address destAddr, short destinationPort, char* header, int headerSize, int payloadSize, TosType priority, clocktype delay, TraceProtocolType traceProtocol)
	(Overloaded for IPv6) Allocate header + virtual data with specified priority and send to UDP. Data is sent to a non-default destination port (port number may not have same value as the AppType).
Message *	APP_UdpCreateNewHeaderVirtualDataWithPriority (Node * node, Address sourceAddr, short sourcePort, Address destAddr, short destinationPort, char* header, int headerSize, int payloadSize, TosType priority, TraceProtocolType traceProtocol)
	Create the message.
Message *	APP_UdpSendNewHeaderVirtualDataWithPriority (Node * node, Address sourceAddr, short sourcePort, Address destAddr, short destinationPort, char* header, int headerSize, int payloadSize, TosType priority, clocktype delay, TraceProtocolType traceProtocol, BOOL isMdpEnabled, Int32 mdpUniqueId)
	(Overloaded for IPv6) Allocate header + virtual data with specified priority and send to UDP. Data is sent to a non-default destination port (port number may not have same value as the AppType).
void	APP_UdpSendNewHeaderVirtualDataWithPriority (Node * node, Address sourceAddr, short sourcePort, Address destAddr, short destinationPort, char* header, int headerSize, int payloadSize, TosType priority, clocktype endTime, UInt32 itemSize, D_Clocktype interval, clocktype delay, TraceProtocolType traceProtocol)
	(Overloaded for IPv6) Allocate header + virtual data with specified priority and send to UDP. Data is sent to a non-default destination port (port number may not have same value as the AppType).
void	APP_TcpServerListen .(Node * node, AppType appType, NodeAddress serverAddr, short serverPort)
	Listen on a server port.
void	APP_TcpServerListen .(Node * node, AppType appType, Address serverAddr, short serverPort)
	(Overloaded for IPv6) Listen on a server port.
void	APP_TcpServerListenWithPriority .(Node * node, AppType appType, NodeAddress serverAddr, short serverPort, TosType priority)
	Listen on a server port with specified priority.
void	APP_TcpServerListenWithPriority .(Node * node, AppType appType, Address serverAddr, short serverPort, TosType priority)
	Listen on a server port with specified priority. (Overloaded for IPv6)
void	APP_TcpOpenConnection .(Node * node, appType appType, NodeAddress localAddr, short localPort, NodeAddress remoteAddr, short remotePort, int uniqueId, clocktype waitTime)

	<p>Open a connection.</p>
void	<p>APP_TcpOpenConnection.(Node * node, appType appType, Address localAddr, short localPort, Address remoteAddr, short remotePort, int uniqueId, clocktype waitTime)</p>
	<p>(Overloaded for IPv6) Open a connection.</p>
void	<p>APP_TcpOpenConnection.(Node * node, appType appType, NodeAddress localAddr, short localPort, NodeAddress remoteAddr, short remotePort, int uniqueId, clocktype waitTime, int outgoingInterface)</p>
	<p>Open a connection.</p>
void	<p>APP_TcpOpenConnection.(Node * node, appType appType, Address localAddr, short localPort, Address remoteAddr, short remotePort, int uniqueId, clocktype waitTime, int outgoingInterface)</p>
	<p>(Overloaded for IPv6) Open a connection.</p>
void	<p>APP_TcpOpenConnectionWithPriority.(Node * node, appType appType, NodeAddress localAddr, short localPort, NodeAddress remoteAddr, short remotePort, int uniqueId, clocktype waitTime, TosType priority)</p>
	<p>Open a connection with specified priority.</p>
void	<p>APP_TcpOpenConnectionWithPriority..(Node * node, appType appType, Address localAddr, short localPort, Address remoteAddr, short remotePort, int uniqueId, clocktype waitTime, TosType priority)</p>
	<p>Open a connection with specified priority. (Overloaded for IPv6)</p>
Message *	<p>App_TcpCreateMessage.(Node * node, int connId, char * payload, int length, traceProtocolType traceProtocol)</p>
	<p>Create the message.</p>
Message *	<p>APP_TcpSendData.(Node * node, int connId, char * payload, int length, raceProtocolType traceProtocol)</p>
	<p>send an application data unit.</p>
Message*	<p>APP_TcpSendNewHeaderVirtualData.(Node * node, int connId, char * header, int headerLength, int payloadSize, raceProtocolType traceProtocol)</p>
	<p>Send header and virtual data using TCP.</p>
void	<p>APP_TcpCloseConnection(Node * node, int connId)</p>
	<p>Close the connection.</p>
void	<p>APP_InitMulticastGroupMembershipIfAny(Node * node, const NodeInput nodeInput)</p>
	<p>Start process of joining multicast group if need to do so.</p>

void	<code>APP_CheckMulticastByParsingSourceAndDestString</code> (Node * node, const char * inputString, const char * sourceString, NodeAddress * sourceNodeId, NodeAddress * sourceAddr, const char * destString, NodeAddress * destNodeId, NodeAddress * destAddr, BOOL * isDestMulticast)	Application input parsing API. Parses the source and destination strings. At the same time validates those strings for multicast address.
void	<code>APP_ParsingSourceAndDestString</code> (Node * node, const char * inputString, const char * sourceString, NodeAddress * sourceNodeId, NodeAddress * sourceAddr, const char * destString, NodeAddress * destNodeId, NodeAddress * destAddr, DestinationType * destType)	API to parse the input source and destination strings read from the *.app file. At the same time checks and fills the destination type parameter.
void	<code>APP_ParsingSourceAndDestString</code> (Node * node, const char * inputString, const char * sourceString, NodeId * sourceNodeId, Address * sourceAddr, const char * destString, NodeId * destNodeId, Address * destAddr, DestinationType * destType)	API to parse the input source and destination strings read from the *.app file. At the same time checks and fills the destination type parameter.
AppInfo*	<code>APP_RegisterNewApp</code> (Node* node, AppType appType, void * dataPtr, short myPort)	Also inserts the port number being used for this app in the port table.
BOOL	<code>APP_IsFreePort</code> (Node* node, short portNumber)	there is an application running at the node that uses an AppType that has been assigned the same value as this port number. This is done since applications such as CBR use the value of AppType as destination port.
short	<code>APP_GetFreePort</code> (Node* node)	
void	<code>APP_InsertInPortTable</code> (Node* node, AppType appType, short myPort)	
unsigned short	<code>APP_GetProtocolType</code> (Node* node, Message* msg)	
BOOL	<code>APP_AssignTos</code> (char array tosString, char array tosValString, unsigned * tosVal)	Application input parsing API. Parses the tos string and tos value strings. At the same time validates those strings for proper ranges.
Message*	<code>APP_UdpCreateNewHeaderVirtualDataWithPriority</code> (Node * node, AppType appType, NodeAddress sourceAddr, short sourcePort, NodeAddress destAddr, char* header, int headerSize, int payloadSize, TosType priority, TraceProtocolType traceProtocol)	Allocate header + virtual data with specified priority and send to UDP. Generally used with messenger app.
Message*	<code>APP_UdpCreateNewHeaderVirtualDataWithPriority</code> (Node * node, AppType appType, Address sourceAddr, short sourcePort, Address destAddr, char* header, int headerSize, int payloadSize, TosType priority, TraceProtocolType traceProtocol)	

	Allocate header + virtual data with specified priority and send to UDP. Generally used with messenger app. APP_RegisterApp (Node* node, void * dataPtr, bool freeData)
void	Remove an application from list of apps on this node. APP_UnregisterApp (Node* node, AppType appType, void * dataPtr, short myPort)
	Also Remove the port number being used for this app in the port table.
BOOL	APP_IsFreePort (Node* node, short portNumber) there is an application running at the node that uses an AppType that has been assigned the same value as this port number. This is done since applications such as CBR use the value of AppType as destination port.
void	APP_RemoveFromPortTable (Node* node, short myPort)
SequenceNumber	APP_ReportStatsDbReceiveEvent (Node* node, Message* msg, SequenceNumber** seqCache, Int64 seqNo, clocktype delay, clocktype jitter, int size, int numRcvd, AppMsgStatus msgStatus) Report receive event to StatsDB app event table This function will check duplicate and out of order msgs

Function / Macro Detail

Function / Macro	Format
APP_GetTimerType(x)	Get the timerType for a received App Layer Timer.
APP_RegisterNewApp	AppInfo* APP_RegisterNewApp (Node* node, AppType appType, void * dataPtr) Parameters: <ul style="list-style-type: none"> • node - node that is registering the application. • appType - application type • dataPtr - pointer to the data space for this app Returns: <ul style="list-style-type: none"> • AppInfo* - pointer to the new AppInfo data structure for this app
APP_SetTimer	void APP_SetTimer (Node* node, AppType appType, int connId, short sourcePort, int timerType, clocktype delay)

	<p>Set a new App Layer Timer and send to self after delay.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node that is issuing the Timer. • <code>appType</code> - application type • <code>connId</code> - if applicable, the TCP connectionId for this timer • <code>sourcePort</code> - the source port of the application setting • <code>timerType</code> - an integer value that can be used to • <code>delay</code> - send the timer to self after this delay. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
APP_UdpSendNewData Allocate data and send to UDP.	<p><code>Message * APP_UdpSendNewData (Node * node, AppType appType, NodeAddress sourceAddr, short sourcePort, NodeAddress destAddr, char * payload, int payloadSize, clocktype delay, TraceProtocolType traceProtocol)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node that is sending the data. • <code>appType</code> - application type, to be used as destination port. • <code>sourceAddr</code> - the source sending the data. • <code>sourcePort</code> - the application source port. • <code>destAddr</code> - the destination node Id data is sent to. • <code>payload</code> - pointer to the data. • <code>payloadSize</code> - size of the data in bytes. • <code>delay</code> - send the data after this delay. • <code>traceProtocol</code> - specify the type of application used for <p>Returns:</p> <ul style="list-style-type: none"> • <code>Message *</code> - None
APP_UdpSendNewDataWithPriority Allocate data with specified priority and send to UDP.	<p><code>Message * APP_UdpSendNewDataWithPriority (Node * node, AppType appType, NodeAddress sourceAddr, short sourcePort, NodeAddress destAddr, int outgoingInterface, char* payload, int payloadSize, TosType priority, clocktype delay, TraceProtocolType traceProtocol, BOOL isMdpEnabled, Int32 uniqueId, char* mdpDataObjectInfo, UInt16 mdpDataInfosize)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node that is sending the data. • <code>appType</code> - application type, to be used as

	<ul style="list-style-type: none"> • <code>sourceAddr</code> - the source sending the data. • <code>sourcePort</code> - the application source port. • <code>destAddr</code> - the destination node Id data • <code>outgoingInterface</code> - interface used to send data. • <code>payload</code> - pointer to the data. • <code>payloadSize</code> - size of the data in bytes. • <code>priority</code> - priority of data. • <code>delay</code> - send the data after this delay. • <code>traceProtocol</code> - specify the type of application • <code>isMdpEnabled</code> - specify whether MDP is enabled. • <code>uniqueId</code> - specify uniqueId related to MDP. • <code>mdpDataObjectInfo</code> - specify the mdp data object info if any. • <code>mdpDataInfoSize</code> - specify the mdp data info size. <p>Returns:</p> <ul style="list-style-type: none"> • <code>Message *</code> - None
APP_UdpSendNewDataWithPriority	<p>Allocate data with specified priority and send to UDP (For IPv6).</p> <p>Message* APP_UdpSendNewDataWithPriority (Node * node, AppType appType, NodeAddress sourceAddr, short sourcePort, NodeAddress destAddr, int outgoingInterface, char* payload, int payloadSize, TosType priority, clocktype delay, TraceProtocolType traceProtocol, BOOL isMdpEnabled, Int32 uniqueId)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node that is sending the data. • <code>appType</code> - application type, to be used as • <code>sourceAddr</code> - the source sending the data. • <code>sourcePort</code> - the application source port. • <code>destAddr</code> - the destination node Id data • <code>outgoingInterface</code> - interface used to send data. • <code>payload</code> - pointer to the data. • <code>payloadSize</code> - size of the data in bytes. • <code>priority</code> - priority of data. • <code>delay</code>

	<ul style="list-style-type: none"> - send the data after this delay. <ul style="list-style-type: none"> • <code>traceProtocol</code> - specify the type of application • <code>isMdpEnabled</code> - specify whether MDP is enabled. • <code>uniqueId</code> - specify uniqueId related to MDP. <p>Returns:</p> <ul style="list-style-type: none"> • <code>Message*</code> - The sent message
APP_UdpSendNewHeaderData	<p>Allocate header and data and send to UDP..</p> <p><code>void APP_UdpSendNewHeaderData (Node * node, AppType appType, NodeAddress sourceAddr, short sourcePort, NodeAddress destAddr, char* header, int headerSize, char* payload, int payloadSize, clocktype delay, TraceProtocolType traceProtocol)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node that is sending the data. • <code>appType</code> - application type, to be used as • <code>sourceAddr</code> - the source sending the data. • <code>sourcePort</code> - the application source port. • <code>destAddr</code> - the destination node Id data • <code>header</code> - header of the payload. • <code>headerSize</code> - size of the header. • <code>payload</code> - pointer to the data. • <code>payloadSize</code> - size of the data in bytes. • <code>delay</code> - send the data after this delay. • <code>traceProtocol</code> - specify the type of application <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
APP_UdpSendNewHeaderDataWithPriority	<p>Allocate header and data with specified priority and send to UDP</p> <p><code>void APP_UdpSendNewHeaderDataWithPriority (Node * node, AppType appType, NodeAddress sourceAddr, short sourcePort, NodeAddress destAddr, int outgoingInterface, char* header, int headerSize, char* payload, int payloadSize, TosType priority, clocktype delay, TraceProtocolType traceProtocol)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node that is sending the data. • <code>appType</code> - application type, to be used as • <code>sourceAddr</code> - the source sending the data.

	<ul style="list-style-type: none"> • <code>sourcePort</code> - the application source port. • <code>destAddr</code> - the destination node Id data • <code>outgoingInterface</code> - interface used to send data. • <code>header</code> - header of the payload. • <code>headerSize</code> - size of the header. • <code>payload</code> - pointer to the data. • <code>payloadSize</code> - size of the data in bytes. • <code>priority</code> - priority of data. • <code>delay</code> - send the data after this delay. • <code>traceProtocol</code> - specify the type of application <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
APP_UdpSendNewHeaderVirtualDataWithPriority	<p>Message * APP_UdpSendNewHeaderVirtualDataWithPriority (Node * node, AppType appType, NodeAddress sourceAddr, short sourcePort, NodeAddress destAddr, char* header, int headerSize, int payloadSize, TosType priority, clocktype delay, TraceProtocolType traceProtocol, BOOL isMdpEnabled, Int32 mdpUniqueId)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node that is sending the data. • <code>appType</code> - application type, to be used as • <code>sourceAddr</code> - the source sending the data. • <code>sourcePort</code> - the application source port. • <code>destAddr</code> - the destination node Id data • <code>header</code> - header of the payload. • <code>headerSize</code> - size of the header. • <code>payloadSize</code> - size of the data in bytes. • <code>priority</code> - priority of data. • <code>delay</code> - send the data after this delay. • <code>traceProtocol</code> - specify the type of application • <code>isMdpEnabled</code> - status of MDP layer. • <code>mdpUniqueId</code> - unique id for MPD session.

	<p>Returns:</p> <ul style="list-style-type: none"> • Message * - None
APP_UdpSendNewHeaderVirtualDataWithPriority (Allocate header + virtual data with specified priority and send to UDP. Data is sent to a non-default destination port (port number may not have same value as the AppType).	<p>void APP_UdpSendNewHeaderVirtualDataWithPriority (Node * node, NodeAddress sourceAddr, short sourcePort, NodeAddress destAddr, short destinationPort, char* infoData, int infoSize, int infoType, int payloadSize, TosType priority, clocktype delay, TraceProtocolType traceProtocol, BOOL isMdpEnabled, Int32 mdpUniqueId)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node that is sending the data. • sourceAddr - the source sending the data. • sourcePort - the application source port. • destAddr - the destination node Id data • destinationPort - the destination port • infoData - UDP header to be added in info. • infoSize - size of the UDP header. • infoType - info type of the UDP header. • payloadSize - size of the data in bytes. • priority - priority of data. • delay - send the data after this delay. • traceProtocol - specify the type of application • isMdpEnabled - status of MDP layer. • mdpUniqueId - unique id for MPD session. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
APP_UdpSendNewHeaderVirtualDataWithPriority (Overloaded for IPv6) Allocate header + virtual data with specified priority and send to UDP. Data is sent to a non-default destination port (port number may not have same value as the AppType).	<p>Message* APP_UdpSendNewHeaderVirtualDataWithPriority (Node * node, Address sourceAddr, short sourcePort, Address destAddr, short destinationPort, char* infoData, int infoSize, int infoType, int payloadSize, TosType priority, clocktype delay, TraceProtocolType traceProtocol, char* appname, BOOL isMdpEnabled, Int32 mdpUniqueId)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node that is sending the data. • sourceAddr - the source sending the data. • sourcePort - the application source port.

	<ul style="list-style-type: none"> • <code>destAddr</code> - the destination node Id data • <code>destinationPort</code> - the destination port • <code>infoData</code> - UDP header to be added. • <code>infoSize</code> - size of the UDP header. • <code>infoType</code> - info type of the UDP header • <code>payloadSize</code> - size of the data in bytes. • <code>priority</code> - priority of data. • <code>delay</code> - send the data after this delay. • <code>traceProtocol</code> - specify the type of application • <code>appname</code> - Application name. • <code>isMdpEnabled</code> - status of MDP layer. • <code>mdpUniqueId</code> - unique id for MPD session. <p>Returns:</p> <ul style="list-style-type: none"> • <code>Message*</code> - The message that was sent
APP_UdpSendNewHeaderVirtualDataWithPriority	<p>(Overloaded for IPv6) Allocate header + virtual data with specified priority and send to UDP. Data is sent to a non-default destination port (port number may not have same value as the AppType).</p> <p><code>void APP_UdpSendNewHeaderVirtualDataWithPriority (Node * node, Address sourceAddr, short sourcePort, Address destAddr, short destinationPort, char* header, int headerSize, int payloadSize, TosType priority, clocktype delay, TraceProtocolType traceProtocol, char* infoData, int infoSize, int infoType, BOOL isMdpEnabled, Int32 mdpUniqueId)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node that is sending the data. • <code>sourceAddr</code> - the source sending the data. • <code>sourcePort</code> - the application source port • <code>destAddr</code> - the destination node Id data • <code>destinationPort</code> - the destination port • <code>header</code> - header of the payload. • <code>headerSize</code> - size of the header. • <code>payloadSize</code> - size of the data in bytes. • <code>priority</code> - priority of data. • <code>delay</code> - send the data after this delay.

	<ul style="list-style-type: none"> • <code>traceProtocol</code> - specify the type of application • <code>infoData</code> - UDP header to be added. • <code>infoSize</code> - size of the UDP header. • <code>infoType</code> - info type of the UDP header • <code>isMdpEnabled</code> - status of MDP layer. • <code>mdpUniqueId</code> - unique id for MPD session. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
APP_UdpSendNewHeaderVirtualDataWithPriority	<p>Allocate header + virtual data with specified priority and send to UDP. Data is sent to a non-default destination port (port number may not have same value as the AppType).</p> <pre>void APP_UdpSendNewHeaderVirtualDataWithPriority (Node * node, NodeAddress sourceAddr, short sourcePort, NodeAddress destAddr, short destinationPort, char* header, int headerSize, int payloadSize, TosType priority, clocktype delay, TraceProtocolType traceProtocol)</pre> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node that is sending the data. • <code>sourceAddr</code> - the source sending the data. • <code>sourcePort</code> - the application source port. • <code>destAddr</code> - the destination node Id data • <code>destinationPort</code> - the destination port • <code>header</code> - header of the payload. • <code>headerSize</code> - size of the header. • <code>payloadSize</code> - size of the data in bytes. • <code>priority</code> - priority of data. • <code>delay</code> - send the data after this delay. • <code>traceProtocol</code> - specify the type of application <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
APP_UdpSendNewHeaderVirtualDataWithPriority	<p>(Overloaded for MDP) Allocate actual + virtual data with specified priority and send to UDP. Data is sent to a non-default destination port (port number may not have</p> <pre>void APP_UdpSendNewHeaderVirtualDataWithPriority (Node * node, AppType appType, Address sourceAddr, short sourcePort, Address destAddr, char* header, int headerSize, int payloadSize, TosType priority, clocktype delay, char* mdpInfo, TraceProtocolType traceProtocol, Message* theMsgPtr)</pre> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node that is sending the data.

<p>same value as the AppType).</p>	<ul style="list-style-type: none"> • <code>appType</code> - specify the application type. • <code>sourceAddr</code> - the source sending the data. • <code>sourcePort</code> - the application source port. • <code>destAddr</code> - the destination node on which • <code>header</code> - pointer to the payload. • <code>headerSize</code> - size of the payload. • <code>payloadSize</code> - size of the virtual data in bytes. • <code>priority</code> - priority of data. • <code>delay</code> - send the data after this delay. • <code>mdpInfo</code> - persistent info for Mdp. • <code>traceProtocol</code> - specify the type of application • <code>theMsgPtr</code> - pointer the original message to copy <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
<p>APP_UdpSendNewHeaderVirtualDataWithPriority</p> <p>(Overloaded for IPv6) Allocate header + virtual data with specified priority and send to UDP. Data is sent to a non-default destination port (port number may not have same value as the AppType).</p>	<p><code>void APP_UdpSendNewHeaderVirtualDataWithPriority (Node * node, Address sourceAddr, short sourcePort, Address destAddr, short destinationPort, char* header, int headerSize, int payloadSize, TosType priority, clocktype delay, TraceProtocolType traceProtocol)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node that is sending the data. • <code>sourceAddr</code> - the source sending the data. • <code>sourcePort</code> - the application source port. • <code>destAddr</code> - the destination node Id data • <code>destinationPort</code> - the destination port • <code>header</code> - header of the payload. • <code>headerSize</code> - size of the header. • <code>payloadSize</code> - size of the data in bytes. • <code>priority</code> - priority of data. • <code>delay</code> - send the data after this delay.

	<p><code>traceProtocol</code> - specify the type of application</p> <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
APP_UdpSendNewHeaderVirtualDataWithPriority	<p>(Overloaded for IPv6) Allocate header + virtual data with specified priority and send to UDP. Data is sent to a non-default destination port (port number may not have same value as the AppType).</p> <p>Message * APP_UdpSendNewHeaderVirtualDataWithPriority (Node * node, Address sourceAddr, short sourcePort, Address destAddr, short destinationPort, char* header, int headerSize, int payloadSize, TosType priority, clocktype delay, TraceProtocolType traceProtocol)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node that is sending the data. • <code>sourceAddr</code> - the source sending the data. • <code>sourcePort</code> - the application source port. • <code>destAddr</code> - the destination node Id data • <code>destinationPort</code> - the destination port • <code>header</code> - header of the payload. • <code>headerSize</code> - size of the header. • <code>payloadSize</code> - size of the data in bytes. • <code>priority</code> - priority of data. • <code>delay</code> - send the data after this delay. • <code>traceProtocol</code> - specify the type of application <p>Returns:</p> <ul style="list-style-type: none"> • <code>Message *</code> - None
APP_UdpCreateNewHeaderVirtualDataWithPriority	<p>Create the message.</p> <p>Message * APP_UdpCreateNewHeaderVirtualDataWithPriority. (Node * node, Address sourceAddr, short sourcePort, Address destAddr, short destinationPort, char* header, int headerSize, int payloadSize, TosType priority, TraceProtocolType traceProtocol)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node that is sending the data. • <code>sourceAddr</code> - the source sending the data. • <code>sourcePort</code> - the application source port. • <code>destAddr</code> - the destination node Id data • <code>destinationPort</code> - the destination port • <code>header</code> - header of the payload.

	<ul style="list-style-type: none"> <code>headerSize</code> - size of the header. <code>payloadSize</code> - size of the data in bytes. <code>priority</code> - priority of data. <code>traceProtocol</code> - specify the type of application <p>Returns:</p> <ul style="list-style-type: none"> <code>Message *</code> - message created in the function
APP_UdpSendNewHeaderVirtualDataWithPriority	<p>(Overloaded for IPv6) Allocate header + virtual data with specified priority and send to UDP. Data is sent to a non-default destination port (port number may not have same value as the AppType).</p> <p>Message * APP_UdpSendNewHeaderVirtualDataWithPriority (Node * node, Address sourceAddr, short sourcePort, Address destAddr, short destinationPort, char* header, int headerSize, int payloadSize, TosType priority, clocktype delay, TraceProtocolType traceProtocol, BOOL isMdpEnabled, Int32 mdpUniqueId)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <code>node</code> - node that is sending the data. <code>sourceAddr</code> - the source sending the data. <code>sourcePort</code> - the application source port. <code>destAddr</code> - the destination node Id data <code>destinationPort</code> - the destination port <code>header</code> - header of the payload. <code>headerSize</code> - size of the header. <code>payloadSize</code> - size of the data in bytes. <code>priority</code> - priority of data. <code>delay</code> - send the data after this delay. <code>traceProtocol</code> - specify the type of application <code>isMdpEnabled</code> - status of MDP layer. <code>mdpUniqueId</code> - unique id for MPD session. <p>Returns:</p> <ul style="list-style-type: none"> <code>Message *</code> - None
APP_UdpSendNewHeaderVirtualDataWithPriority	<p>void APP_UdpSendNewHeaderVirtualDataWithPriority (Node * node, Address sourceAddr, short sourcePort, Address destAddr, short destinationPort, char* header, int headerSize, int payloadSize, TosType priority, clocktype endTime, UInt32 itemSize, D_Clocktype interval, clocktype delay, TraceProtocolType traceProtocol)</p> <p>(Overloaded for IPv6) Allocate header + virtual data with specified priority and send to UDP. Data is sent to a</p> <p>Parameters:</p> <ul style="list-style-type: none"> <code>node</code>

<p>non-default destination port (port number may not have same value as the AppType).</p>	<ul style="list-style-type: none"> - node that is sending the data. • <code>sourceAddr</code> - the source sending the data. • <code>sourcePort</code> - the application source port. • <code>destAddr</code> - the destination node Id data • <code>destinationPort</code> - the destination port • <code>header</code> - header of the payload. • <code>headerSize</code> - size of the header. • <code>payloadSize</code> - size of the data in bytes. • <code>priority</code> - priority of data. • <code>endTime</code> - zigbeeApp end time. • <code>itemSize</code> - zigbeeApp item size. • <code>interval</code> - zigbeeApp interval • <code>delay</code> - send the data after this delay. • <code>traceProtocol</code> - specify the type of application <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
<p>APP_TcpServerListen.</p> <p>Listen on a server port.</p>	<p><code>void APP_TcpServerListen. (Node * node, AppType appType, NodeAddress serverAddr, short serverPort)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Node pointer that the protocol is • <code>appType</code> - which application initiates this request • <code>serverAddr</code> - server address • <code>serverPort</code> - server port number <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
<p>APP_TcpServerListen.</p> <p>(Overloaded for IPv6) Listen on a server port.</p>	<p><code>void APP_TcpServerListen. (Node * node, AppType appType, Address serverAddr, short serverPort)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Node pointer that the protocol is • <code>appType</code> - which application initiates this request

	<ul style="list-style-type: none"> • <code>serverAddr</code> - server address • <code>serverPort</code> - server port number <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
APP_TcpServerListenWithPriority.	<p>Listen on a server port with specified priority.</p> <p><code>void APP_TcpServerListenWithPriority. (Node * node, AppType appType, NodeAddress serverAddr, short serverPort, TosType priority)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Node pointer that the protocol is • <code>appType</code> - which application initiates this request • <code>serverAddr</code> - server address • <code>serverPort</code> - server port number • <code>priority</code> - priority of this data for <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
APP_TcpServerListenWithPriority.	<p>Listen on a server port with specified priority. (Overloaded for IPv6)</p> <p><code>void APP_TcpServerListenWithPriority. (Node * node, AppType appType, Address serverAddr, short serverPort, TosType priority)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Node pointer that the protocol is • <code>appType</code> - which application initiates this request • <code>serverAddr</code> - server address • <code>serverPort</code> - server port number • <code>priority</code> - priority of this data for <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
APP_TcpOpenConnection.	<p>Open a connection.</p> <p><code>void APP_TcpOpenConnection. (Node * node, appType appType, NodeAddress localAddr, short localPort, NodeAddress remoteAddr, short remotePort, int uniqueId, clocktype waitTime)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Node pointer that the protocol is • <code>appType</code> - which application initiates this request

	<ul style="list-style-type: none"> • <code>localAddr</code> - address of the source node. • <code>localPort</code> - port number on the source node. • <code>remoteAddr</code> - address of the remote node. • <code>remotePort</code> - port number on the remote node (server port). • <code>uniqueId</code> - used to determine which client is requesting • <code>waitTime</code> - time until the session starts. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
APP_TcpOpenConnection. (Overloaded for IPv6) Open a connection.	<p><code>void APP_TcpOpenConnection. (Node * node, appType appType, Address localAddr, short localPort, Address remoteAddr, short remotePort, int uniqueId, clocktype waitTime)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Node pointer that the protocol is • <code>appType</code> - which application initiates this request • <code>localAddr</code> - address of the source node. • <code>localPort</code> - port number on the source node. • <code>remoteAddr</code> - address of the remote node. • <code>remotePort</code> - port number on the remote node (server port). • <code>uniqueId</code> - used to determine which client is requesting • <code>waitTime</code> - time until the session starts. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
APP_TcpOpenConnection. Open a connection.	<p><code>void APP_TcpOpenConnection. (Node * node, appType appType, NodeAddress localAddr, short localPort, NodeAddress remoteAddr, short remotePort, int uniqueId, clocktype waitTime, int outgoingInterface)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Node pointer that the protocol is • <code>appType</code> - which application initiates this request • <code>localAddr</code> - address of the source node. • <code>localPort</code> - port number on the source node. • <code>remoteAddr</code> - address of the remote node.

	<ul style="list-style-type: none"> • <code>remotePort</code> - port number on the remote node (server port). • <code>uniqueId</code> - used to determine which client is requesting • <code>waitTime</code> - time until the session starts. • <code>outgoingInterface</code> - User specific outgoing Interface. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
APP_TcpOpenConnection. (Overloaded for IPv6) Open a connection.	<p><code>void APP_TcpOpenConnection. (Node * node, appType appType, Address localAddr, short localPort, Address remoteAddr, short remotePort, int uniqueId, clocktype waitTime, int outgoingInterface)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Node pointer that the protocol is • <code>appType</code> - which application initiates this request • <code>localAddr</code> - address of the source node. • <code>localPort</code> - port number on the source node. • <code>remoteAddr</code> - address of the remote node. • <code>remotePort</code> - port number on the remote node (server port). • <code>uniqueId</code> - used to determine which client is requesting • <code>waitTime</code> - time until the session starts. • <code>outgoingInterface</code> - User specific outgoing Interface. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
APP_TcpOpenConnectionWithPriority. Open a connection with specified priority.	<p><code>void APP_TcpOpenConnectionWithPriority. (Node * node, appType appType, NodeAddress localAddr, short localPort, NodeAddress remoteAddr, short remotePort, int uniqueId, clocktype waitTime, TosType priority)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Node pointer that the protocol is • <code>appType</code> - which application initiates this request • <code>localAddr</code> - address of the source node. • <code>localPort</code> - port number on the source node. • <code>remoteAddr</code> - address of the remote node. • <code>remotePort</code> - port number on the remote node (server port).

	<ul style="list-style-type: none"> <code>uniqueId</code> - used to determine which client is requesting <code>waitTime</code> - time until the session starts. <code>priority</code> - priority of the data. <p>Returns:</p> <ul style="list-style-type: none"> <code>void</code> - None
APP_TcpOpenConnectionWithPriority..	<p>Open a connection with specified priority. (Overloaded for IPv6)</p> <p><code>void APP_TcpOpenConnectionWithPriority.. (Node * node, appType appType, Address localAddr, short localPort, Address remoteAddr, short remotePort, int uniqueId, clocktype waitTime, TosType priority)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <code>node</code> - Node pointer that the protocol is <code>appType</code> - which application initiates this request <code>localAddr</code> - address of the source node. <code>localPort</code> - port number on the source node. <code>remoteAddr</code> - address of the remote node. <code>remotePort</code> - port number on the remote node (server port). <code>uniqueId</code> - used to determine which client is requesting <code>waitTime</code> - time until the session starts. <code>priority</code> - priority of the data. <p>Returns:</p> <ul style="list-style-type: none"> <code>void</code> - None
App_TcpCreateMessage.	<p>Create the message.</p> <p><code>Message * App_TcpCreateMessage. (Node * node, int connId, char * payload, int length, traceProtocolType traceProtocol)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <code>node</code> - Node pointer that the protocol is <code>connId</code> - connection id. <code>payload</code> - data to send. <code>length</code> - length of the data to send. <code>traceProtocol</code> - specify the type of application <p>Returns:</p>

	<p>Message * - message created in the function</p>
APP_TcpSendData. send an application data unit.	<p>Message * APP_TcpSendData. (Node * node, int connId, char * payload, int length, raceProtocolType traceProtocol)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Node pointer that the protocol is • connId - connection id. • payload - data to send. • length - length of the data to send. • traceProtocol - specify the type of application <p>Returns:</p> <ul style="list-style-type: none"> • Message * - None
APP_TcpSendNewHeaderVirtualData. Send header and virtual data using TCP.	<p>Message* APP_TcpSendNewHeaderVirtualData. (Node * node, int connId, char * header, int headerLength, int payloadSize, raceProtocolType traceProtocol)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Node pointer that the protocol is • connId - connection id. • header - header to send. • headerLength - length of the header to send. • payloadSize - size of data to send along with header. • traceProtocol - specify the type of application <p>Returns:</p> <ul style="list-style-type: none"> • Message* - Sent message
APP_TcpCloseConnection Close the connection.	<p>void APP_TcpCloseConnection (Node * node, int connId)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Node pointer that the protocol is • connId - connection id. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
APP_InitMulticastGroupMembershipIfAny	void APP_InitMulticastGroupMembershipIfAny (Node * node, const NodeInput nodeInput)

	<p>Start process of joining multicast group if need to do so.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node - node that is joining a group. • nodeInput - used to access configuration file. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
APP_CheckMulticastByParsingSourceAndDestString	<p>Application input parsing API. Parses the source and destination strings. At the same time validates those strings for multicast address.</p> <p>void APP_CheckMulticastByParsingSourceAndDestString (Node * node, const char * inputString, const char * sourceString, NodeAddress * sourceNodeId, NodeAddress * sourceAddr, const char * destString, NodeAddress * destNodeId, NodeAddress * destAddr, BOOL * isDestMulticast)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - A pointer to Node. • inputString - The input string. • sourceString - The source string. • sourceNodeId - A pointer to NodeAddress. • sourceAddr - A pointer to NodeAddress. • destString - The destination string. • destNodeId - A pointer to NodeAddress. • destAddr - A pointer to NodeAddress. • isDestMulticast - Pointer to multicast checking flag. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
APP_ParsingSourceAndDestString	<p>API to parse the input source and destination strings read from the *.app file. At the same time checks and fills the destination type parameter.</p> <p>void APP_ParsingSourceAndDestString (Node * node, const char * inputString, const char * sourceString, NodeAddress * sourceNodeId, NodeAddress * sourceAddr, const char * destString, NodeAddress * destNodeId, NodeAddress * destAddr, DestinationType * destType)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - A pointer to Node. • inputString - The input string. • sourceString - The source string. • sourceNodeId - A pointer to NodeAddress. • sourceAddr - A pointer to NodeAddress. • destString - The destination string.

	<ul style="list-style-type: none"> • destNodeId - A pointer to NodeAddress. • destAddr - A pointer to NodeAddress. • destType - A pointer to Destinationtype. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
APP_ParsingSourceAndDestString	<p>void APP_ParsingSourceAndDestString (Node * node, const char * inputString, const char * sourceString, NodeId * sourceNodeId, Address * sourceAddr, const char * destString, NodeId * destNodeId, Address * destAddr, DestinationType * destType)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - A pointer to Node. • inputString - The input string. • sourceString - The source string. • sourceNodeId - A pointer to NodeAddress. • sourceAddr - A pointer to NodeAddress. • destString - The destination string. • destNodeId - A pointer to NodeAddress. • destAddr - A pointer to NodeAddress. • destType - A pointer to DestinationType. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
APP_RegisterNewApp	<p>AppInfo* APP_RegisterNewApp (Node* node, AppType appType, void * dataPtr, short myPort)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node that is registering the application. • appType - application type • dataPtr - pointer to the data space for this app • myPort - port number to be inserted in the port table <p>Returns:</p> <ul style="list-style-type: none"> • AppInfo* - pointer to the new AppInfo data structure

APP_IsFreePort	<p>BOOL APP_IsFreePort (Node* node, short portNumber)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node that is checking it's port table • portNumber - port number to check <p>Returns:</p> <ul style="list-style-type: none"> • BOOL - indicates if the port is free
APP_GetFreePort	<p>short APP_GetFreePort (Node* node)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node that is requesting a free port <p>Returns:</p> <ul style="list-style-type: none"> • short - returns a free port
APP_InserInPortTable	<p>void APP_InserInPortTable (Node* node, AppType appType, short myPort)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node that needs to be insert in port table • appType - application running at the port • myPort - port number to check <p>Returns:</p> <ul style="list-style-type: none"> • void - None
APP_GetProtocolType	<p>unsigned short APP_GetProtocolType (Node* node, Message* msg)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node that received the message • msg - pointer to the message received <p>Returns:</p> <ul style="list-style-type: none"> • unsigned short - protocol which will receive the message
APP_AssignTos	<p>BOOL APP_AssignTos (char array tosString, char array tosValString, unsigned * tosVal)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • tosString - The tos string. • tosValString - The tos value string.
Application input parsing API. Parses the tos string and tos value strings. At the same time validates those strings for proper ranges.	

	<ul style="list-style-type: none"> • <code>tosVal</code> - A pointer to equivalent 8-bit TOS value. <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - None
APP_UdpCreateNewHeaderVirtualDataWithPriority	<p>Allocate header + virtual data with specified priority and send to UDP. Generally used with messenger app.</p> <p><code>Message* APP_UdpCreateNewHeaderVirtualDataWithPriority (Node * node, AppType appType, NodeAddress sourceAddr, short sourcePort, NodeAddress destAddr, char* header, int headerSize, int payloadSize, TosType priority, TraceProtocolType traceProtocol)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node that is sending the data. • <code>appType</code> - type of application data • <code>sourceAddr</code> - the source sending the data. • <code>sourcePort</code> - the application source port. • <code>destAddr</code> - the destination node Id data • <code>header</code> - header of the payload. • <code>headerSize</code> - size of the header. • <code>payloadSize</code> - size of the data in bytes. • <code>priority</code> - priority of data. • <code>traceProtocol</code> - specify the type of application <p>Returns:</p> <ul style="list-style-type: none"> • <code>Message*</code> - Pointer to allocated message structure
APP_UdpCreateNewHeaderVirtualDataWithPriority	<p>Allocate header + virtual data with specified priority and send to UDP. Generally used with messenger app.</p> <p><code>Message* APP_UdpCreateNewHeaderVirtualDataWithPriority (Node * node, AppType appType, Address sourceAddr, short sourcePort, Address destAddr, char* header, int headerSize, int payloadSize, TosType priority, TraceProtocolType traceProtocol)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node that is sending the data. • <code>appType</code> - type of application data • <code>sourceAddr</code> - the source sending the data. • <code>sourcePort</code> - the application source port. • <code>destAddr</code> - the destination node Id data • <code>header</code> - header of the payload.

	<p><code>headerSize</code> - size of the header.</p> <ul style="list-style-type: none"> • <code>payloadSize</code> - size of the data in bytes. • <code>priority</code> - priority of data. • <code>traceProtocol</code> - specify the type of application <p>Returns:</p> <ul style="list-style-type: none"> • <code>Message*</code> - Pointer to allocated message structure
APP_UnregisterApp	<p><code>void APP_UnregisterApp (Node* node, void * dataPtr, bool freeData)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node that is unregistering the application. • <code>dataPtr</code> - pointer to the data space for this app. • <code>freeData</code> - if true, free (via <code>MEM_free</code>) the dataPtr <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
APP_UnregisterApp	<p><code>void APP_UnregisterApp (Node* node, AppType appType, void * dataPtr, short myPort)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node that is registering the application. • <code>appType</code> - application type • <code>dataPtr</code> - pointer to the data space for this app • <code>myPort</code> - port number to be inserted in the port table <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
APP_IsFreePort	<p><code>BOOL APP_IsFreePort (Node* node, short portNumber)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node that is checking it's port table. • <code>portNumber</code> - port number to check. <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - indicates if the port is free.
APP_RemoveFromPortTable	<code>void APP_RemoveFromPortTable (Node* node, short myPort)</code>

	<p>Parameters:</p> <ul style="list-style-type: none"> • node - node that needs to be remove from port table • myPort - port number to check <p>Returns:</p> <ul style="list-style-type: none"> • void - None
APP_ReportStatsDbReceiveEvent	<p>SequenceNumber APP_ReportStatsDbReceiveEvent (Node* node, Message* msg, SequenceNumber** seqCache, Int64 seqNo, clocktype delay, clocktype jitter, int size, int numRcvd, AppMsgStatus msgStatus)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to a node who receives the msg • msg - The received message or fragment • seqCache - Pointer to the sequence number cache • seqNo - Sequence number of the message or fragment • delay - Delay of the message/fragment • jitter - Smoothed jitter of the received message • size - Size of msg/fragment to be report to db • numRcvd - # of msgs/frags received so far • msgStatus - This is for performance optimization. If <p>Returns:</p> <ul style="list-style-type: none"> • SequenceNumber - Status or out of order or new

Contact [QualNet Support](#) for questions pertaining to the QualNet API Reference. This document is confidential and proprietary.
It may not be reproduced or distributed without the expressed written consent of [Scalable Network Technologies](#).



QualNet® is a Registered Trademark of [Scalable Network Technologies](#).

Copyright © 2001-2012 [Scalable Network Technologies, Inc.](#). All rights reserved.

QualNet 6.1 API Reference

APPLICATION LAYER

This file describes data structures and functions used by the Application Layer.

Constant / Data Structure Summary

Type	Name
CONSTANT	APP_DEFAULT_TOS
	Application default tos value
CONSTANT	APP_MAX_DATA_SIZE
	Maximum size of data unit
CONSTANT	DEFAULT_APP_QUEUE_SIZE
	Default size of Application layer queue (in byte)
CONSTANT	PORT_TABLE_HASH_SIZE
	Prime number indicating port table size
CONSTANT	MAC_LINK16_FRAG_SIZE
	Maximum fragment size supported by LINK16 MAC protocol. For Link16, it seems the fragment size should be $8 * 9$ bytes = 72 bytes
CONSTANT	MAC_DEFAULT_INTERFACE
	Default interface of MAC layer. ASSUMPTION :: Source and Destination node must have only one interface with TADIL network.
ENUMERATION	AppType
	Enumerates the type of application protocol
STRUCT	Link16GatewayData
	Store Link16/IP gateway forwarding table
STRUCT	AppInfo

	Information relevant to specific app layer protocol PortInfo
STRUCT	Store port related information AppMultimedia
STRUCT	Store multimedia signalling related information AppData
STRUCT	Details of application data structure in node structure AppTimer
	Timer structure used by applications

Function / Macro Summary

Return Type	Summary
void	InitiateConnectionType (Node* node, void* voip) Multimedia callback function to open request for a TCP connection from the initiating terminal
void	TerminateConnectionType (Node* node, void* voip) Multimedia callback function to close TCP connection as requested by VOIP application
BOOL	IsHostCallingType (Node* node) Multimedia callback function to check whether node is in initiator mode
BOOL	IsHostCalledType (Node* node) Multimedia callback function to check whether node is in receiver mode

Constant / Data Structure Detail

Constant	APP_DEFAULT_TOS 0x00 Application default tos value
Constant	APP_MAX_DATA_SIZE IP_MAXPACKET-MSG_MAX_HDR_SIZE Maximum size of data unit
Constant	DEFAULT_APP_QUEUE_SIZE 640000 Default size of Application layer queue (in byte)
Constant	PORT_TABLE_HASH_SIZE 503 Prime number indicating port table size
Constant	MAC_LINK16_FRAG_SIZE 72 Maximum fragment size supported by LINK16 MAC protocol. For Link16, it seems the fragment size should be 8 * 9 bytes = 72 bytes
Constant	MAC_DEFAULT_INTERFACE 0 Default interface of MAC layer. ASSUMPTION :: Source and Destination node must have only one interface with TADIL network.
Enumeration	AppType Enumerates the type of application protocol
Structure	Link16GatewayData Store Link16/IP gateway forwarding table
Structure	AppInfo Information relevant to specific app layer protocol
Structure	PortInfo Store port related information

Structure	AppMultimedia Store multimedia signalling related information
Structure	AppData Details of application data structure in node structure
Structure	AppTimer Timer structure used by applications

Function / Macro Detail

Function / Macro	Format
InitiateConnectionType Multimedia callback function to open request for a TCP connection from the initiating terminal	void InitiateConnectionType (Node* node, void* voip) Parameters: <ul style="list-style-type: none">• node - Pointer to the node• voip - Pointer to the voip application Returns: <ul style="list-style-type: none">• void - NULL
TerminateConnectionType Multimedia callback function to close TCP connection as requested by VOIP application	void TerminateConnectionType (Node* node, void* voip) Parameters: <ul style="list-style-type: none">• node - Pointer to the node• voip - Pointer to the voip application Returns: <ul style="list-style-type: none">• void - NULL
IsHostCallingType Multimedia callback function to check whether node is in initiator mode	BOOL IsHostCallingType (Node* node) Parameters: <ul style="list-style-type: none">• node - Pointer to the node Returns:

	<p>BOOL - TRUE if the node is initiator, FALSE otherwise</p>
IsHostCalledType Multimedia callback function to check whether node is in receiver mode	<p>BOOL IsHostCalledType (Node* node)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to the node <p>Returns:</p> <ul style="list-style-type: none"> • BOOL - TRUE if the node is receiver, FALSE otherwise



Contact [QualNet Support](#) for questions pertaining to the QualNet API Reference. This document is confidential and proprietary.
It may not be reproduced or distributed without the expressed written consent of [Scalable Network Technologies](#).

QualNet® is a Registered Trademark of [Scalable Network Technologies](#).

Copyright © 2001-2012 [Scalable Network Technologies, Inc.](#) All rights reserved.

QualNet 6.1 API Reference

BUFFER

This file describes data structures and functions to implement buffers.

Constant / Data Structure Summary

Type	Name
STRUCT	DataBuffer structure for the data buffer
STRUCT	ReassemblyBuffer Format for the Reassembly buffer
STRUCT	PacketBuffer structure for the packet buffer

Function / Macro Summary

Return Type	Summary
MACRO	BUFFER_GetCurrentSize(x) Returns the current size of the buffer
MACRO	BUFFER_GetMaxSize(x) Returns maximum allowable size of the buffer
MACRO	BUFFER_GetData(x) Returns a pointer to the data in the buffer
MACRO	BUFFER_GetAnticipatedSize(x)

	Returns the initial size of the buffer
MACRO	BUFFER_SetCurrentSize(x,y)
	Sets current size of the buffer
MACRO	BUFFER_GetFreeSpace(x)
	Get free space available in the buffer
MACRO	BUFFER_ReturnTop(x)
	Returns top of the buffer
MACRO	BUFFER_NumberOfBlocks(X)
	Returns the no. of blocks in the buffer
void	BUFFER_InitializeDataBuffer(DataBuffer* buffer, int size)
	Initializing Data buffer. Keeping in mind that buffer will be initialized once and the guess for initial size is a good one. For all the other manipulation of the buffer will try to allocate in the initial if not asked to increase size and this size will remain until end of program for re-using unless the buffer is closed completely.
void	BUFFER_AddSpaceToDataBuffer(DataBuffer* buffer, int size)
	Adding memory space to the buffer
void	BUFFER_ClearDataFromDataBuffer(DataBuffer* buffer, char * startLocation, int size, BOOL destroy)
	clear data from the buffer(already used portion of buffer Not any unused portion unless u clear till end)
void	BUFFER_DestroyDataBuffer(DataBuffer* buffer)
	To Destroy a buffer
void	BUFFER_AddDataToDataBuffer(DataBuffer* buffer, char * data, int size)
	Add data to databuffer
void	BUFFER_RemoveDataFromDataBuffer(DataBuffer* buffer, char* startLocation, int size)
	To remove data from the data buffer
void	InitializeReassemblyBuffer(ReassemblyBuffer* buffer, int size)

	Initialize Reassembly buffer
void	BUFFER_AddDataToAssemblyBuffer (ReassemblyBuffer* buffer, char* data, int size, BOOL allowOverflow)
	Appending data to the reassembly buffer
void	BUFFER_ClearAssemblyBuffer (ReassemblyBuffer* buffer, int max, BOOL setSize)
	clear the buffer
void	BUFFER_SetAnticipatedSizeForAssemblyBuffer (ReassemblyBuffer* buffer, int size)
	To set the anticipated size of the assemblyBuffer
PacketBuffer *	BUFFER_AllocatePacketBuffer (int initialSize, int anticipatedHeaderMax, BOOL allowOverflow, char ** dataPtr)
	To allocate packet buffer
PacketBuffer *	BUFFER_AllocatePacketBufferWithInitialHeader (int initialSize, int initialHeaderSize, int anticipatedHeaderMax, BOOL allowOverflow, char ** dataPtr, char ** headerPtr)
	To allocate buffer with Intial header
void	BUFFER_AddHeaderToPacketBuffer (PacketBuffer* buffer, int headerSize, char** headerPtr)
	To add header to buffer
void	BUFFER_RemoveHeaderFromPacketBuffer (PacketBuffer* buffer, int headerSize, char** dataPtr)
	To remove header from packet buffer
void	BUFFER_ClearPacketBufferData (PacketBuffer* buffer)
	To clear data from current buffer
void	BUFFER_FreePacketBuffer (PacketBuffer* buffer)
	Free the packet buffer passed as argument
void	BUFFER_ConcatenatePacketBuffer (const PacketBuffer* source, PacketBuffer* dest)
	Add useful contents of source buffer as header to to the destination buffer

Constant / Data Structure Detail

Structure	DataBuffer structure for the data buffer
Structure	ReassemblyBuffer Format for the Reassembly buffer
Structure	PacketBuffer structure for the packet buffer

Function / Macro Detail

Function / Macro	Format
BUFFER_GetCurrentSize(x)	Returns the current size of the buffer
BUFFER_GetMaxSize(x)	Returns maximum allowable size of the buffer
BUFFER_GetData(x)	Returns a pointer to the data in the buffer
BUFFER_GetAnticipatedSize(x)	Returns the intial size of the buffer
BUFFER_SetCurrentSize(x,y)	Sets current size of the buffer
BUFFER_GetFreeSpace(x)	Get free space available in the buffer
BUFFER_ReturnTop(x)	Returns top of the buffer
BUFFER_NumberOfBlocks(X)	Returns the no. of blocks in the buffer
BUFFER_InitializeDataBuffer	void BUFFER_InitializeDataBuffer (DataBuffer* buffer, int size) Parameters:

<p>Initializing Data buffer. Keeping in mind that buffer will be initialized once and the guess for initial size is a good one. For all the other manipulation of the buffer will try to allocate in the initial if not asked to increase size and this size will remain until end of program for re-using unless the buffer is closed completely.</p>	<ul style="list-style-type: none"> • <code>buffer</code> - buffer to be initialized • <code>size</code> - intial size of the buffer <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
<p>BUFFER_AddSpaceToDataBuffer</p> <p>Adding memory space to the buffer</p>	<p>void BUFFER_AddSpaceToDataBuffer (DataBuffer* buffer, int size)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>buffer</code> - buffer to which to add space • <code>size</code> - size to be added <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
<p>BUFFER_ClearDataFromDataBuffer</p> <p>clear data from the buffer(already used portion of buffer Not any unused portion unless u clear till end)</p>	<p>void BUFFER_ClearDataFromDataBuffer (DataBuffer* buffer, char * startLocation, int size, BOOL destroy)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>buffer</code> - buffer from which data is cleared • <code>startLocation</code> - starting location • <code>size</code> - intial size of the buffer • <code>destroy</code> - destroy or not <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
<p>BUFFER_DestroyDataBuffer</p> <p>To Destroy a buffer</p>	<p>void BUFFER_DestroyDataBuffer (DataBuffer* buffer)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>buffer</code> - buffer to be destroyed <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
<p>BUFFER_AddDataToDataBuffer</p> <p>Add data to databuffer</p>	<p>void BUFFER_AddDataToDataBuffer (DataBuffer* buffer, char * data, int size)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>buffer</code> - buffer to which data is added • <code>data</code> - pointer to data that is added

	<ul style="list-style-type: none"> • <code>size</code> - initial size of the buffer <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
BUFFER_RemoveDataFromDataBuffer	<p>To remove data from the data buffer</p> <p><code>void BUFFER_RemoveDataFromDataBuffer (DataBuffer* buffer, char* startLocation, int size)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>buffer</code> - buffer from which data is to be removed • <code>startLocation</code> - starting location from whcih data is removed • <code>size</code> - size of the buffer <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
InitializeReassemblyBuffer	<p>Initialize Reassembly buffer</p> <p><code>void InitializeReassemblyBuffer (ReassemblyBuffer* buffer, int size)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>buffer</code> - ReassemblyBuffer to be initialized • <code>size</code> - maximum allowable size of the buffer <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
BUFFER_AddDataToAssemblyBuffer	<p>Appending data to the reassembly buffer</p> <p><code>void BUFFER_AddDataToAssemblyBuffer (ReassemblyBuffer* buffer, char* data, int size, BOOL allowOverflow)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>buffer</code> - Pointer to ReassemblyBuffer • <code>data</code> - data to be added • <code>size</code> - size of the data • <code>allowOverflow</code> - To allow overflow or not <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
BUFFER_ClearAssemblyBuffer	<p>clear the buffer</p> <p><code>void BUFFER_ClearAssemblyBuffer (ReassemblyBuffer* buffer, int max, BOOL setSize)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>buffer</code> - Pointer to ReassemblyBuffer • <code>max</code> - the maximum size you want to set, if <code>setSize</code> is TRUE

	<ul style="list-style-type: none"> • <code>setSize</code> - TRUE, if the buffer max-size is to be re-set <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
BUFFER_SetAnticipatedSizeForAssemblyBuffer	<p>To set the anticipated size of the assemblyBuffer</p> <p><code>void BUFFER_SetAnticipatedSizeForAssemblyBuffer (ReassemblyBuffer* buffer, int size)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>buffer</code> - Pointer to ReassemblyBuffer • <code>size</code> - size to be set <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
BUFFER_AllocatePacketBuffer	<p>To allocate packet buffer</p> <p><code>PacketBuffer * BUFFER_AllocatePacketBuffer (int initialSize, int anticipatedHeaderMax, BOOL allowOverflow, char ** dataPtr)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>initialSize</code> - intial size of buffer • <code>anticipatedHeaderMax</code> - expected max header size • <code>allowOverflow</code> - if overflow is allowed • <code>dataPtr</code> - pointer to data array <p>Returns:</p> <ul style="list-style-type: none"> • <code>PacketBuffer * -</code> Pointer to packetbuffer
BUFFER_AllocatePacketBufferWithInitialHeader	<p>To allocate buffer with Intial header</p> <p><code>PacketBuffer * BUFFER_AllocatePacketBufferWithInitialHeader (int initialSize, int initialHeaderSize, int anticipatedHeaderMax, BOOL allowOverflow, char ** dataPtr, char ** headerPtr)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>initialSize</code> - intial buffer size • <code>initialHeaderSize</code> - initial header size • <code>anticipatedHeaderMax</code> - expected max header size • <code>allowOverflow</code> - if overflow is allowed • <code>dataPtr</code> - pointer to array • <code>headerPtr</code> - pointer to array <p>Returns:</p>

	<ul style="list-style-type: none"> • <code>PacketBuffer * -</code> Pointer to packetbuffer
BUFFER_AddHeaderToPacketBuffer	<p>void BUFFER_AddHeaderToPacketBuffer (<code>PacketBuffer* buffer, int headerSize, char** headerPtr</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>buffer -</code> Pointer to PacketBuffer • <code>headerSize -</code> size of header • <code>headerPtr -</code> Pointer to an array of strings <p>Returns:</p> <ul style="list-style-type: none"> • <code>void -</code> None
BUFFER_RemoveHeaderFromPacketBuffer	<p>void BUFFER_RemoveHeaderFromPacketBuffer (<code>PacketBuffer* buffer, int headerSize, char** dataPtr</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>buffer -</code> Pointer to PacketBuffer • <code>headerSize -</code> size of header • <code>dataPtr -</code> Pointer to an strings array <p>Returns:</p> <ul style="list-style-type: none"> • <code>void -</code> None
BUFFER_ClearPacketBufferData	<p>void BUFFER_ClearPacketBufferData (<code>PacketBuffer* buffer</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>buffer -</code> Pointer to PacketBuffer <p>Returns:</p> <ul style="list-style-type: none"> • <code>void -</code> None
BUFFER_FreePacketBuffer	<p>void BUFFER_FreePacketBuffer (<code>PacketBuffer* buffer</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>buffer -</code> Pointer to PacketBuffer <p>Returns:</p> <ul style="list-style-type: none"> • <code>void -</code> None
BUFFER_ConcatenatePacketBuffer	<p>void BUFFER_ConcatenatePacketBuffer (<code>const PacketBuffer* source, PacketBuffer* dest</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>source -</code> Pointer to PacketBuffer

the destination buffer

- `dest` - Pointer to PacketBuffer

Returns:

- `void` - None

Contact [QualNet Support](#) for questions pertaining to the QualNet API Reference. This document is confidential and proprietary.
It may not be reproduced or distributed without the expressed written consent of [Scalable Network Technologies](#).



[QualNet](#)® is a Registered Trademark of [Scalable Network Technologies](#).

Copyright © 2001-2012 [Scalable Network Technologies, Inc.](#). All rights reserved.

QualNet 6.1 API Reference

CIRCULAR-BUFFER

This file describes data structures and functions used for circular buffer implementation.

Constant / Data Structure Summary

Type	Name
CONSTANT	CIR_BUF_SIZE Default Circular Buffer Size
ENUMERATION	Type of Circular Buffer Operation
ENUMERATION	Type of Wrap operation

Function / Macro Summary

Return Type	Summary
bool	CircularBuffer.incPos (Int32 increment, Int32 operation) increment read/write position based on operation
None	CircularBuffer.CircularBuffer () Constructor
None	CircularBuffer.CircularBuffer (Int32 queueSize) Constructor
None	CircularBuffer.CircularBuffer (unsigned short index) Constructor
None	CircularBuffer.CircularBuffer (unsigned short index, Int32 queueSize)

	Constructor
Node	<code>CircularBuffer::~CircularBuffer()</code>
	Destructor
bool	<code>CircularBuffer.create(Int32 queueSize)</code>
	Memory allocation for Circular Buffer
void	<code>CircularBuffer.release(void)</code>
	To free the allocated memory
void	<code>CircularBuffer.reset(void)</code>
	reset position and wrap values
bool	<code>CircularBuffer.getCount(Int32& count, Int32 operation)</code>
	gets the number of bytes to read
Int32	<code>CircularBuffer.lengthToEnd(Int32 operation)</code>
	get the circular buffer's allocated size
bool	<code>CircularBuffer.readWithCount(unsigned char* data, Int32& length)</code>
	Read data from Buffer and pass the length of data read
bool	<code>CircularBuffer.readFromBuffer(unsigned char* data, Int32 length, bool noIncrement)</code>
	Reading the required no. of bytes from the circular buffer
bool	<code>CircularBuffer.write(unsigned char* data, Int32 length)</code>
	Write to the circular buffer
bool	<code>CircularBuffer.read(unsigned char* buffer)</code>
	To Read data from Buffer
Int32	<code>CircularBuffer.getIndex(Int32 operation)</code>

	get the circular buffer's allocated size <code>CircularBuffer.getBufSize(void none)</code>
Int32	get the circular buffer's allocated size <code>CircularBuffer.getIndex(void none)</code>

Constant / Data Structure Detail

Constant	CIR_BUF_SIZE 256 Default Circular Buffer Size
Enumeration	Type of Circular Buffer Operation
Enumeration	Type of Wrap operation

Function / Macro Detail

Function / Macro	Format
<code>CircularBuffer.incPos</code> increment read/write position based on operation	bool <code>CircularBuffer.incPos</code> (Int32 increment, Int32 operation) Parameters: <ul style="list-style-type: none">• increment - How much will be incremented• operation - Type of Operation (Read or Write) Returns: <ul style="list-style-type: none">• bool - Successful or not
<code>CircularBuffer.CircularBuffer</code> Constructor	None <code>CircularBuffer.CircularBuffer()</code> Parameters: Returns:

	<ul style="list-style-type: none"> • None - None
CircularBuffer.CircularBuffer Constructor	<p>None CircularBuffer.CircularBuffer (Int32 queueSize)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • queueSize - Size of the Queue <p>Returns:</p> <ul style="list-style-type: none"> • None - None
CircularBuffer.CircularBuffer Constructor	<p>None CircularBuffer.CircularBuffer (unsigned short index)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • index - Circular Buffer Index <p>Returns:</p> <ul style="list-style-type: none"> • None - None
CircularBuffer.CircularBuffer Constructor	<p>None CircularBuffer.CircularBuffer (unsigned short index, Int32 queueSize)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • index - Circular Buffer Index • queueSize - Size of the queue <p>Returns:</p> <ul style="list-style-type: none"> • None - None
CircularBuffer.~CircularBuffer Destructor	<p>Node CircularBuffer.~CircularBuffer ()</p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> • Node - None
CircularBuffer.create Memory allocation for Circular Buffer	<p>bool CircularBuffer.create (Int32 queueSize)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • queueSize - Size of queue <p>Returns:</p> <ul style="list-style-type: none"> • bool - Successful or not
CircularBuffer.release	void CircularBuffer.release (void)

	<p>To free the allocated memory</p> <p>Parameters:</p> <ul style="list-style-type: none"> • - None <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
<p>CircularBuffer.reset</p> <p>reset position and wrap values</p>	<p><code>void CircularBuffer.reset (void)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • - None <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
<p>CircularBuffer.getCount</p> <p>gets the number of bytes to read</p>	<p><code>bool CircularBuffer.getCount (Int32& count, Int32 operation)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>count</code> - the parameter to be filled up • <code>operation</code> - Type of Operation (Read or Write) <p>Returns:</p> <ul style="list-style-type: none"> • <code>bool</code> - successful or not
<p>CircularBuffer.lengthToEnd</p> <p>get the circular buffer's allocated size</p>	<p><code>Int32 CircularBuffer.lengthToEnd (Int32 operation)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>operation</code> - Read or Write Operation <p>Returns:</p> <ul style="list-style-type: none"> • <code>Int32</code> - Total length of data to be read
<p>CircularBuffer.readWithCount</p> <p>Read data from Buffer and pass the length of data read</p>	<p><code>bool CircularBuffer.readWithCount (unsigned char* data, Int32& length)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>data</code> - Container to which data will be read • <code>length</code> - length of data read <p>Returns:</p> <ul style="list-style-type: none"> • <code>bool</code> - Successful or not
<p>CircularBuffer.readFromBuffer</p>	<p><code>bool CircularBuffer.readFromBuffer (unsigned char* data, Int32 length, bool noIncrement)</code></p> <p>Parameters:</p>

<p>Reading the required no. of bytes from the circular buffer</p>	<ul style="list-style-type: none"> • <code>data</code> - Container to which data will be read • <code>length</code> - length of data to be read • <code>noIncrement</code> - Whether the read pointer is to be incremented or not <p>Returns:</p> <ul style="list-style-type: none"> • <code>bool</code> - successful or not
<p>CircularBuffer.write</p> <p>Write to the circular buffer</p>	<p><code>bool CircularBuffer.write (unsigned char* data, Int32 length)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>data</code> - Container to which data will be written • <code>length</code> - Length of data to be written <p>Returns:</p> <ul style="list-style-type: none"> • <code>bool</code> - successful or not
<p>CircularBuffer.read</p> <p>To Read data from Buffer</p>	<p><code>bool CircularBuffer.read (unsigned char* buffer)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>buffer</code> - Container to which data will be read <p>Returns:</p> <ul style="list-style-type: none"> • <code>bool</code> - Succesful or not
<p>CircularBuffer.getIndex</p> <p>get the circular buffer's allocated size</p>	<p><code>Int32 CircularBuffer.getIndex (Int32 operation)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>operation</code> - Read or Write Operation <p>Returns:</p> <ul style="list-style-type: none"> • <code>Int32</code> - Current operation Position
<p>CircularBuffer.getBufSize</p> <p>get the circular buffer's allocated size</p>	<p><code>Int32 CircularBuffer.getBufSize (void none)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>none</code> - None <p>Returns:</p> <ul style="list-style-type: none"> • <code>Int32</code> - circular buffer's allocated size
<p>CircularBuffer.getIndex</p>	<p><code>unsigned short CircularBuffer.getIndex (void none)</code></p>

get the circular buffer's unique index

Parameters:

- none - None

Returns:

- unsigned short - unique index

Contact [QualNet Support](#) for questions pertaining to the QualNet API Reference. This document is confidential and proprietary.
It may not be reproduced or distributed without the expressed written consent of [Scalable Network Technologies](#).



QualNet® is a Registered Trademark of [Scalable Network Technologies](#).

Copyright © 2001-2012 [Scalable Network Technologies, Inc.](#) All rights reserved.

QualNet 6.1 API Reference

CLOCK

This file describes data structures and functions used for time-related operations.

Constant / Data Structure Summary

Type	Name
CONSTANT	CLOCKTYPE_MAX
	CLOCKTYPE_MAX is the maximum value of clocktype. This value can be anything as long as it is less than or equal to the maximum value of the type which is typedefed to clocktype. Users can simulate the model up to CLOCKTYPE_MAX - 1.
CONSTANT	NANO_SECOND
	Defined as basic unit of clocktype
CONSTANT	MICRO_SECOND
	Defined as 1000 times the basic unit of clocktype
CONSTANT	MILLI_SECOND
	unit of time equal to 1000 times MICRO_SECOND
CONSTANT	SECOND
	simulation unit of time =1000 times MILLI_SECOND
CONSTANT	MINUTE
	unit of simulation time = 60 times SECOND
CONSTANT	HOUR
	unit of simulation time = 60 times MINUTE
CONSTANT	DAY
	unit of simulation time = 24 times HOUR

CONSTANT	PROCESS IMMEDIATELY
	Used to prioritize a process

Function / Macro Summary

Return Type	Summary
MACRO	ctoa like sprintf, prints a clocktype to a string
MACRO	atoc like atoi or atof, converts a string to a clocktype
MACRO	getSimTime(node) To get the simulation time of a node
MACRO	getSimStartTime(node) To get the simulation start time of a node
MACRO	TIME getSimTime(node) Gets current simulation time of a node
clocktype	TIME ConvertToClock(char* buf) Read the string in "buf" and provide the corresponding clocktype value for the string using the following conversions: NS - nano-seconds MS - milli-seconds S - seconds (default if no specification) H - hours D - days
void	TIME PrintClockInSecond(clocktype clock, char * stringInSecond) Print a clocktype value in second.The result is copied to string in Seconds
void	TIME PrintClockInSecond(clocktype clock, char * stringInSecond, Node * node) Print a clocktype value in second.The result is copied to string in Seconds
void	TIME PrintClockInSecond(clocktype clock, char * stringInSecond, PartitionData * partition) Print a clocktype value in second.The result is copied to string in Seconds

	Print a clocktype value in second.The result is copied to string in Seconds <code>TIME_ReturnMaxSimClock(Node* node)</code>
clocktype	Return the maximum simulation clock <code>TIME_ReturnStartSimClock(Node* node)</code>

Constant / Data Structure Detail

Constant	CLOCKTYPE_MAX Platform dependent CLOCKTYPE_MAX is the maximum value of clocktype. This value can be anything as long as it is less than or equal to the maximum value of the type which is typedefed to clocktype. Users can simulate the model up to CLOCKTYPE_MAX - 1.
Constant	NANO_SECOND ((clocktype) 1) Defined as basic unit of clocktype
Constant	MICRO_SECOND (1000 * NANO_SECOND) Defined as 1000 times the basic unit of clocktype
Constant	MILLI_SECOND (1000 * MICRO_SECOND) unit of time equal to 1000 times MICRO_SECOND
Constant	SECOND (1000 * MILLI_SECOND) simulation unit of time =1000 times MILLI_SECOND
Constant	MINUTE (60 * SECOND) unit of simulation time = 60 times SECOND
Constant	HOUR (60 * MINUTE)

	unit of simulation time = 60 times MINUTE
Constant	DAY (24 * HOUR) unit of simulation time = 24 times HOUR
Constant	PROCESS_IMMEDIATELY 0 Used to prioritize a process

Function / Macro Detail

Function / Macro	Format
ctoa	like sprintf, prints a clocktype to a string
atoc	like atoi or atof, converts a string to a clocktype
getSimTime(node)	To get the simulation time of a node
getSimStartTime(node)	To get the simulation start time of a node
TIME_getSimTime(node)	Gets current simulation time of a node
TIME_ConvertToClock Read the string in "buf" and provide the corresponding clocktype value for the string using the following conversions: NS - nano-seconds MS - milli-seconds S - seconds (default if no specification) H - hours D - days	clocktype TIME_ConvertToClock (char* buf) Parameters: <ul style="list-style-type: none">• buf - The time string Returns: <ul style="list-style-type: none">• clocktype - Time in clocktype
TIME_PrintClockInSecond Print a clocktype value in second.The result is copied to string in Seconds	void TIME_PrintClockInSecond (clocktype clock , char * stringInSecond) Parameters: <ul style="list-style-type: none">• clock - Time in clocktype• stringInSecond - string containing time in seconds Returns:

	<ul style="list-style-type: none"> • void - None
TIME_PrintClockInSecond	<p>Print a clocktype value in second.The result is copied to string in Seconds</p> <p>void TIME_PrintClockInSecond (clocktype clock, char * stringInSecond, Node * node)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • clock - Time in clocktype • stringInSecond - string containing time in seconds • node - Input node <p>Returns:</p> <ul style="list-style-type: none"> • void - None
TIME_PrintClockInSecond	<p>Print a clocktype value in second.The result is copied to string in Seconds</p> <p>void TIME_PrintClockInSecond (clocktype clock, char * stringInSecond, PartitionData * partition)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • clock - Time in clocktype • stringInSecond - string containing time in seconds • partition - Input partition <p>Returns:</p> <ul style="list-style-type: none"> • void - None
TIME_ReturnMaxSimClock	<p>Return the maximum simulation clock</p> <p>clocktype TIME_ReturnMaxSimClock (Node* node)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Input node <p>Returns:</p> <ul style="list-style-type: none"> • clocktype - Returns maximum simulation time
TIME_ReturnStartSimClock	<p>Return the simulation start clock</p> <p>clocktype TIME_ReturnStartSimClock (Node* node)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Input node <p>Returns:</p> <ul style="list-style-type: none"> • clocktype - Returns simulation start time



Contact [QualNet Support](#) for questions pertaining to the QualNet API Reference. This document is confidential and proprietary.
It may not be reproduced or distributed without the expressed written consent of [Scalable Network Technologies](#).

[QualNet®](#) is a Registered Trademark of [Scalable Network Technologies](#).

Copyright © 2001-2012 [Scalable Network Technologies, Inc.](#). All rights reserved.

QualNet 6.1 API Reference

COORDINATES

This file describes data structures and functions used for coordinates-related operations.

Constant / Data Structure Summary

Type	Name
CONSTANT	PI Defines the value of constant PI
CONSTANT	ANGLE_RESOLUTION Defines ANGLE_RESOLUTION
CONSTANT	IN_RADIAN Defines the constant IN_RADIAN
CONSTANT	EARTH_RADIUS Defines the above constant EARTH_RADIUS
ENUMERATION	EarthRepresentationType Defines the type of Earth that is represented Replaces coordinate_system_type
ENUMERATION	CoordinateRepresentationType Defines the coordinate system that a coordinate is given in reference to
ENUMERATION	coordinate_system_type Defines the type of coordinate system
STRUCT	cartesian_coordinates Defines the three cartesian coordinates
STRUCT	latlonalt_coordinates

	Defines the three latlonalt coordinates common_coordinates
STRUCT	Defines the three common coordinates Coordinates
STRUCT	Defines coordinates Orientation
STRUCT	Defines the orientation structure

Function / Macro Summary

Return Type	Summary
MACRO	MAX(X, Y) Finds the maximum of two entries
MACRO	MIN(X, Y) Finds the minimum of two entries
MACRO	COORD_ShortestPropagationDelay(dist) Calculate the shortest propagation delay. Shortest delay is assumed with light speed. Actual delay could be longer if propagation medium is not electromagnatic waves, such as acoustic wave.
BOOL	COORD_CoordinatesAreTheSame(const Coordinates c1, const Coordinates c2) To compare two coordinates and determine if they are the same
BOOL	COORD_OrientationsAreTheSame(const Orientation o1, const Orientation o2) To compare two coordinates and determine if they have the same orientation
static int	COORD_NormalizeAzimuthAngle(int angle)

COORDINATES

	To normalize the azimuth angle COORD_NormalizeElevationAngle(int angle)
static int	To normalize the elevation angle COORD_NormalizeAngleIndex(int angleIndex, int angleResolution)
static int	To normalize the angleIndex COORD_CalcDistance(int coordinateSystemType, const Coordinates* position1, const Coordinates* position2, CoordinateType distance)
BOOL	To calculate the distance between two nodes(points) given the coordinateSystemType and the coordinates of the two points COORD_CalcDistanceAndAngle(int coordinateSystemType, const position1, const position2, double* distance, Orientation* DOA1, Orientation* DOA2)
static void	To calculate the Distance and Angle COORD_ChangeCoordinateSystem(const CoordinateRepresentationType source_type, const Coordinates* const source, const CoordinateRepresentationType target_type, Coordinates* const target)
static void	Re-calculate coordinate in a new coordinate system COORD_ChangeCoordinateSystem(const Coordinates* const source, const CoordinateRepresentationType target_type, Coordinates* const target)
static void	Re-calculate coordinate in a new coordinate system COORD_GeodeticToGeocentricCartesian(const Coordinates* const source, Coordinates* const target)
static void	Convert coordinate from GEODETIC to GEOCENTRIC_CARTESIAN COORD_GeocentricCartesianToGeodetic(const Coordinates* const source, Coordinates* const target)
static void	Convert coordinate from GEOCENTRIC_CARTESIAN to GEODETIC COORD_JGISToGeodetic(const Coordinates* const source, Coordinates* const target)
static void	Convert coordinate from JGIS to GEODETIC COORD_JGISToUnreferencedCartesian(const Coordinates* const source, Coordinates* const target)
static void	Convert coordinate from JGIS to UNREFERENCED_CARTESIAN COORD_ConvertToCoordinates(char* buf, Coordinates* coordinates)

	Read the string in "buf" and provide the corresponding coordinates for the string. COORD_MapCoordinateSystemToType (int coordinateSystem, Coordinates* coordinates)
static void	Set coordinates type field (CoordinateRepresentationType) based on the user-provided coordinate system (coordinate_system_type) COORD_NormalizeLongitude (Coordinates* coordinates)
static void	Correct the longitude value to between -180 and 180. This function assumes the coordinate system is LLA.
bool	COORD_PointWithinRange (int coordinateSystemType, Coordinates* sw, Coordinates* ne, Coordinates* point)
bool	Is the point within the given range. Assume -90 <= lat <= 90 and -180 <= long <= 180 for all inputs. COORD_RegionsOverlap (int coordinateSystemType, Coordinates* sw1, Coordinates* ne1, Coordinates* sw2, Coordinates* ne2)
void	Determine whether the given regions overlap at all. COORD_LongitudeDelta (CoordinateType long1, CoordinateType long2)
void	Convenience function for geodetic that, given two longitudes, returns the difference (in degrees) in the shorter direction. COORD_PrintCoordinates (int coordinateSystemType, Coordinates* point)
	Prints the coordinates in a human readable format.

Constant / Data Structure Detail

Constant	PI 3.14159265358979323846264338328 Defines the value of constant PI
Constant	ANGLE_RESOLUTION 360 Defines ANGLE_RESOLUTION
Constant	IN_RADIAN (PI / 180.0)

COORDINATES

	Defines the constant IN_RADIAN
Constant	EARTH_RADIUS 6375000.0 Defines the above constant EARTH_RADIUS
Enumeration	EarthRepresentationType Defines the type of Earth that is represented Replaces coordinate_system_type
Enumeration	CoordinateRepresentationType Defines the coordinate system that a coordinate is given in reference to
Enumeration	coordinate_system_type Defines the type of coordinate system
Structure	cartesian_coordinates Defines the three cartesian coordinates
Structure	latlonalt_coordinates Defines the three latlonalt coordinates
Structure	common_coordinates Defines the three common coordinates
Structure	Coordinates Defines coordinates
Structure	Orientation Defines the orientation structure

Function / Macro Detail

Function / Macro	Format
------------------	--------

MAX(X, Y)	Finds the maximum of two entries
MIN(X, Y)	Finds the minimum of two entries
COORD_ShortestPropagationDelay(dist)	Calculate the shortest propagation delay. Shortest delay is assumed with light speed. Actual delay could be longer if propagation medium is not electromagnetic waves, such as acoustic wave.
COORD_CoordinatesAreTheSame To compare two coordinates and determine if they are the same	BOOL COORD_CoordinatesAreTheSame (const Coordinates c1, const Coordinates c2) Parameters: <ul style="list-style-type: none">• c1 - coordinates of a point• c2 - coordinates of a point Returns: <ul style="list-style-type: none">• BOOL - whether the points are the same
COORD_OrientationsAreTheSame To compare two orientations and determine if they have the same orientation	BOOL COORD_OrientationsAreTheSame (const Orientation o1, const Orientation o2) Parameters: <ul style="list-style-type: none">• o1 - orientation of a point• o2 - orientation of a point Returns: <ul style="list-style-type: none">• BOOL - whether the points have the same orientation
COORD_NormalizeAzimuthAngle To normalize the azimuth angle	static int COORD_NormalizeAzimuthAngle (int angle) Parameters: <ul style="list-style-type: none">• angle - azimuth angle Returns: <ul style="list-style-type: none">• static int - None
COORD_NormalizeElevationAngle To normalize the elevation angle	static int COORD_NormalizeElevationAngle (int angle) Parameters: <ul style="list-style-type: none">• angle - Angle of elevation Returns: <ul style="list-style-type: none">• static int - None
COORD_NormalizeAngleIndex	static int COORD_NormalizeAngleIndex (int angleIndex, int angleResolution)

<p>To normalize the angleIndex</p>	<p>Parameters:</p> <ul style="list-style-type: none"> • angleIndex - angleIndex • angleResolution - angleResolution <p>Returns:</p> <ul style="list-style-type: none"> • static int - Return normalized angleIndex
<p>COORD_CalcDistance</p> <p>To calculate the distance between two nodes(points) given the coordinateSystemType and the coordinates of the two points</p>	<p>BOOL COORD_CalcDistance (int coordinateSystemType, const Coordinates* position1, const Coordinates* position2, CoordinateType distance)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • coordinateSystemType - type of coordinate system • position1 - coordinates of a point • position2 - coordinates of a point • distance - distance between two points <p>Returns:</p> <ul style="list-style-type: none"> • BOOL - whether the distance is calculated from position1 to position2
<p>COORD_CalcDistanceAndAngle</p> <p>To calculate the Distance and Angle</p>	<p>static void COORD_CalcDistanceAndAngle (int coordinateSystemType, const position1, const position2, double* distance, Orientation* DOA1, Orientation* DOA2)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • coordinateSystemType - coordinateSystem Type • position1 - Coordinates* • position2 - Coordinates* • distance - distance • DOA1 - DOA 1 • DOA2 - DOA 2 <p>Returns:</p> <ul style="list-style-type: none"> • static void - None
<p>COORD_ChangeCoordinateSystem</p> <p>Re-calculate coordinate in a new coordinate system</p>	<p>static void COORD_ChangeCoordinateSystem (const CoordinateRepresentationType source_type, const Coordinates* const source, const CoordinateRepresentationType target_type, Coordinates* const target)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • source_type - coordinate system of

	<ul style="list-style-type: none"> • source - coordinates of point to convert • target_type - coordinate system to • target - coordinate in new coordinate system <p>Returns:</p> <ul style="list-style-type: none"> • static void - None
COORD_ChangeCoordinateSystem	<p>static void COORD_ChangeCoordinateSystem (const Coordinates* const source, const CoordinateRepresentationType target_type, Coordinates* const target)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • source - coordinates of point to convert • target_type - coordinate systme to • target - coordinate in new coordinate system <p>Returns:</p> <ul style="list-style-type: none"> • static void - None
COORD_GeodeticToGeocentricCartesian	<p>static void COORD_GeodeticToGeocentricCartesian (const Coordinates* const source, Coordinates* const target)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • source - coordinate in GEODETIC • target - new coordinate in GEOCENTRIC_CARTESIAN <p>Returns:</p> <ul style="list-style-type: none"> • static void - None
COORD_GeocentricCartesianToGeodetic	<p>static void COORD_GeocentricCartesianToGeodetic (const Coordinates* const source, Coordinates* const target)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • source - coordinate in GEOCENTRIC_CARTESIAN • target - new coordinate in GEODETIC <p>Returns:</p> <ul style="list-style-type: none"> • static void - None
COORD_JGISToGeodetic	<p>static void COORD_JGISToGeodetic (const Coordinates* const source, Coordinates* const target)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • source - coordinate in JGIS

	<ul style="list-style-type: none"> • target - new coordinate in GEODETIC <p>Returns:</p> <ul style="list-style-type: none"> • static void - None
COORD_JGISToUnreferencedCartesian	<p>static void COORD_JGISToUnreferencedCartesian (const Coordinates* const source, Coordinates* const target)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • source - coordinate in JGIS • target - new coordinate in UNREFERENCED_CARTESIAN <p>Returns:</p> <ul style="list-style-type: none"> • static void - None
COORD_ConvertToCoordinates	<p>static void COORD_ConvertToCoordinates (char* buf, Coordinates* coordinates)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • buf - input string to be converted to coordinates • coordinates - Pointer to the coordinates <p>Returns:</p> <ul style="list-style-type: none"> • static void - None
COORD_MapCoordinateSystemToType	<p>static void COORD_MapCoordinateSystemToType (int coordinateSystem, Coordinates* coordinates)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • coordinateSystem - enum value indicating coordinate system • coordinates - Pointer to the coordinates <p>Returns:</p> <ul style="list-style-type: none"> • static void - None
COORD_NormalizeLongitude	<p>static void COORD_NormalizeLongitude (Coordinates* coordinates)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • coordinates - Pointer to the coordinates <p>Returns:</p> <ul style="list-style-type: none"> • static void - None
COORD_PointWithinRange	bool COORD_PointWithinRange (int coordinateSystemType, Coordinates* sw, Coordinates* ne, Coordinates* point)

	<p>Is the point within the given range. Assume -90 <= lat <= 90 and -180 <= long <= 180 for all inputs.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • coordinateSystemType - Cartesian or Geodetic • sw - Pointer to the SW corner (0,0) if Cartesian • ne - Pointer to the NE corner (dimensions if Cartesian) • point - Pointer to the coordinates <p>Returns:</p> <ul style="list-style-type: none"> • bool - True if within range
COORD_RegionsOverlap Determine whether the given regions overlap at all.	<p>bool COORD_RegionsOverlap (int coordinateSystemType, Coordinates* sw1, Coordinates* ne1, Coordinates* sw2, Coordinates* ne2)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • coordinateSystemType - Cartesian or Geodetic • sw1 - Pointer to the SW corner of the first region • ne1 - Pointer to the NE corner of the first region • sw2 - Pointer to the SW corner of the second region • ne2 - Pointer to the NE corner of the second region <p>Returns:</p> <ul style="list-style-type: none"> • bool - true if the regions overlap at all.
COORD_LongitudeDelta Convenience function for geodetic that, given two longitudes, returns the difference (in degrees) in the shorter direction.	<p>void COORD_LongitudeDelta (CoordinateType long1, CoordinateType long2)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • long1 - coordinate 1 • long2 - coordinate 2 <p>Returns:</p> <ul style="list-style-type: none"> • void - None
COORD_PrintCoordinates Prints the coordinates in a human readable format.	<p>void COORD_PrintCoordinates (int coordinateSystemType, Coordinates* point)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • coordinateSystemType - Cartesian or Geodetic • point - Pointer to the coordinates <p>Returns:</p>

- void - None



Contact [QualNet Support](#) for questions pertaining to the QualNet API Reference. This document is confidential and proprietary.
It may not be reproduced or distributed without the expressed written consent of [Scalable Network Technologies](#).

QualNet® is a Registered Trademark of [Scalable Network Technologies](#).

Copyright © 2001-2012 [Scalable Network Technologies, Inc.](#). All rights reserved.

QualNet 6.1 API Reference

ERROR

This file defines data structures and functions used in error-handling.

Constant / Data Structure Summary

Type	Name
CONSTANT	ERROR_ASSERTION Defines the ERROR_ASSERTION constant
CONSTANT	ERROR_ERROR Defines the ERROR_ERROR constant
CONSTANT	ERROR_WARNING Defines the ERROR_WARNING constant

Function / Macro Summary

Return Type	Summary
MACRO	ERROR Assert(expr, str) May be used in place of assert,to include an error message
MACRO	assert(expr) In DEBUG mode assert macro will be replaced by ERROR_WriteError with ERROR_ASSERTION type
MACRO	ERROR_ReportError(str) Function call used to report an error condition in QualNet, and notify GUI of such.
MACRO	ERROR_ReportWarning(str)

	Function call used to report a recoverable error condition. This macro in turns calls ERROR_WriteError with ERROR_WARNING type. It reports a warning message in QualNet, and notify GUI of such ERROR_WriteError(int type, char* condition, char* msg, char* file, int lineno)
extern BOOL	Function call used to report failed assertions, errors, and warnings, and notify the GUI of such. The user should not call this function directly, but should use one of the previously defined macros. ERROR_InstallHandler(int type, char* condition, char* msg, char* file, int lineno, QErrorHandler functionPointer)
void	Function used to register a callback function. The callback function will be invoked by ERROR_ when ERROR_WriteError () is invoked. For example - logging error messages into a log file or send the error messages to another application (e.g. to the Qualnet IDE that started the simulation.) ERROR_ReportMissingAddon(const char* model, const char* addon)
void	Reports an error when user attempts to use a model that hasn't been installed, either because the customer hasn't purchased that feature, or they haven't downloaded and compiled it. ERROR_ReportMissingInterface(const char* model, const char* iface)
void	Reports an error when user attempts to use a model that hasn't been installed, either because the customer hasn't purchased that feature, or they haven't downloaded and compiled it. ERROR_ReportMissingLibrary(const char* model, const char* library)

Constant / Data Structure Detail

Constant	ERROR_ASSERTION 0 Defines the ERROR_ASSERTION constant
Constant	ERROR_ERROR 1 Defines the ERROR_ERROR constant
Constant	ERROR_WARNING 2

Function / Macro Detail

Function / Macro	Format
ERROR Assert(expr, str)	May be used in place of assert,to include an error message
assert(expr)	In DEBUG mode assert macro will be replaced by ERROR_WriteError with ERROR_ASSERTION type
ERROR_ReportError(str)	Function call used to report an error condition in QualNet, and notify GUI of such.
ERROR_ReportWarning(str)	Function call used to report a recoverable error condition. This macro in turns calls ERROR_WriteError with ERROR_WARNING type. It reports a warning message in QualNet, and notify GUI of such
ERROR_WriteError	<p>extern BOOL ERROR_WriteError (int type, char* condition, char* msg, char* file, int lineno)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • type - assertion, error, or warning • condition - a string representing the failed boolean condition • msg - an error message • file - the file name in which the assertion failed • lineno - the line on which the assertion failed. <p>Returns:</p> <ul style="list-style-type: none"> • extern BOOL - None
ERROR_InstallHandler	<p>void ERROR_InstallHandler (int type, char* condition, char* msg, char* file, int lineno, QErrorHandler functionPointer)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • type - assertion, error, or warning • condition - a string representing the failed boolean condition • msg - an error message • file - the file name in which the assertion failed • lineno - the line on which the assertion failed. • functionPointer - pointer to a function with signature

	<p>Returns:</p> <ul style="list-style-type: none"> • void - None
ERROR_ReportMissingAddon	<p>void ERROR_ReportMissingAddon (const char* model, const char* addon)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • model - the name of the model/protocol being used. • addon - the name of the addon to which the model belongs <p>Returns:</p> <ul style="list-style-type: none"> • void - None
ERROR_ReportMissingInterface	<p>void ERROR_ReportMissingInterface (const char* model, const char* iface)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • model - the name of the model/protocol being used. • iface - the name of the interface to which the model belongs <p>Returns:</p> <ul style="list-style-type: none"> • void - None
ERROR_ReportMissingLibrary	<p>void ERROR_ReportMissingLibrary (const char* model, const char* library)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • model - the name of the model/protocol being used. • library - the name of the library to which the model belongs <p>Returns:</p> <ul style="list-style-type: none"> • void - None

Contact [QualNet Support](#) for questions pertaining to the QualNet API Reference. This document is confidential and proprietary.
It may not be reproduced or distributed without the expressed written consent of [Scalable Network Technologies](#).



QualNet® is a Registered Trademark of [Scalable Network Technologies](#).

QualNet 6.1 API Reference

EXTERNAL

This file defines the generic interface to external modules.

Constant / Data Structure Summary

Type	Name
CONSTANT	EXTERNAL_MAX_TIME The maximum possible time
CONSTANT	EXTERNAL_NUM_CPU_TIMING_INTERVAL_GUESSES The number of guesses to make for the cpu timing interval
CONSTANT	EXTERNAL_MAPPING_TABLE_SIZE The size of an interface's mapping hash table
CONSTANT	EXTERNAL_NUM_FUNCTIONS The number of functions an interface may implement
CONSTANT	EXTERNAL_RT_INDICATOR_INTERVAL The report interval of the realtime indication
CONSTANT	EXTERNAL_RT_INDICATOR_THRESHOLD red flag if the difference between realtime and the sim time is bigger than thread
ENUMERATION	ExternalInterfaceType Enumeration of different types of external interfaces
STRUCT	EXTERNAL_ThreadingMessage A struct containing data needed to send a message from an external thread to the main thread.
STRUCT	EXTERNAL_ThreadingForwarded

	A struct containing data needed to send a forwarded packet from the main thread to an external forward function EXTERNAL_Mapping
STRUCT	A linked list node containing one mapping. The key may be of any size, specified by keySize. The value the key maps to is a pointer to some piece of data. It is assumed that whoever created the mapping will know what to do with the pointer. The user will not use this structure directly.
STRUCT	EXTERNAL_MobilityEvent
STRUCT	A linked list of mobility events EXTERNAL_MobilityEventBuffer
STRUCT	A buffer containing all mobility events yet to be added to the simulation. EXTERNAL_InterfaceList
STRUCT	A list containing all of the registered external entities EXTERNAL_Interface
STRUCT	The information pertaining to one external interface

Function / Macro Summary

Return Type	Summary
EXTERNAL_Interface *	<p>EXTERNAL_RegisterExternalInterface(EXTERNAL_InterfaceList* list, char* name, EXTERNAL_PerformanceParameters params)</p> <p>This function will register a new external interface with QualNet and create the necessary data structures. This function must be called before any other function that requires an EXTERNAL_Interface* argument.</p>
EXTERNAL_Interface *	<p>EXTERNAL_RegisterExternalInterface(EXTERNAL_InterfaceList* list, char* name, EXTERNAL_PerformanceParameters params, ExternalInterfaceType type)</p> <p>This function is an overloaded variation for registering a new external interface with QualNet</p>
void	<p>EXTERNAL_RegisterFunction(EXTERNAL_Interface* iface, EXTERNAL_FunctionType type, EXTERNAL function)</p> <p>Register a new function for an interface.</p>

void	EXTERNAL_SetTimeManagementRealTime (EXTERNAL_Interface* iface, clocktype lookahead)
void	Turns time management on and specifies the lookahead value. The lookahead value may be changed later by calling EXTERNAL_ChangeRealTimeLookahead().
void	EXTERNAL_ChangeRealTimeLookahead (EXTERNAL_Interface* iface, clocktype lookahead)
void	Modifies the lookahead value. Must be called after EXTERNAL_SetTimeManagementRealTime(). May be called during the simulation.
void	EXTERNAL_InitializeWarmupParams (EXTERNAL_Interface* iface, NodeInput* nodeInput)
void	EXTERNAL_RealtimeIndicator (EXTERNAL_Interface* iface, NodeInput* nodeInput)
void	for realtime indicator initialization
void	EXTERNAL_SetWarmupTime (EXTERNAL_Interface* iface, clocktype warmup)
void	Sets this interface's warmup time. The actual warmup time used is the maximum of all interface's. The default is no warmup time (warmup == -1). This function must be called before or during the initialize nodes step. It will have no effect during the simulation.
void	EXTERNAL_BeginWarmup (EXTERNAL_Interface* iface)
clocktype	Each interface that calls EXTERNAL_SetWarmupTime must call EXTERNAL_BeginWarmup when it is ready to enter warmup time.
clocktype	EXTERNAL_QueryWarmupTime (EXTERNAL_Interface* iface)
clocktype	Get the warmup time for the entire simulation. Interfaces should use this function to test when warmup time is over.
BOOL	EXTERNAL_IsInWarmup (EXTERNAL_Interface* iface)
BOOL	Check if QualNet is in the warmup phase
BOOL	EXTERNAL_IsInWarmup (PartitionData* partitionData)
void	Check if QualNet is in the warmup phase
void	EXTERNAL_Pause (EXTERNAL_Interface* iface)
void	Pause every interface. Only usable when running in real-time.
void	EXTERNAL_Resume (EXTERNAL_Interface* iface)
clocktype	Resume every interface. Only usable when running in real-time, and after calling pause.
clocktype	EXTERNAL_QueryExternalTime (EXTERNAL_Interface* iface)

	This function will return the External Time of an external interface
clocktype	EXTERNAL_QuerySimulationTime (EXTERNAL_Interface* iface)
	This function will return the Simulation Time
void	EXTERNAL_Sleep (clocktype amount)
	This function will sleep for a minimum amount of time as indicated by the amount parameter. Depending on which platform it is called on the amount of time spent sleeping could be greater.
void	EXTERNAL_SetReceiveDelay (EXTERNAL_Interface* iface, clocktype delay)
	This function will set the minimum delay between two consecutive calls to the receive function. The time used is the simulation time.
void	EXTERNAL_SendMessage (EXTERNAL_Interface* iface, Node* node, Message* msg, clocktype timestamp)
	This function will send a message from the external interface. This function is thread-safe.
void	EXTERNAL_ForwardData (EXTERNAL_Interface* iface, Node* node, void* forwardData, int forwardSize, EXTERNAL_ForwardData_ReceiverOpt FwdReceiverOpt)
	Send data back to the external source with no time stamp. The user defined Forward function will receive this message and process it. This will handle threading issues if necessary.
void	EXTERNAL_RemoteForwardData (EXTERNAL_Interface* iface, Node* node, void* forwardData, int forwardSize, int partitionId)
	Send data back to the external source with no time stamp. This function is similar to EXTERNAL_ForwardData, except that this function can forward the message to an external interface on a different partition.
void	EXTERNAL_ForwardDataTimeStamped (EXTERNAL_Interface* iface, Node* node, Message* message, clocktype timestamp)
	Send data in the form of a message back to the external source with a time stamp. The user defined Forward function will receive this message and process it. This will handle threading issues if necessary.
void	EXTERNAL_UserFunctionRegistration (EXTERNAL_InterfaceList * list, NodeInput* nodeInput)
	This function will give a convenient place for users to add their function registration code. This is the only part of the External Interface API code that the user is expected to modify.
void	EXTERNAL_InitializeInterface (EXTERNAL_Interface* iface)
	This function will initialize an external interface

EXTERNAL

void	<code>EXTERNAL_FinalizeExternalInterface</code> (EXTERNAL_Interface* iface)
void	This function will free an external interface, as well as call the finalize function registered by EXTERNAL_RegisterFinalizeFunction()
void	<code>EXTERNAL_InitializeInterfaceList</code> (EXTERNAL_InterfaceList* list, PartitionData* partition)
void	This function will initialize an external interface list
void	<code>EXTERNAL_Bootstrap</code> (int argc, char* argv [])
void	This function will be called early in the simulation initialization process (after MPI_Init()), but before partitions are created, and before EXTERNAL_InitializeInterfaceList(). In a shared parallel simulation the threads for partitions won't be created yet.
void	<code>EXTERNAL_PreBootstrap</code> (int argc, char* argv [])
void	This function will be called early in the simulation initialization process (after MPI_Init()), but before partitions are created, and before EXTERNAL_InitializeInterfaceList(). In a shared parallel simulation the threads for partitions won't be created yet. This function handles the mini-configuration file conversion, and make sure that if simProps needs to change it is changed and then a broadcast message is sent to other partitions.
void	<code>EXTERNAL_FinalizeInterfaceList</code> (EXTERNAL_InterfaceList* list)
void	This function will finalize all ExternalInterfaces in the list, as well as the list itself
EXTERNAL_Interface*	<code>EXTERNAL_GetInterfaceByName</code> (EXTERNAL_InterfaceList* list, char* name)
void	This function will search an interface list for an interface with the given name
void	<code>EXTERNAL_CallInitializeFunctions</code> (EXTERNAL_InterfaceList* list, NodeInput* nodeInput)
void	This function will call all initialize functions
void	<code>EXTERNAL_CallInitializeNodesFunctions</code> (EXTERNAL_InterfaceList* list, NodeInput* nodeInput)
void	This function will call all intialize nodes functions
clocktype	<code>EXTERNAL_StartThreads</code> (EXTERNAL_InterfaceList* list)
void	This function will start the receive/forward threads for all threaded interfaces. Called after EXTERNAL_CallInitializeNodesFunctions.
clocktype	<code>EXTERNAL_CalculateMinSimulationHorizon</code> (EXTERNAL_InterfaceList* list, clocktype now)
void	This function will call all simulation horizon functions to determine how far into the future the simulation can run. An individual simulation horizon function will only be called if the current time (now) is >= that interface's current horizon.
void	<code>EXTERNAL_CallReceiveFunctions</code> (EXTERNAL_InterfaceList* list)

	This function will call all receive function that were not started in a thread
void	<code>EXTERNAL_CallFinalizeFunctions(INTERNAL_InterfaceList* list)</code>
	This function will call all finalize functions
void	<code>EXTERNAL_InitializeExternalInterfaces(partitionData* partitionData)</code>
	Function used to initialize a generic interface to an external source of messages, e.g. an HLA federate. Called before nodes are created.
void	<code>EXTERNAL_PostInitialize(partitionData* partitionData)</code>
	Function used to initialize a generic interface to an external source of messages, e.g. an HLA federate. Called after nodes are created. The developer can use either this function, the preceding one or both.
void	<code>EXTERNAL_GetExternalMessages(partitionData* partitionData, clocktype nextInternalEventTime)</code>
	Function used to retrieve messages from a remote source, such as a DIS gateway or HLA federation. Called before events at time X are executed. Many events at time X may be executed before the next call
void	<code>EXTERNAL_Finalize(partitionData* partitionData)</code>
	Shuts down interfaces to external simulators
void	<code>EXTERNAL_SetActive(partitionData* partitionData)</code>
	Sets isActive parameter based on interface registration
void	<code>EXTERNAL_DeactivateInterface(INTERNAL_Interface* ifaceToDeactivate)</code>
	Remove the indicated interface for the list of currently activated interfaces.
void	<code>EXTERNAL_ProcessEvent(Node* node, Message* msg)</code>
	Process events meant for external code.
clocktype	<code>GetNextInternalEventTime(PartitionData* partitionData)</code>
	Get the next internal event on the given partition. This includes both regular events and mobility events.

Constant / Data Structure Detail

Constant	<code>EXTERNAL_MAX_TIME</code> The maximum possible time
Constant	<code>EXTERNAL_NUM_CPU_TIMING_INTERVAL_GUESSES</code> 4 The number of guesses to make for the cpu timing interval
Constant	<code>EXTERNAL_MAPPING_TABLE_SIZE</code> 31 The size of an interface's mapping hash table
Constant	<code>EXTERNAL_NUM_FUNCTIONS</code> 8 The number of functions an interface may implement
Constant	<code>EXTERNAL_RT_INDICATOR_INTERVAL</code> 0.1 second The report interval of the realtime indication
Constant	<code>EXTERNAL_RT_INDICATOR_THRESHOLD</code> 1 second now red flag if the difference between realtime and the sim time is bigger than thread
Enumeration	<code>ExternalInterfaceType</code> Enumeration of different types of external interfaces
Structure	<code>EXTERNAL_ThreadedMessage</code> A struct containing data needed to send a message from an external thread to the main thread.
Structure	<code>EXTERNAL_ThreadedForwarded</code> A struct containing data needed to send a forwarded packet from the main thread to an external forward function
Structure	<code>EXTERNAL_Mapping</code> A linked list node containing one mapping. The key may be of any size, specified by keySize. The value the key maps to is a pointer to

	some piece of data. It is assumed that whoever created the mapping will know what to do with the pointer. The user will not use this structure directly.
Structure	<p>EXTERNAL_MobilityEvent</p> <p>A linked list of mobility events</p>
Structure	<p>EXTERNAL_MobilityEventBuffer</p> <p>A buffer containing all mobility events yet to be added to the simulation.</p>
Structure	<p>EXTERNAL_InterfaceList</p> <p>A list containing all of the registered external entities</p>
Structure	<p>EXTERNAL_Interface</p> <p>The information pertaining to one external interface</p>

Function / Macro Detail

Function / Macro	Format
EXTERNAL_RegisterExternalInterface This function will register a new external interface with QualNet and create the necessary data structures. This function must be called before any other function that requires an EXTERNAL_Interface* argument.	<p>EXTERNAL_Interface * EXTERNAL_RegisterExternalInterface (EXTERNAL_InterfaceList* list, char* name, EXTERNAL_PerformanceParameters params)</p> <p>Parameters:</p> <ul style="list-style-type: none"> list - The list of external interfaces name - The name of the external interface. params - The performance parameters <p>Returns:</p> <ul style="list-style-type: none"> EXTERNAL_Interface * - A pointer to the newly registered external interface
EXTERNAL_RegisterExternalInterface This function is an overloaded variation. for registering a new external interface with QualNet	<p>EXTERNAL_Interface * EXTERNAL_RegisterExternalInterface (EXTERNAL_InterfaceList* list, char* name, EXTERNAL_PerformanceParameters params, ExternalInterfaceType type)</p> <p>Parameters:</p> <ul style="list-style-type: none"> list - The list of external interfaces name - The name of the external interface. params - The performance parameters

	<ul style="list-style-type: none"> • <code>type</code> - PartitionData's interfaceTable will be <p>Returns:</p> <ul style="list-style-type: none"> • <code>EXTERNAL_Interface *</code> - A pointer to the newly registered external interface
EXTERNAL_RegisterFunction	<p>Register a new function for an interface.</p> <p><code>void EXTERNAL_RegisterFunction (EXTERNAL_Interface* iface, EXTERNAL_FunctionType type, EXTERNAL function)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>iface</code> - The external interface • <code>type</code> - the type of function • <code>function</code> - Function pointer to be called <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
EXTERNAL_SetTimeManagementRealTime	<p>Turns time management on and specifies the lookahead value. The lookahead value may be changed later by calling <code>EXTERNAL_ChangeRealTimeLookahead()</code>.</p> <p><code>void EXTERNAL_SetTimeManagementRealTime (EXTERNAL_Interface* iface, clocktype lookahead)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>iface</code> - The external interface • <code>lookahead</code> - How far into the future the simulation is <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
EXTERNAL_ChangeRealTimeLookahead	<p>Modifies the lookahead value. Must be called after <code>EXTERNAL_SetTimeManagementRealTime()</code>. May be called during the simulation.</p> <p><code>void EXTERNAL_ChangeRealTimeLookahead (EXTERNAL_Interface* iface, clocktype lookahead)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>iface</code> - The external interface • <code>lookahead</code> - The new lookahead value <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
EXTERNAL_InitializeWarmupParams	<p><code>void EXTERNAL_InitializeWarmupParams (EXTERNAL_Interface* iface, NodeInput* nodeInput)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>iface</code> - The external interface • <code>nodeInput</code> - The configuration file. <p>Returns:</p>

	<ul style="list-style-type: none"> • void - None
EXTERNAL_RealtimeIndicator for realtime indicator initialization	<p>void EXTERNAL_RealtimeIndicator (EXTERNAL_Interface* iface, NodeInput* nodeInput)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • iface - The external interface • nodeInput - The configuration file. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
EXTERNAL_SetWarmupTime Sets this interface's warmup time. The actual warmup time used is the maximum of all interface's. The default is no warmup time (warmup == -1). This function must be called before or during the initialize nodes step. It will have no effect during the simulation.	<p>void EXTERNAL_SetWarmupTime (EXTERNAL_Interface* iface, clocktype warmup)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • iface - The external interface • warmup - The warmup time for this interface <p>Returns:</p> <ul style="list-style-type: none"> • void - None
EXTERNAL_BeginWarmup Each interface that calls EXTERNAL_SetWarmupTime must call EXTERNAL_BeginWarmup when it is ready to enter warmup time.	<p>void EXTERNAL_BeginWarmup (EXTERNAL_Interface* iface)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • iface - The external interface <p>Returns:</p> <ul style="list-style-type: none"> • void - None
EXTERNAL_QueryWarmupTime Get the warmup time for the entire simulation. Interfaces should use this function to test when warmup time is over.	<p>clocktype EXTERNAL_QueryWarmupTime (EXTERNAL_Interface* iface)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • iface - The external interface <p>Returns:</p> <ul style="list-style-type: none"> • clocktype - The inclusive end of warmup time. -1 if no warmup time.
EXTERNAL_IsInWarmup Check if QualNet is in the warmup phase	<p>BOOL EXTERNAL_IsInWarmup (EXTERNAL_Interface* iface)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • iface - The external interface <p>Returns:</p>

	<ul style="list-style-type: none"> • BOOL - TRUE if in warmup, FALSE if not This is now a wrapper function ONLY. It passes a pointer to partition data. We overload this function in order to check if simulator is in warm-up phase even when we do not have access to External interface
EXTERNAL_IsInWarmup	<p>BOOL EXTERNAL_IsInWarmup (PartitionData* partitionData)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>partitionData</code> - pointer to partition's data structure <p>Returns:</p> <ul style="list-style-type: none"> • BOOL - TRUE if in warmup, FALSE if not
EXTERNAL_Pause	<p>void EXTERNAL_Pause (EXTERNAL_Interface* iface)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>iface</code> - The external interface <p>Returns:</p> <ul style="list-style-type: none"> • void - None
EXTERNAL_Resume	<p>void EXTERNAL_Resume (EXTERNAL_Interface* iface)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>iface</code> - The external interface <p>Returns:</p> <ul style="list-style-type: none"> • void - None
EXTERNAL_QueryExternalTime	<p>clocktype EXTERNAL_QueryExternalTime (EXTERNAL_Interface* iface)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>iface</code> - The external interface <p>Returns:</p> <ul style="list-style-type: none"> • <code>clocktype</code> - The External Time. Returns EXTERNAL_MAX_TIME if no time function is defined.
EXTERNAL_QuerySimulationTime	<p>clocktype EXTERNAL_QuerySimulationTime (EXTERNAL_Interface* iface)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>iface</code> - The external interface <p>Returns:</p> <ul style="list-style-type: none"> • <code>clocktype</code> - The Simulation Time

EXTERNAL_Sleep	<p>This function will sleep for a minimum amount of time as indicated by the amount parameter. Depending on which platform it is called on the amount of time spent sleeping could be greater.</p>	<p>void EXTERNAL_Sleep (clocktype amount)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • amount - The amount of time to sleep <p>Returns:</p> <ul style="list-style-type: none"> • void - None
EXTERNAL_SetReceiveDelay	<p>This function will set the minimum delay between two consecutive calls to the receive function. The time used is the simulation time.</p>	<p>void EXTERNAL_SetReceiveDelay (EXTERNAL_Interface* iface, clocktype delay)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • iface - The external interface • delay - The minimum delay <p>Returns:</p> <ul style="list-style-type: none"> • void - None
EXTERNAL_SendMessage	<p>This function will send a message from the external interface. This function is thread-safe.</p>	<p>void EXTERNAL_SendMessage (EXTERNAL_Interface* iface, Node* node, Message* msg, clocktype timestamp)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • iface - The external interface • node - Node sending the message • msg - The message to send • timestamp - The timestamp for this message. Since this message is <p>Returns:</p> <ul style="list-style-type: none"> • void - None
EXTERNAL_ForwardData	<p>Send data back to the external source with no time stamp. The user defined Forward function will receive this message and process it. This will handle threading issues if necessary.</p>	<p>void EXTERNAL_ForwardData (EXTERNAL_Interface* iface, Node* node, void* forwardData, int forwardSize, EXTERNAL_ForwardData_ReceiverOpt FwdReceiverOpt)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • iface - The external interface • node - The node that is forwarding the data • forwardData - The data to forward • forwardSize - The size of the data to forward • FwdReceiverOpt - Whether to store the <p>Returns:</p>

	<ul style="list-style-type: none"> • void - None
EXTERNAL_RemoteForwardData	<p>Send data back to the external source with no time stamp. This function is similar to EXTERNAL_ForwardData, except that this function can forward the message to an external interface on a different partition.</p> <p>void EXTERNAL_RemoteForwardData (EXTERNAL_Interface* iface, Node* node, void* forwardData, int forwardSize, int partitionId)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • iface - The external interface • node - The node that is forwarding the data • forwardData - The data to forward • forwardSize - The size of the data to forward • partitionId - The partition Id to forward the message to <p>Returns:</p> <ul style="list-style-type: none"> • void - None
EXTERNAL_ForwardDataTimeStamped	<p>Send data in the form of a message back to the external source with a time stamp. The user defined Forward function will receive this message and process it. This will handle threading issues if necessary.</p> <p>void EXTERNAL_ForwardDataTimeStamped (EXTERNAL_Interface* iface, Node* node, Message* message, clocktype timestamp)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • iface - The external interface • node - The node that is forwarding the data • message - The message • timestamp - The time stamp. This value is in external <p>Returns:</p> <ul style="list-style-type: none"> • void - None
EXTERNAL_UserFunctionRegistration	<p>This function will give a convenient place for users to add their function registration code. This is the only part of the External Interface API code that the user is expected to modify.</p> <p>void EXTERNAL_UserFunctionRegistration (EXTERNAL_InterfaceList * list, NodeInput* nodeInput)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • list - The list of external interfaces • nodeInput - The configuration file <p>Returns:</p> <ul style="list-style-type: none"> • void - None
EXTERNAL_InitializeInterface	<p>This function will initialize an external interface</p> <p>void EXTERNAL_InitializeInterface (EXTERNAL_Interface* iface)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • iface - The external interface

	<p>Returns:</p> <ul style="list-style-type: none"> • void - None
EXTERNAL_FinalizeExternalInterface	<p>This function will free an external interface, as well as call the finalize function registered by EXTERNAL_RegisterFinalizeFunction()</p> <p>void EXTERNAL_FinalizeExternalInterface (EXTERNAL_Interface* iface)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • iface - The external interface <p>Returns:</p> <ul style="list-style-type: none"> • void - None
EXTERNAL_InitializeInterfaceList	<p>This function will initialize an external interface list</p> <p>void EXTERNAL_InitializeInterfaceList (EXTERNAL_InterfaceList* list, PartitionData* partition)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • list - The external interface list • partition - The partition it will run on <p>Returns:</p> <ul style="list-style-type: none"> • void - None
EXTERNAL_Bootstrap	<p>This function will be called early in the simulation initialization process (after MPI_Init(), but before partitions are created, and before EXTERNAL_InitializeInterfaceList()). In a shared parallel simulation the threads for partitions won't be created yet.</p> <p>void EXTERNAL_Bootstrap (int argc, char* argv [])</p> <p>Parameters:</p> <ul style="list-style-type: none"> • argc - The command line argument count • argv [] - The command line arguments <p>Returns:</p> <ul style="list-style-type: none"> • void - None
EXTERNAL_PreBootstrap	<p>This function will be called early in the simulation initialization process (after MPI_Init(), but before partitions are created, and before EXTERNAL_InitializeInterfaceList()). In a shared parallel simulation the threads for partitions won't be created yet. This function handles the mini-configuration file conversion, and make sure that if simProps needs to change it is changed and then a broadcast message is sent to other partitions.</p> <p>void EXTERNAL_PreBootstrap (int argc, char* argv [])</p> <p>Parameters:</p> <ul style="list-style-type: none"> • argc - The command line argument count • argv [] - The command line arguments <p>Returns:</p> <ul style="list-style-type: none"> • void - None
EXTERNAL_FinalizeInterfaceList	void EXTERNAL_FinalizeInterfaceList (EXTERNAL_InterfaceList* list)

	<p>This function will finalize all ExternalInterfaces in the list, as well as the list itself</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>list</code> - The external interface list <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
EXTERNAL_GetInterfaceByName <p>This function will search an interface list for an interface with the given name</p>	<p><code>EXTERNAL_Interface* EXTERNAL_GetInterfaceByName (EXTERNAL_InterfaceList* list, char* name)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>list</code> - The external interface list • <code>name</code> - The interface's name <p>Returns:</p> <ul style="list-style-type: none"> • <code>EXTERNAL_Interface*</code> - The interface, NULL if not found
EXTERNAL_CallInitializeFunctions <p>This function will call all initialize functions</p>	<p><code>void EXTERNAL_CallInitializeFunctions (EXTERNAL_InterfaceList* list, NodeInput* nodeInput)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>list</code> - The list of external interfaces • <code>nodeInput</code> - The input configuration file <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
EXTERNAL_CallInitializeNodesFunctions <p>This function will call all intialize nodes functions</p>	<p><code>void EXTERNAL_CallInitializeNodesFunctions (EXTERNAL_InterfaceList* list, NodeInput* nodeInput)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>list</code> - The list of external interfaces • <code>nodeInput</code> - The input configuration file <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
EXTERNAL_StartThreads <p>This function will start the receive/forward threads for all threaded interfaces. Called after EXTERNAL_CallInitializeNodesFunctions.</p>	<p><code>void EXTERNAL_StartThreads (EXTERNAL_InterfaceList* list)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>list</code> - The list of external interfaces <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None

EXTERNAL_CalculateMinSimulationHorizon	<p>This function will call all simulation horizon functions to determine how far into the future the simulation can run. An individual simulation horizon function will only be called if the current time (now) is \geq that interface's current horizon.</p>	<p>clocktype EXTERNAL_CalculateMinSimulationHorizon (EXTERNAL_InterfaceList* list, clocktype now)</p> <p>Parameters:</p> <ul style="list-style-type: none"> list - The list of external interfaces now - The current time <p>Returns:</p> <ul style="list-style-type: none"> clocktype - The minimum Simulation Horizon, or EXTERNAL_MAX_TIME if no horizon.
EXTERNAL_CallReceiveFunctions	<p>This function will call all receive function that were not started in a thread</p>	<p>void EXTERNAL_CallReceiveFunctions (EXTERNAL_InterfaceList* list)</p> <p>Parameters:</p> <ul style="list-style-type: none"> list - The list of external interfaces <p>Returns:</p> <ul style="list-style-type: none"> void - None
EXTERNAL_CallFinalizeFunctions	<p>This function will call all finalize functions</p>	<p>void EXTERNAL_CallFinalizeFunctions (EXTERNAL_InterfaceList* list)</p> <p>Parameters:</p> <ul style="list-style-type: none"> list - The list of external interfaces <p>Returns:</p> <ul style="list-style-type: none"> void - None
EXTERNAL_InitializeExternalInterfaces	<p>Function used to initialize a generic interface to an external source of messages, e.g. an HLA federate. Called before nodes are created.</p>	<p>void EXTERNAL_InitializeExternalInterfaces (partitionData* partitionData)</p> <p>Parameters:</p> <ul style="list-style-type: none"> partitionData - pointer to data for this partition <p>Returns:</p> <ul style="list-style-type: none"> void - None
EXTERNAL_PostInitialize	<p>Function used to initialize a generic interface to an external source of messages, e.g. an HLA federate. Called after nodes are created. The developer can use either this function, the preceding one or both.</p>	<p>void EXTERNAL_PostInitialize (partitionData* partitionData)</p> <p>Parameters:</p> <ul style="list-style-type: none"> partitionData - pointer to data for this partition <p>Returns:</p> <ul style="list-style-type: none"> void - None
EXTERNAL_GetExternalMessages		<p>void EXTERNAL_GetExternalMessages (partitionData* partitionData, clocktype nextInternalEventTime)</p> <p>Parameters:</p>

<p>Function used to retrieve messages from a remote source, such as a DIS gateway or HLA federation. Called before events at time X are executed. Many events at time X may be executed before the next call</p>	<ul style="list-style-type: none"> • <code>partitionData</code> - pointer to data for this partition • <code>nextInternalEventTime</code> - the time of the next event, <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
<p>EXTERNAL_Finalize</p> <p>Shuts down interfaces to external simulators</p>	<p><code>void EXTERNAL_Finalize (partitionData* partitionData)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>partitionData</code> - pointer to data for this partition <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
<p>EXTERNAL_SetActive</p> <p>Sets <code>isActive</code> parameter based on interface registration</p>	<p><code>void EXTERNAL_SetActive (partitionData* partitionData)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>partitionData</code> - pointer to data for this partition <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
<p>EXTERNAL_DeactivateInterface</p> <p>Remove the indicated interface for the list of currently activated interfaces.</p>	<p><code>void EXTERNAL_DeactivateInterface (EXTERNAL_Interface* ifaceToDeactivate)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>ifaceToDeactivate</code> - Pointer to the interface <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
<p>EXTERNAL_ProcessEvent</p> <p>Process events meant for external code.</p>	<p><code>void EXTERNAL_ProcessEvent (Node* node, Message* msg)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node data structure. • <code>msg</code> - Message to be processed. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
<p>GetNextInternalEventTime</p>	<p><code>clocktype GetNextInternalEventTime (PartitionData* partitionData)</code></p> <p>Parameters:</p>

Get the next internal event on the given partition.
This includes both regular events and mobility
events.

- partitionData - Pointer to the partition

Returns:

- clocktype - The next internal event



Contact [QualNet Support](#) for questions pertaining to the QualNet API Reference. This document is confidential and proprietary.
It may not be reproduced or distributed without the expressed written consent of [Scalable Network Technologies](#).

QualNet® is a Registered Trademark of [Scalable Network Technologies](#).

Copyright © 2001-2012 [Scalable Network Technologies, Inc.](#). All rights reserved.

QualNet 6.1 API Reference

EXTERNAL_SOCKET

This file describes utilities for managing socket connections to external programs.

Constant / Data Structure Summary

Type	Name
CONSTANT	EXTERNAL_DEFAULT_VAR_ARRAY_SIZE The default size of a VarArray
CONSTANT	THREADED_BUFFER_SIZE The thread buffer size
ENUMERATION	EXTERNAL_SocketErrorType A listing of error types that could occur.
STRUCT	EXTERNAL_VarArray A variable sized array. This structure is primarily used to assemble data to be sent on a socket connection.
STRUCT	EXTERNAL_Socket The socket data structure

Function / Macro Summary

Return Type	Summary
void	EXTERNAL_VarArrayInit (EXTERNAL_VarArray* array, unsigned int size) This function will initialize a VarArray and allocate memory for the array. When the array is finished being used, call EXTERNAL_VarArrayFree to free the memory.
void	EXTERNAL_VarArrayAccomodateSize (EXTERNAL_VarArray* array, unsigned int size)

	This function will increase the maximum size of the VarArray so that it can contain at least "size" bytes. EXTERNAL_VarArrayAppendData (EXTERNAL_VarArray* array, char* data, unsigned int size)
void	This function will add data to the end of the VarArray. The size of the VarArray will be increased if necessary. EXTERNAL_VarArrayConcatString (EXTERNAL_VarArray* array, char* string)
void	This function will add a string to the end of the VarArray including the terminating NULL character. This function ASSUMES that the previous data in the VarArray is also a string -- ie, several bytes of data terminated with a NULL character. If this is not the case then the function EXTERNAL_VarArrayAppendData should be used instead.
void	EXTERNAL_VarArrayFree (EXTERNAL_VarArray* array)
void	This function will free all memory allocated to the VarArray EXTERNAL_hton (void* ptr, unsigned size)
void	Convert data from host byte order to network byte order EXTERNAL_ntoh (void* ptr, unsigned size)
void	Convert data from network byte order to host byte order EXTERNAL_swapBitfield (void* ptr, unsigned size)
void	EXTERNAL_SocketInit (EXTERNAL_Socket* socket)
EXTERNAL_SocketErrorType	Initialize a socket. Must be called before all other socket API calls on the individual socket. EXTERNAL_SocketInitUDP (EXTERNAL_Socket* socket)
bool	Initialize a UDP socket. Must be called before all other socket API calls on the individual socket. EXTERNAL_SocketValid (EXTERNAL_Socket* socket)
EXTERNAL_SocketErrorType	Check if a socket connection is valid and no errors have occurred. EXTERNAL_SocketListen (EXTERNAL_Socket* listenSocket, int port, EXTERNAL_Socket* connectSocket)
EXTERNAL_SocketErrorType	Listen and accept a connections on a socket. This function is a wrapper for EXTERNAL_SocketInitListen and EXTERNAL_SocketAccept. EXTERNAL_SocketInitListen (EXTERNAL_Socket* listenSocket, int port)

	<p>Initialize an input socket and have it listen on the given port. Call EXTERNAL_SocketAccept to accept connections on the socket. Call EXTERNAL_SocketDataAvailable to see if there is an incoming connection that has not been accepted.</p>
EXTERNAL_SocketErrorType	<p>EXTERNAL_SocketAccept(EXTERNAL_Socket* listenSocket, EXTERNAL_Socket* connectSocket)</p> <p>Accept a connection on a listening socket. This operation may block if there is no incoming connection, use EXTERNAL_SocketDataAvailable to check if there is an incoming connection.</p>
EXTERNAL_SocketErrorType	<p>EXTERNAL_SocketDataAvailable(EXTERNAL_Socket* s, bool* available)</p> <p>Test if a socket has readable data. For a listening socket this will test for an incoming connection. For a data socket this will test if there is incoming data.</p>
EXTERNAL_SocketErrorType	<p>EXTERNAL_SocketConnect(EXTERNAL_Socket* socket, char* address, int port, int maxAttempts)</p> <p>Connect to a listening socket. The socket is set to non-blocking mode.</p>
EXTERNAL_SocketErrorType	<p>EXTERNAL_SocketSend(EXTERNAL_Socket* socket, char* data, unsigned int size, bool block)</p> <p>Send data on a connected socket. Since the socket is non-blocking, it is possible that the send would result in a block: If the "block" parameter is FALSE, then EXTERNAL_DataNotSent is returned, and no data is sent.</p>
EXTERNAL_SocketErrorType	<p>EXTERNAL_SocketSend(EXTERNAL_Socket* socket, EXTERNAL_VarArray* data, bool block)</p> <p>This is a wrapper for the above overloaded function.</p>
EXTERNAL_SocketErrorType	<p>EXTERNAL_SocketRecv(EXTERNAL_Socket* socket, char* data, unsigned int size, unsigned int* size, bool block)</p> <p>Receive data on a connected socket. Since the socket is non-blocking, it is possible that the send would result in a block: If the "block" parameter is FALSE, the "receiveSize" parameter will be set to the amount of data received before the blocking operation. This amount could be any value between 0 and size - 1.</p>
EXTERNAL_SocketErrorType	<p>EXTERNAL_SocketRecv(EXTERNAL_Socket* socket, char* data, unsigned int size, unsigned int* size, int* ip, int* port, bool block)</p> <p>Receive data on a UDP socket. Since the socket is non-blocking, it is possible that the send would result in a block: If the "block" parameter is FALSE, the "receiveSize" parameter will be set to the amount of data received before the blocking operation. This amount could be any value between 0 and size - 1.</p>
EXTERNAL_SocketErrorType	<p>EXTERNAL_SocketRecv(EXTERNAL_Socket* socket, std data)</p> <p>Receive data on a connected socket. Continues reading until a '\n' character is found. This function always blocks.</p>
EXTERNAL_SocketErrorType	<p>EXTERNAL_SocketClose(EXTERNAL_Socket* socket)</p> <p>Close a socket. Must be called for each socket that is listening or connected.</p>

Constant / Data Structure Detail

Constant	EXTERNAL_DEFAULT_VAR_ARRAY_SIZE 512 The default size of a VarArray
Constant	THREADED_BUFFER_SIZE 2000000 The thread buffer size
Enumeration	EXTERNAL_SocketErrorType A listing of error types that could occur.
Structure	EXTERNAL_VarArray A variable sized array. This structure is primarily used to assemble data to be sent on a socket connection.
Structure	EXTERNAL_Socket The socket data structure

Function / Macro Detail

Function / Macro	Format
EXTERNAL_VarArrayInit This function will initialize a VarArray and allocate memory for the array. When the array is finished being used, call EXTERNAL_VarArrayFree to free the memory.	void EXTERNAL_VarArrayInit (EXTERNAL_VarArray* array, unsigned int size) Parameters: <ul style="list-style-type: none">• array - Pointer to the uninitialized VarArray• size - The initial size of the array in bytes . Defaults Returns: <ul style="list-style-type: none">• void - None
EXTERNAL_VarArrayAccomodateSize	void EXTERNAL_VarArrayAccomodateSize (EXTERNAL_VarArray* array, unsigned int size) Parameters:

<p>This function will increase the maximum size of the VarArray so that it can contain at least "size" bytes.</p>	<ul style="list-style-type: none"> • <code>array</code> - Pointer to the VarArray • <code>size</code> - The new minimum size of the VarArray <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
<p>EXTERNAL_VarArrayAppendData</p> <p>This function will add data to the end of the VarArray. The size of the VarArray will be increased if necessary.</p>	<p><code>void EXTERNAL_VarArrayAppendData (EXTERNAL_VarArray* array, char* data, unsigned int size)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>array</code> - Pointer to the VarArray • <code>data</code> - Pointer to the data to add • <code>size</code> - The size of the data to add <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
<p>EXTERNAL_VarArrayConcatString</p> <p>This function will add a string to the end of the VarArray including the terminating NULL character. This function ASSUMES that the previous data in the VarArray is also a string -- ie, several bytes of data terminated with a NULL character. If this is not the case then the function <code>EXTERNAL_VarArrayAppendData</code> should be used instead.</p>	<p><code>void EXTERNAL_VarArrayConcatString (EXTERNAL_VarArray* array, char* string)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>array</code> - Pointer to the VarArray • <code>string</code> - The string <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
<p>EXTERNAL_VarArrayFree</p> <p>This function will free all memory allocated to the VarArray</p>	<p><code>void EXTERNAL_VarArrayFree (EXTERNAL_VarArray* array)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>array</code> - Pointer to the VarArray <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
<p>EXTERNAL_hton</p> <p>Convert data from host byte order to network byte order</p>	<p><code>void EXTERNAL_hton (void* ptr, unsigned size)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>ptr</code> - Pointer to the data • <code>size</code> - Size of the data

	<p>Returns:</p> <ul style="list-style-type: none"> • void - None
EXTERNAL_ntoh	<p>void EXTERNAL_ntoh (void* ptr, unsigned size)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • ptr - Pointer to the data • size - Size of the data <p>Returns:</p> <ul style="list-style-type: none"> • void - None
EXTERNAL_swapBitfield	<p>void EXTERNAL_swapBitfield (void* ptr, unsigned size)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • ptr - Pointer to the data • size - Size of the data (in bytes) <p>Returns:</p> <ul style="list-style-type: none"> • void - None
EXTERNAL_SocketInit	<p>void EXTERNAL_SocketInit (EXTERNAL_Socket* socket)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • socket - Pointer to the socket <p>Returns:</p> <ul style="list-style-type: none"> • void - None
EXTERNAL_SocketInitUDP	<p>EXTERNAL_SocketErrorType EXTERNAL_SocketInitUDP (EXTERNAL_Socket* socket)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • socket - Pointer to the socket <p>Returns:</p> <ul style="list-style-type: none"> • EXTERNAL_SocketErrorType - Error type
EXTERNAL_SocketValid	<p>bool EXTERNAL_SocketValid (EXTERNAL_Socket* socket)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • socket - Pointer to the socket <p>Returns:</p>

	<ul style="list-style-type: none"> • <code>bool</code> - true if valid, FALSE if closed or errors
EXTERNAL_SocketListen	<p>Listen and accept a connections on a socket. This function is a wrapper for EXTERNAL_SocketInitListen and EXTERNAL_SocketAccept.</p> <p>EXTERNAL_SocketErrorType EXTERNAL_SocketListen (EXTERNAL_Socket* listenSocket, int port, EXTERNAL_Socket* connectSocket)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>listenSocket</code> - Pointer to the socket to listen on • <code>port</code> - The port to listen on • <code>connectSocket</code> - Pointer to the socket that will <p>Returns:</p> <ul style="list-style-type: none"> • <code>EXTERNAL_SocketErrorType</code> - EXTERNAL_NoSocketError if successful, different error if not successful which could be due to a number of reasons.
EXTERNAL_SocketInitListen	<p>Initialize an input socket and have it listen on the given port. Call EXTERNAL_SocketAccept to accept connections on the socket. Call EXTERNAL_SocketDataAvailable to see if there is an incoming connection that has not been accepted.</p> <p>EXTERNAL_SocketErrorType EXTERNAL_SocketInitListen (EXTERNAL_Socket* listenSocket, int port)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>listenSocket</code> - Pointer to the socket to listen on • <code>port</code> - The port to listen on <p>Returns:</p> <ul style="list-style-type: none"> • <code>EXTERNAL_SocketErrorType</code> - EXTERNAL_NoSocketError if successful, different error if not successful which could be due to a number of reasons.
EXTERNAL_SocketAccept	<p>Accept a connection on a listening socket. This operation may block if there is no incoming connection, use EXTERNAL_SocketDataAvailable to check if there is an incoming connection.</p> <p>EXTERNAL_SocketErrorType EXTERNAL_SocketAccept (EXTERNAL_Socket* listenSocket, EXTERNAL_Socket* connectSocket)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>listenSocket</code> - Pointer to the socket to listen on. • <code>connectSocket</code> - Pointer to the newly created socket <p>Returns:</p> <ul style="list-style-type: none"> • <code>EXTERNAL_SocketErrorType</code> - EXTERNAL_NoSocketError if successful, different error if not successful which could be due to a number of reasons.
EXTERNAL_SocketDataAvailable	<p>Test if a socket has readable data. For a listening socket this will test for an incoming connection. For a data socket this will test if there is incoming data.</p> <p>EXTERNAL_SocketErrorType EXTERNAL_SocketDataAvailable (EXTERNAL_Socket* s, bool* available)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>s</code> - Pointer to the socket • <code>available</code> - TRUE if data is available

	<p>Returns:</p> <ul style="list-style-type: none"> • EXTERNAL_SocketErrorType - EXTERNAL_NoSocketError if successful, different error if not successful which could be due to a number of reasons.
EXTERNAL_SocketConnect	<p>Connect to a listening socket. The socket is set to non-blocking mode.</p> <p>EXTERNAL_SocketErrorType EXTERNAL_SocketConnect (EXTERNAL_Socket* socket, char* address, int port, int maxAttempts)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • socket - Pointer to the socket • address - String represent the address to connect to • port - The port to connect to • maxAttempts - Number of times to attempt connecting before an <p>Returns:</p> <ul style="list-style-type: none"> • EXTERNAL_SocketErrorType - EXTERNAL_NoSocketError if successful, different error if not successful which could be due to a number of reasons.
EXTERNAL_SocketSend	<p>Send data on a connected socket. Since the socket is non-blocking, it is possible that the send would result in a block: If the "block" parameter is FALSE, then EXTERNAL_DataNotSent is returned, and no data is sent.</p> <p>EXTERNAL_SocketErrorType EXTERNAL_SocketSend (EXTERNAL_Socket* socket, char* data, unsigned int size, bool block)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • socket - Pointer to the socket • data - Pointer to the data • size - Size of the data • block - If this call may block. Defaults to TRUE. <p>Returns:</p> <ul style="list-style-type: none"> • EXTERNAL_SocketErrorType - EXTERNAL_NoSocketError if successful, different error if not successful which could be due to a number of reasons.
EXTERNAL_SocketSend	<p>This is a wrapper for the above overloaded function.</p> <p>EXTERNAL_SocketErrorType EXTERNAL_SocketSend (EXTERNAL_Socket* socket, EXTERNAL_VarArray* data, bool block)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • socket - Pointer to the socket • data - Pointer to the VarArray to send • block - If this call may block. Defaults to TRUE. <p>Returns:</p> <ul style="list-style-type: none"> • EXTERNAL_SocketErrorType - EXTERNAL_NoSocketError if successful, different error if not successful which could be

	<p>due to a number of reasons.</p>
EXTERNAL_SocketRecv	<p>Receive data on a connected socket. Since the socket is non-blocking, it is possible that the send would result in a block: If the "block" parameter is FALSE, the "receiveSize" parameter will be set to the amount of data received before the blocking operation. This amount could be any value between 0 and size - 1.</p> <p>EXTERNAL_SocketErrorType EXTERNAL_SocketRecv (EXTERNAL_Socket* socket, char* data, unsigned int size, unsigned int* size, bool block)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>socket</code> - Pointer to the socket • <code>data</code> - Pointer to the destination • <code>size</code> - The amount of data to receive in bytes • <code>size</code> - The number of bytes received. This could be less • <code>block</code> - TRUE if the call can block, FALSE if non-blocking. <p>Returns:</p> <ul style="list-style-type: none"> • <code>EXTERNAL_SocketErrorType</code> - EXTERNAL_NoSocketError if successful, different error if not successful which could be due to a number of reasons.
EXTERNAL_SocketRecv	<p>Receive data on a UDP socket. Since the socket is non-blocking, it is possible that the send would result in a block: If the "block" parameter is FALSE, the "receiveSize" parameter will be set to the amount of data received before the blocking operation. This amount could be any value between 0 and size - 1.</p> <p>EXTERNAL_SocketErrorType EXTERNAL_SocketRecv (EXTERNAL_Socket* socket, char* data, unsigned int size, unsigned int* size, int* ip, int* port, bool block)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>socket</code> - Pointer to the socket • <code>data</code> - Pointer to the destination • <code>size</code> - The amount of data to receive in bytes • <code>size</code> - The number of bytes received. This could be less • <code>ip</code> - The IP address it was received from • <code>port</code> - The port it was received from • <code>block</code> - TRUE if the call can block, FALSE if non-blocking. <p>Returns:</p> <ul style="list-style-type: none"> • <code>EXTERNAL_SocketErrorType</code> - EXTERNAL_NoSocketError if successful, different error if not successful which could be due to a number of reasons.
EXTERNAL_SocketRecv	<p>Receive data on a connected socket. Continues reading until a '\n' character is found. This function always blocks.</p> <p>EXTERNAL_SocketErrorType EXTERNAL_SocketRecv (EXTERNAL_Socket* socket, std data)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>socket</code> - Pointer to the socket • <code>data</code> - string* <p>Returns:</p>

	<ul style="list-style-type: none"> • EXTERNAL_SocketErrorType - EXTERNAL_NoSocketError if successful, different error if not successful which could be due to a number of reasons.
EXTERNAL_SocketClose	<p>EXTERNAL_SocketErrorType EXTERNAL_SocketClose (EXTERNAL_Socket* socket)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • socket - Pointer to the socket <p>Returns:</p> <ul style="list-style-type: none"> • EXTERNAL_SocketErrorType - EXTERNAL_NoSocketError if successful, different error if not successful which could be due to a number of reasons.



Contact [QualNet Support](#) for questions pertaining to the QualNet API Reference. This document is confidential and proprietary.
It may not be reproduced or distributed without the expressed written consent of [Scalable Network Technologies](#).

QualNet® is a Registered Trademark of [Scalable Network Technologies](#).

Copyright © 2001-2012 [Scalable Network Technologies, Inc.](#) All rights reserved.

QualNet 6.1 API Reference

EXTERNAL_UTILITIES

This file describes utilities for external interfaces.

Constant / Data Structure Summary

Type	Name
ENUMERATION	ExternalScheduleType
	Enumeration of allowed scheduling operations - e.g. EXTERNAL_ActivateNode
STRUCT	EXTERNAL_TreeNode
	Structure of each node of a Splaytree
STRUCT	EXTERNAL_Tree
	Structure of a Splaytree
STRUCT	EXTERNAL_ForwardInstantiate
	Info field used for instantiating a forward app
STRUCT	EXTERNAL_ForwardSendUdpData
	Info field used for sending a UDP forward app
STRUCT	EXTERNAL_ForwardSendTcpData
	Info field used for sending a TCP forward app
STRUCT	EXTERNAL_TableRecord
	A record in the table. Contains a pointer value and a timestamp, as well as information for maintaining a linked list.
STRUCT	EXTERNAL_SimulationDurationInfo
	A duration of simulation time
STRUCT	EXTERNAL_TableOverflow

	A overflow record.
STRUCT	EXTERNAL_Table
STRUCT	A table. Generally used for storing external packet data, but can be used for anything. EXTERNAL_NetworkLayerPacket A packet that will be sent at the network layer. Created by EXTERNAL_SendDataNetworkLayer, sent by EXTERNAL_SendNetworkLayerPacket

Function / Macro Summary

Return Type	Summary
void	EXTERNAL_TreeInitialize (EXTERNAL_Tree* tree, BOOL useStore, int maxStore) To initialize the splaytree
void	SCHED_SplayTreeInsert (EXTERNAL_Tree* tree, EXTERNAL_TreeNode* treeNode) To insert a node into the Splaytree
void	EXTERNAL_TreePeekMin (EXTERNAL_Tree* tree) To look up a node in the Splaytree
void	EXTERNAL_InitializeTable (EXTERNAL_Table* table, int size) This function will initialize the table. The size parameter represents the number of records that will be allocated in one block.
void	EXTERNAL_FinalizeTable (EXTERNAL_Table* table) This function will finalize the table
EXTERNAL_TableRecord*	EXTERNAL_GetUnusedRecord (EXTERNAL_Table* table) This function will retrieve an unused record from the table. If the packet table is full it will allocate a new block of records. The user may fill in the record's contents. It will never return NULL.
EXTERNAL_TableRecord*	EXTERNAL_GetEarliestRecord (EXTERNAL_Table* table)

	This function will retrieve the earliest record in the table or NULL if the table is empty.
BOOL	EXTERNAL_GetEarliestRecord (EXTERNAL_Table* table, char* data)
	This function will check if a data pointer is still in the table.
EXTERNAL_TableRecord*	EXTERNAL_FreeRecord (EXTERNAL_Table* table)
	This function frees a record previously returned from EXTERNAL_GetUnusedRecord(). The memory contained in the data portion of the record is the user's responsibility to free.
void	EXTERNAL_SendDataAppLayerUDP (EXTERNAL_Interface* iface, NodeAddress from, NodeAddress to, char* data, int dataSize, clocktype timestamp, AppType app, TraceProtocolType trace, TosType priority)
	Sends data originating from the app layer using UDP. When the packet reaches its destination it will call the forward function of the external interface, if it exists.
void	EXTERNAL_SendDataAppLayerUDP (EXTERNAL_Interface* iface, NodeAddress from, NodeAddress to, char* header, int headerSize, int virtualDataSize, clocktype timestamp, AppType app, TraceProtocolType trace, TosType priority)
	Sends virtual data originating from the app layer using UDP. When the packet reaches its destination it will call the forward function of the external interface, if it exists.
void	EXTERNAL_SendDataAppLayerTCP (EXTERNAL_Interface* iface, NodeAddress from, NodeAddress to, char* data, int dataSize, clocktype timestamp)
	Sends data originating from the app layer using TCP. When the last byte of data reaches its destination it will call the forward function of the external interface, if it exists.
void	EXTERNAL_SendDataNetworkLayer (EXTERNAL_Interface* iface, NodeAddress from, NodeAddress srcAddr, NodeAddress destAddr, TosType tos, unsigned char protocol, unsigned int ttl, char* payload, int payloadSize, clocktype timestamp)
	Sends data originating from network layer. No provisions are made for handling this data once it enters the QualNet network. This is the responsibility of the external interface or protocols the data is sent to.
void	EXTERNAL_SendDataNetworkLayerOnInterface (EXTERNAL_Interface* iface, NodeAddress from, NodeAddress srcAddr, NodeAddress destAddr, unsigned short identification, BOOL dontFragment, BOOL moreFragments, unsigned short fragmentOffset, TosType tos, unsigned char protocol, unsigned int ttl, char* payload, int payloadSize, int interfaceIndex, clocktype timestamp)
	Sends data originating from network layer on a specific interface of the node. No provisions are made for handling this data once it enters the QualNet network. This is the responsibility of the external interface or protocols the data is sent to.
void	EXTERNAL_SendVirtualDataNetworkLayer (EXTERNAL_Interface* iface, NodeAddress from, NodeAddress srcAddr, NodeAddress destAddr, TosType tos, unsigned char protocol, unsigned int ttl, char* payload, int dataSize, int virtualSize, clocktype timestamp)
	Sends data originating from network layer. No provisions are made for handling this data once it enters the QualNet network. This is the responsibility of the external interface or protocols the data is sent to.

	responsibility of the external interface or protocols the data is sent to.
void	EXTERNAL_SendDataNetworkLayer (EXTERNAL_Interface* iface, NodeAddress from, NodeAddress srcAddr, NodeAddress destAddr, unsigned short identification, BOOL dontFragment, BOOL moreFragments, unsigned short fragmentOffset, TostType tos, unsigned char protocol, unsigned int ttl, char* payload, int payloadSize, int ipHeaderLength, char* ipOptions, clocktype timestamp)
	Sends data originating from network layer. No provisions are made for handling this data once it enters the QualNet network. This is the responsibility of the external interface or protocols the data is sent to.
void	EXTERNAL_SendNetworkLayerPacket (Node* node, Message* msg)
	Sends the packet from EXTERNAL_SendDataNetworkLayer after some delay. This function should never be called directly.
void	EXTERNAL_CreateMapping (EXTERNAL_Interface* iface, char* key, int keySize, char* value, int valueSize)
	Creates a mapping between a key and a value. The key may be any value and any length, such as an IP address, a MAC address, or a generic string. The value may be anything and is the responsibility of the user. Memory will be allocated for the key and the value.
int	EXTERNAL_ResolveMapping (EXTERNAL_Interface* iface, char* key, int keySize, char** value, int* valueSize)
	Resolves a mapping created by EXTERNAL_CreateMapping. If it exists it is placed in the value and valueSize parameters and returns 0. The returned value will point to the memory block allocated by EXTERNAL_CreateMapping. If it does not exist it returns non-zero and the value and valueSize parameters are invalid.
int	EXTERNAL_DeleteMapping (EXTERNAL_Interface* iface, char* key, int keySize)
	Deletes a mapping created by EXTERNAL_CreateMapping.
void	EXTERNAL_ActivateNode (EXTERNAL_Interface* iface, Node* node)
	Activate a node so that it can begin processing events.
void	EXTERNAL_DectivateNode (EXTERNAL_Interface* iface, Node* node)
	Dectivate a node so that it stops processing events.
void	EXTERNAL_PHY_SetTxPower (Node* node, int phyIndex, double newTxPower)
	Just like PHY_SetTxPower (), but able to handle setting transmission power when node is owned by a remote partition. Change to TxPower will be scheduled as "best-effort" for remote nodes. The range of coordinate values depends on the terrain data.
void	EXTERNAL_PHY_GetTxPower (Node* node, int phyIndex, double * txPowerPtr)
	Just like PHY_GetTxPower (), but able to handle getting transmission power when node is owned by a remote partition.
void	EXTERNAL_ChangeNodePosition (EXTERNAL_Interface* iface, Node* node, double c1, double c2, double c3)

	<p>Change the position of a node. This function will work using both coordinate systems. Orientation is not changed. Coordinate values are checked to be in the proper range, and are converted if they are not. The range of coordinate values depends on the terrain data.</p>
void	<code>EXTERNAL_ChangeNodeOrientation(EXTERNAL_Interface* iface, Node* node, short azimuth, short elevation)</code>
	<p>Change the orientation of a node. Position is not changed. Azimuth/elevation are checked to be in the proper range, and are converted if they are not.</p>
void	<code>EXTERNAL_ChangeNodePositionAndOrientation(EXTERNAL_Interface* iface, Node* node, double c1, double c2, double c3, short azimuth, short elevation)</code>
	<p>Change both the position and orientation of a node. This function will work using both coordinate systems. Coordinate values and Azimuth/elevation values are checked to be in the proper range, and are converted if they are not. The range of coordinate values depends on the terrain data.</p>
void	<code>EXTERNAL_ChangeNodePositionOrientationAndSpeedAtTime(EXTERNAL_Interface* iface, Node* node, clocktype mobilityEventTime, double c1, double c2, double c3, short azimuth, short elevation, double speed)</code>
	<p>Change the position, orientation, and speed of a node at a user-specified time. This function will work using both coordinate systems. Coordinate values, azimuth/elevation, and speed values are checked to be in the proper range, and are converted if they are not. The range of coordinate values depends on the terrain data.</p>
void	<code>EXTERNAL_ChangeNodePositionOrientationAndVelocityAtTime(EXTERNAL_Interface* iface, Node* node, clocktype mobilityEventTime, double c1, double c2, double c3, short azimuth, short elevation, double speed, double c1Speed, double c2Speed, double c3Speed)</code>
	<p>Update the position, orientation, and velocity vector of a node at a user-specified time. The velocity vector is expected to be in the same distance units used for the the position, per one second. The speed parameter must also be provided, accurate for the provided velocity vector, and always in meters per second. Coordinate values, azimuth/elevation, and speed values are checked to be in the proper range, and are converted if they are not. The range of coordinate values depends on the terrain data.</p>
void	<code>EXTERNAL_ChangeNodePositionOrientationAndVelocityAtTime(EXTERNAL_Interface* iface, Node* node, clocktype mobilityEventTime, double c1, double c2, double c3, short azimuth, short elevation, double c1Speed, double c2Speed, double c3Speed)</code>
	<p>Update the position, orientation, and velocity vector of a node at a user-specified time. The velocity vector is expected to be in the same distance units used for the the position, per one second. Coordinate values, azimuth/elevation, and speed values are checked to be in the proper range, and are converted if they are not. The range of coordinate values depends on the terrain data.</p>
void	<code>EXTERNAL_ChangeNodeVelocityAtTime(EXTERNAL_Interface* iface, Node* node, clocktype mobilityEventTime, double speed, double c1Speed, double c2Speed, double c3Speed)</code>
	<p>Update the velocity vector of a node at a user-specified time. The velocity vector is expected to be in the same distance units used for the terrain, per one second. The speed parameter must also be provided, accurate for the provided velocity vector, and always in meters per second.</p>
void	<code>EXTERNAL_ChangeNodeVelocityAtTime(EXTERNAL_Interface* iface, Node* node, clocktype mobilityEventTime, double c1Speed, double c2Speed, double c3Speed)</code>

	Update the velocity vector of a node at a user-specified time. The velocity vector is expected to be in the same distance units used for the terrain, per one second.
BOOL	EXTERNAL_ConfigStringPresent (NodeInput* nodeInput, char* string)
	This function will check the config file for a string. Typically this is used during interface registration to see if the interface is turned on in the config file.
BOOL	EXTERNAL_ConfigStringIsYes (NodeInput* nodeInput, char* string)
	This function will check the config file for a string. Typically this is used during interface registration to see if the interface is turned on in the config file. Checks that the string is YES.
void	EXTERNAL_MESSAGE_RemoteSend (EXTERNAL_Interface* iface, int destinationPartitionId, Message * msg, clocktype delay, ExternalScheduleType scheduling)
	Send a message to the external interface on a different partition. This function makes it possible for your external interface to send a message to your external interface that is on on a different/remote partition. You will then need to add your message handler into the function EXTERNAL_ProcessEvent(). Lastly, you can request a best-effort delivery of your message to the remote external interface by passing in a delay value of 0 and a scheduling type of EXTERNAL_SCHEDULE_LOOSELY. Be aware that best-effort messages may be scheduled at slightly different simulation times each time you run your simulation. Further notes about scheduling. If your external event won't result in additional qualnet events, except those that will be scheduled after safe time, then you can use LOOSELY. If, your event is going to schedule additional qualnet event though, then you must use EXTERNAL_SCHEDULE_SAFE (so that the event is delayed to the next safe time). If you violate safe time you will get assertion failures for safe time of signal receive time.
void	EXTERNAL_SetSimulationEndTime (partitionData* partitionData, clocktype endTime)
	This function is a means to programatically set the end of the simulation. The endTime argument can be omitted, in which case the endTime is the current simulation time. If the requested time has already passed, the simulation will end as soon as possible.
clocktype	EXTERNAL_QueryRealTime ()
	This function will return the wall clock time in the qualnet time format. NOTE: Interfaces that are running in real-time should not use this function to check the simulation time. The simulation time will not be the same as real time if the simulation was paused. Use the interface's time function instead.
clocktype	EXTERNAL_QueryRealTime ()
	This function will return the wall clock time in the qualnet time format. NOTE: Interfaces that are running in real-time should not use this function to check the simulation time. The simulation time will not be the same as real time if the simulation was paused. Use the interface's time function instead.
clocktype	EXTERNAL_QueryCPUTime (EXTERNAL_Interface* iface)
	This function will return the amount of Cpu time used by QualNet. The first call to this function will by an interface will return 0, and timing will begin from that point.
void	EXTERNAL_Sleep (clocktype amount)

This function will sleep for a minimum amount of time as indicated by the amount parameter. Depending on which platform it is called on the amount of time spent sleeping could be greater.

Constant / Data Structure Detail

Enumeration	ExternalScheduleType Enumeration of allowed scheduling operations - e.g. EXTERNAL_ActivateNode
Structure	EXTERNAL_TreeNode Structure of each node of a Splaytree
Structure	EXTERNAL_Tree Structure of a Splaytree
Structure	EXTERNAL_ForwardInstantiate Info field used for instantiating a forward app
Structure	EXTERNAL_ForwardSendUdpData Info field used for sending a UDP forward app
Structure	EXTERNAL_ForwardSendTcpData Info field used for sending a TCP forward app
Structure	EXTERNAL_TableRecord A record in the table. Contains a pointer value and a timestamp, as well as information for maintaining a linked list.
Structure	EXTERNAL_SimulationDurationInfo A duration of simulation time
Structure	EXTERNAL_TableOverflow

	A overflow record.
Structure	EXTERNAL_Table
Structure	A table. Generally used for storing external packet data, but can be used for anything. EXTERNAL_NetworkLayerPacket A packet that will be sent at the network layer. Created by EXTERNAL_SendDataNetworkLayer, sent by EXTERNAL_SendNetworkLayerPacket

Function / Macro Detail

Function / Macro	Format
EXTERNAL_TreeInitialize To initialize the splaytree	<p>void EXTERNAL_TreeInitialize (EXTERNAL_Tree* tree, BOOL useStore, int maxStore)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • tree - Pointer to the splaytree • useStore - Use Store • maxStore - Max Store <p>Returns:</p> <ul style="list-style-type: none"> • void - None
SCHED_SplayTreeInsert To insert a node into the Splaytree	<p>void SCHED_SplayTreeInsert (EXTERNAL_Tree* tree, EXTERNAL_TreeNode* treeNode)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • tree - Pointer to the splaytree • treeNode - Pointer to the splayNode <p>Returns:</p> <ul style="list-style-type: none"> • void - None
EXTERNAL_TreePeekMin To look up a node in the Splaytree	<p>void EXTERNAL_TreePeekMin (EXTERNAL_Tree* tree)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • tree - Pointer to the splaytree <p>Returns:</p>

	<ul style="list-style-type: none"> • void - None
EXTERNAL_InitializeTable	<p>This function will initialize the table. The size parameter represents the number of records that will be allocated in one block.</p> <p>void EXTERNAL_InitializeTable (EXTERNAL_Table* table, int size)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • table - The table • size - The size of the table <p>Returns:</p> <ul style="list-style-type: none"> • void - None
EXTERNAL_FinalizeTable	<p>This function will finalize the table</p> <p>void EXTERNAL_FinalizeTable (EXTERNAL_Table* table)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • table - The table <p>Returns:</p> <ul style="list-style-type: none"> • void - None
EXTERNAL_GetUnusedRecord	<p>This function will retrieve an unused record from the table. If the packet table is full it will allocate a new block of records. The user may fill in the record's contents. It will never return NULL.</p> <p>EXTERNAL_TableRecord* EXTERNAL_GetUnusedRecord (EXTERNAL_Table* table)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • table - The table <p>Returns:</p> <ul style="list-style-type: none"> • EXTERNAL_TableRecord* - The retrieved record
EXTERNAL_GetEarliestRecord	<p>This function will retrieve the earliest record in the table or NULL if the table is empty.</p> <p>EXTERNAL_TableRecord* EXTERNAL_GetEarliestRecord (EXTERNAL_Table* table)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • table - The table <p>Returns:</p> <ul style="list-style-type: none"> • EXTERNAL_TableRecord* - The retrieved record
EXTERNAL_GetEarliestRecord	<p>This function will check if a data pointer is still in the table.</p> <p>BOOL EXTERNAL_GetEarliestRecord (EXTERNAL_Table* table, char* data)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • table - The table • data - The data to check for <p>Returns:</p>

	<ul style="list-style-type: none"> • <code>BOOL</code> - TRUE if it is in the table, FALSE if not
EXTERNAL_FreeRecord	<p>This function frees a record previously returned from <code>EXTERNAL_GetUnusedRecord()</code>. The memory contained in the data portion of the record is the user's responsibility to free.</p> <p><code>EXTERNAL_TableRecord* EXTERNAL_FreeRecord (EXTERNAL_Table* table)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>table</code> - The table <p>Returns:</p> <ul style="list-style-type: none"> • <code>EXTERNAL_TableRecord*</code> - The retrieved record
EXTERNAL_SendDataAppLayerUDP	<p>Sends data originating from the app layer using UDP. When the packet reaches its destination it will call the forward function of the external interface, if it exists.</p> <p><code>void EXTERNAL_SendDataAppLayerUDP (EXTERNAL_Interface* iface, NodeAddress from, NodeAddress to, char* data, int dataSize, clocktype timestamp, AppType app, TraceProtocolType trace, TosType priority)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>iface</code> - The external interface • <code>from</code> - The address of the sending node • <code>to</code> - The address of the receiving node • <code>data</code> - The data that is to be sent. This may be NULL if there • <code>dataSize</code> - The size of the data • <code>timestamp</code> - The time to send this message. Pass 0 to send • <code>app</code> - The application to send to, defaults to APP_FORWARD • <code>trace</code> - The trace protocol, defaults to TRACE_FORWARD • <code>priority</code> - The priority to send this message at <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
EXTERNAL_SendDataAppLayerUDP	<p>Sends virtual data originating from the app layer using UDP. When the packet reaches its destination it will call the forward function of the external interface, if it exists.</p> <p><code>void EXTERNAL_SendDataAppLayerUDP (EXTERNAL_Interface* iface, NodeAddress from, NodeAddress to, char* header, int headerSize, int virtualDataSize, clocktype timestamp, AppType app, TraceProtocolType trace, TosType priority)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>iface</code> - The external interface • <code>from</code> - The address of the sending node • <code>to</code> - The address of the receiving node • <code>header</code> - The header that is to be sent. • <code>headerSize</code> - The size of the header

	<ul style="list-style-type: none"> <code>virtualDataSize</code> - The size of the virtual data <code>timestamp</code> - The time to send this message. Pass 0 to send <code>app</code> - The application to send to, defaults to APP_FORWARD <code>trace</code> - The trace protocol, defaults to TRACE_FORWARD <code>priority</code> - The priority to send this message at. defaults to IPTOS_PREC_ROUTINE <p>Returns:</p> <ul style="list-style-type: none"> <code>void</code> - None
EXTERNAL_SendDataAppLayerTCP	<p>Sends data originating from the app layer using TCP. When the last byte of data reaches its destination it will call the forward function of the external interface, if it exists.</p> <p><code>void EXTERNAL_SendDataAppLayerTCP (EXTERNAL_Interface* iface, NodeAddress from, NodeAddress to, char* data, int dataSize, clocktype timestamp)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <code>iface</code> - The external interface <code>from</code> - The address of the sending node <code>to</code> - The address of the receiving node <code>data</code> - The data that is to be sent. This may be NULL if there <code>dataSize</code> - The size of the data <code>timestamp</code> - The time to send this message. Pass 0 to send <p>Returns:</p> <ul style="list-style-type: none"> <code>void</code> - None
EXTERNAL_SendDataNetworkLayer	<p>Sends data originating from network layer. No provisions are made for handling this data once it enters the QualNet network. This is the responsibility of the external interface or protocols the data is sent to.</p> <p><code>void EXTERNAL_SendDataNetworkLayer (EXTERNAL_Interface* iface, NodeAddress from, NodeAddress srcAddr, NodeAddress destAddr, TosType tos, unsigned char protocol, unsigned int ttl, char* payload, int payloadSize, clocktype timestamp)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <code>iface</code> - The external interface <code>from</code> - The address of the node that will send the <code>srcAddr</code> - The IP address of the node originally <code>destAddr</code> - The address of the receiving node <code>tos</code> - The Type of Service field in the IP header <code>protocol</code> - The protocol field in the IP header <code>ttl</code> - The Time to Live field in the IP header

	<ul style="list-style-type: none"> • payload - The data that is to be sent. This should include • payloadSize - The size of the data • timestamp - The time to send this packet. Pass 0 to send <p>Returns:</p> <ul style="list-style-type: none"> • void - None
EXTERNAL_SendDataNetworkLayerOnInterface	<p>Sends data originating from network layer on a specific interface of the node. No provisions are made for handling this data once it enters the QualNet network. This is the responsibility of the external interface or protocols the data is sent to.</p> <p>void EXTERNAL_SendDataNetworkLayerOnInterface (EXTERNAL_Interface* iface, NodeAddress from, NodeAddress srcAddr, NodeAddress destAddr, unsigned short identification, BOOL dontFragment, BOOL moreFragments, unsigned short fragmentOffset, TosType tos, unsigned char protocol, unsigned int ttl, char* payload, int payloadSize, int interfaceIndex, clocktype timestamp)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • iface - The external interface • from - The address of the node that will send the • srcAddr - The IP address of the node originally • destAddr - The address of the receiving node • identification - The identification field in the IP • dontFragment - Whether to set the dont fragment bit in the IP • moreFragments - Whether to set the more fragments bit in the IP • fragmentOffset - The fragment offset field in the IP • tos - The Type of Service field in the IP header • protocol - The protocol field in the IP header • ttl - The Time to Live field in the IP header • payload - The data that is to be sent. This should include • payloadSize - The size of the data • interfaceIndex - The interface index • timestamp - The time to send this packet. Pass 0 to send <p>Returns:</p> <ul style="list-style-type: none"> • void - None
EXTERNAL_SendVirtualDataNetworkLayer	<p>void EXTERNAL_SendVirtualDataNetworkLayer (EXTERNAL_Interface* iface, NodeAddress from, NodeAddress srcAddr, NodeAddress destAddr, TosType tos, unsigned char protocol, unsigned int ttl, char* payload, int dataSize, int virtualSize, clocktype timestamp)</p>

Sends data originating from network layer. No provisions are made for handling this data once it enters the QualNet network. This is the responsibility of the external interface or protocols the data is sent to.

Parameters:

- `iface` - The external interface
- `from` - The address of the node that will send the
- `srcAddr` - The IP address of the node originally
- `destAddr` - The address of the receiving node
- `tos` - The Type of Service field in the IP header
- `protocol` - The protocol field in the IP header
- `ttl` - The Time to Live field in the IP header
- `payload` - The data that is to be sent. This should include
- `dataSize` - The size of the data
- `virtualSize` - The size of the virtual data
- `timestamp` - The time to send this packet. Pass 0 to send

Returns:

- `void` - None

EXTERNAL_SendDataNetworkLayer

Sends data originating from network layer. No provisions are made for handling this data once it enters the QualNet network. This is the responsibility of the external interface or protocols the data is sent to.

```
void EXTERNAL_SendDataNetworkLayer (EXTERNAL_Interface* iface, NodeAddress from,
NodeAddress srcAddr, NodeAddress destAddr, unsigned short identification, BOOL dontFragment,
BOOL moreFragments, unsigned short fragmentOffset, TosType tos, unsigned char protocol, unsigned
int ttl, char* payload, int payloadSize, int ipHeaderLength, char* ipOptions, clocktype timestamp)
```

Parameters:

- `iface` - The external interface
- `from` - The address of the node that will send the
- `srcAddr` - The IP address of the node originally
- `destAddr` - The address of the receiving node
- `identification` - The identification field in the IP
- `dontFragment` - Whether to set the dont fragment bit in the IP
- `moreFragments` - Whether to set the more fragments bit in the IP
- `fragmentOffset` - The fragment offset field in the IP
- `tos` - The Type of Service field in the IP header
- `protocol` - The protocol field in the IP header

	<ul style="list-style-type: none"> • <code>ttl</code> - The Time to Live field in the IP header • <code>payload</code> - The data that is to be sent. This should include • <code>payloadSize</code> - The size of the data • <code>ipHeaderLength</code> - length of the IP Header including options if any • <code>ipOptions</code> - pointer to the IP Option. • <code>timestamp</code> - The time to send this packet. Pass 0 to send <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
EXTERNAL_SendNetworkLayerPacket	<p>Sends the packet from EXTERNAL_SendDataNetworkLayer after some delay. This function should never be called directly.</p> <p>void EXTERNAL_SendNetworkLayerPacket (Node* node, Message* msg)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - The node sending the packet • <code>msg</code> - The message <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
EXTERNAL_CreateMapping	<p>Creates a mapping between a key and a value. The key may be any value and any length, such as an IP address, a MAC address, or a generic string. The value may be anything and is the responsibility of the user. Memory will be allocated for the key and the value.</p> <p>void EXTERNAL_CreateMapping (EXTERNAL_Interface* iface, char* key, int keySize, char* value, int valueSize)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>iface</code> - The external interface • <code>key</code> - The address of the key • <code>keySize</code> - The size of the key in bytes • <code>value</code> - The address of what the value maps to • <code>valueSize</code> - The size of the value in bytes <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
EXTERNAL_ResolveMapping	<p>Resolves a mapping created by EXTERNAL_CreateMapping. If it exists it is placed in the value and valueSize parameters and returns 0. The returned value will point to the memory block allocated by EXTERNAL_CreateMapping. If it does not exist it returns non-zero</p> <p>int EXTERNAL_ResolveMapping (EXTERNAL_Interface* iface, char* key, int keySize, char** value, int* valueSize)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>iface</code> - The external interface • <code>key</code> - Pointer to the key

and the value and valueSize parameters are invalid.

- `keySize` - The size of the key in bytes
- `value` - Pointer to the value (output)
- `valueSize` - The size of the key in bytes (output)

Returns:

- `int` - 0 if the mapping resolved, non-zero if it did not

EXTERNAL_DeleteMapping

Deletes a mapping created by EXTERNAL_CreateMapping.

int **EXTERNAL_DeleteMapping** (EXTERNAL_Interface* iface, char* key, int keySize)

Parameters:

- `iface` - The external interface
- `key` - Pointer to the key
- `keySize` - The size of the key in bytes

Returns:

- `int` - 0 if the mapping resolved, non-zero if it did not

EXTERNAL_ActivateNode

Activate a node so that it can begin processing events.

void **EXTERNAL_ActivateNode** (EXTERNAL_Interface* iface, Node* node)

Parameters:

- `iface` - The external interface
- `node` - The node

Returns:

- `void` - None

EXTERNAL_DectivateNode

Dectivate a node so that it stops processing events.

void **EXTERNAL_DectivateNode** (EXTERNAL_Interface* iface, Node* node)

Parameters:

- `iface` - The external interface
- `node` - The node

Returns:

- `void` - None

EXTERNAL_PHY_SetTxPower

Just like PHY_SetTxPower (), but able to handle setting transmission power when node is owned by a remote partition. Change to TxPower

void **EXTERNAL_PHY_SetTxPower** (Node* node, int phyIndex, double newTxPower)

Parameters:

- `node` - The node (can be either a local node or remote)

	<p>will be scheduled as "best-effort" for remote nodes. The range of coordinate values depends on the terrain data.</p> <ul style="list-style-type: none"> • <code>phyIndex</code> - The physical index • <code>newTxPower</code> - The new transmission power. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
EXTERNAL_PHY_GetTxPower	<p>Just like <code>PHY_GetTxPower()</code>, but able to handle getting transmission power when node is owned by a remote partition.</p> <p><code>void EXTERNAL_PHY_GetTxPower (Node* node, int phyIndex, double * txPowerPtr)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - The node (can be either a local node or remote) • <code>phyIndex</code> - The physical index • <code>txPowerPtr</code> - (OUT) value of transmission power will be <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
EXTERNAL_ChangeNodePosition	<p>Change the position of a node. This function will work using both coordinate systems. Orientation is not changed. Coordinate values are checked to be in the proper range, and are converted if they are not. The range of coordinate values depends on the terrain data.</p> <p><code>void EXTERNAL_ChangeNodePosition (EXTERNAL_Interface* iface, Node* node, double c1, double c2, double c3)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>iface</code> - The external interface • <code>node</code> - The node • <code>c1</code> - The first coordinate • <code>c2</code> - The second coordinate • <code>c3</code> - The third coordinate <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
EXTERNAL_ChangeNodeOrientation	<p>Change the orientation of a node. Position is not changed. Azimuth/elevation are checked to be in the proper range, and are converted if they are not.</p> <p><code>void EXTERNAL_ChangeNodeOrientation (EXTERNAL_Interface* iface, Node* node, short azimuth, short elevation)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>iface</code> - The external interface • <code>node</code> - The node • <code>azimuth</code> - The azimuth, $0 \leqslant \text{azimuth} \leqslant 359$ • <code>elevation</code> - The elevation, $-180 \leqslant \text{elevation} \leqslant 180$ <p>Returns:</p>

	<ul style="list-style-type: none"> • <code>void</code> - None
EXTERNAL_ChangeNodePositionAndOrientation	<p>Change both the position and orientation of a node. This function will work using both coordinate systems. Coordinate values and Azimuth/elevation values are checked to be in the proper range, and are converted if they are not. The range of coordinate values depends on the terrain data.</p> <p>void EXTERNAL_ChangeNodePositionAndOrientation (EXTERNAL_Interface* iface, Node* node, double c1, double c2, double c3, short azimuth, short elevation)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>iface</code> - The external interface • <code>node</code> - The node • <code>c1</code> - The first coordinate • <code>c2</code> - The second coordinate • <code>c3</code> - The third coordinate • <code>azimuth</code> - The azimuth, $0 \leqslant \text{azimuth} \leqslant 359$ • <code>elevation</code> - The elevation, $-180 \leqslant \text{elevation} \leqslant 180$ <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
EXTERNAL_ChangeNodePositionOrientationAndSpeedAtTime	<p>Change the position, orientation, and speed of a node at a user-specified time. This function will work using both coordinate systems. Coordinate values, azimuth/elevation, and speed values are checked to be in the proper range, and are converted if they are not. The range of coordinate values depends on the terrain data.</p> <p>void EXTERNAL_ChangeNodePositionOrientationAndSpeedAtTime (EXTERNAL_Interface* iface, Node* node, clocktype mobilityEventTime, double c1, double c2, double c3, short azimuth, short elevation, double speed)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>iface</code> - The external interface • <code>node</code> - The node • <code>mobilityEventTime</code> - The absolute simulation time (not delay) • <code>c1</code> - The first coordinate • <code>c2</code> - The second coordinate • <code>c3</code> - The third coordinate • <code>azimuth</code> - The azimuth, $0 \leqslant \text{azimuth} \leqslant 359$ • <code>elevation</code> - The elevation, $-180 \leqslant \text{elevation} \leqslant 180$ • <code>speed</code> - The speed in m/s <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
EXTERNAL_ChangeNodePositionOrientationAndVelocityAtTime	void EXTERNAL_ChangeNodePositionOrientationAndVelocityAtTime (EXTERNAL_Interface* iface,

Update the position, orientation, and velocity vector of a node at a user-specified time. The velocity vector is expected to be in the same distance units used for the the position, per one second. The speed parameter must also be provided, accurate for the provided velocity vector, and always in meters per second. Coordinate values, azimuth/elevation, and speed values are checked to be in the proper range, and are converted if they are not. The range of coordinate values depends on the terrain data.

`Node* node, clocktype mobilityEventTime, double c1, double c2, double c3, short azimuth, short elevation, double speed, double c1Speed, double c2Speed, double c3Speed)`

Parameters:

- `iface` - The external interface
- `node` - The node
- `mobilityEventTime` - The absolute simulation time (not delay)
- `c1` - The first coordinate
- `c2` - The second coordinate
- `c3` - The third coordinate
- `azimuth` - The azimuth, $0 \leqslant \text{azimuth} \leqslant 359$
- `elevation` - The elevation, $-180 \leqslant \text{elevation} \leqslant 180$
- `speed` - The speed in m/s
- `c1Speed` - The rate of change of the first coordinate in the
- `c2Speed` - The rate of change of the second coordinate in the
- `c3Speed` - The rate of change of the third coordinate in the

Returns:

- `void` - None

EXTERNAL_ChangeNodePositionOrientationAndVelocityAtTime

Update the position, orientation, and velocity vector of a node at a user-specified time. The velocity vector is expected to be in the same distance units used for the the position, per one second. Coordinate values, azimuth/elevation, and speed values are checked to be in the proper range, and are converted if they are not. The range of coordinate values depends on the terrain data.

`void EXTERNAL_ChangeNodePositionOrientationAndVelocityAtTime (EXTERNAL_Interface* iface, Node* node, clocktype mobilityEventTime, double c1, double c2, double c3, short azimuth, short elevation, double c1Speed, double c2Speed, double c3Speed)`

Parameters:

- `iface` - The external interface
- `node` - The node
- `mobilityEventTime` - The absolute simulation time (not delay)
- `c1` - The first coordinate
- `c2` - The second coordinate
- `c3` - The third coordinate
- `azimuth` - The azimuth, $0 \leqslant \text{azimuth} \leqslant 359$
- `elevation` - The elevation, $-180 \leqslant \text{elevation} \leqslant 180$

	<p><code>c1Speed</code> - The rate of change of the first coordinate in the</p> <ul style="list-style-type: none"> • <code>c2Speed</code> - The rate of change of the second coordinate in the • <code>c3Speed</code> - The rate of change of the third coordinate in the <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
EXTERNAL_ChangeNodeVelocityAtTime	<p>Update the velocity vector of a node at a user-specified time. The velocity vector is expected to be in the same distance units used for the terrain, per one second. The speed parameter must also be provided, accurate for the provided velocity vector, and always in meters per second.</p> <p><code>void EXTERNAL_ChangeNodeVelocityAtTime (EXTERNAL_Interface* iface, Node* node, clocktype mobilityEventTime, double speed, double c1Speed, double c2Speed, double c3Speed)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>iface</code> - The external interface • <code>node</code> - The node • <code>mobilityEventTime</code> - The absolute simulation time (not delay) • <code>speed</code> - The speed in m/s • <code>c1Speed</code> - The rate of change of the first coordinate in the • <code>c2Speed</code> - The rate of change of the second coordinate in the • <code>c3Speed</code> - The rate of change of the third coordinate in the <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
EXTERNAL_ChangeNodeVelocityAtTime	<p>Update the velocity vector of a node at a user-specified time. The velocity vector is expected to be in the same distance units used for the terrain, per one second.</p> <p><code>void EXTERNAL_ChangeNodeVelocityAtTime (EXTERNAL_Interface* iface, Node* node, clocktype mobilityEventTime, double c1Speed, double c2Speed, double c3Speed)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>iface</code> - The external interface • <code>node</code> - The node • <code>mobilityEventTime</code> - The absolute simulation time (not delay) • <code>c1Speed</code> - The rate of change of the first coordinate in the • <code>c2Speed</code> - The rate of change of the second coordinate in the • <code>c3Speed</code> - The rate of change of the third coordinate in the <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
EXTERNAL_ConfigStringPresent	<code>BOOL EXTERNAL_ConfigStringPresent (NodeInput* nodeInput, char* string)</code>

	<p>This function will check the config file for a string. Typically this is used during interface registration to see if the interface is turned on in the config file.</p>
EXTERNAL_ConfigStringIsYes	<p>BOOL EXTERNAL_ConfigStringIsYes (NodeInput* nodeInput, char* string)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>nodeInput</code> - The configuration file • <code>string</code> - The string to check for <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - TRUE if the string is present, FALSE otherwise
EXTERNAL_MESSAGE_RemoteSend	<p>Send a message to the external interface on a different partition. This function makes it possible for your external interface to send a message to your external interface that is on on a different/remote partition. You will then need to add your message handler into the function <code>EXTERNAL_ProcessEvent ()</code>. Lastly, you can request a best-effort delivery of your message to the remote external interface by passing in a delay value of 0 and a scheduling type of <code>EXTERNAL_SCHEDULE_LOOSELY</code>. Be aware that best-effort messages may be scheduled at slightly different simulation times each time you run your simulation. Further notes about scheduling. If your external event won't result in additional qualnet events, except those that will be scheduled after safe time, then you can use <code>LOOSELY</code>. If, your event is going to schedule additional qualnet event though, then you must use <code>EXTERNAL_SCHEDULE_SAFE</code> (so that the event is delayed to the next safe time). If you violate safe time you will get assertion failures for safe time of signal receive time.</p>
EXTERNAL_SetSimulationEndTime	<p>This function is a means to programatically set the end of the simulation. The <code>endTime</code> argument can be omitted, in which case the <code>endTime</code> is the current simulation time. If the requested time has already passed, the simulation will end as soon as possible.</p> <p>void EXTERNAL_SetSimulationEndTime (partitionData* partitionData, clocktype endTime)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>partitionData</code> - pointer to data for this partition • <code>endTime</code> - The simulation time to end at. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None

EXTERNAL_QueryRealTime	<p>This function will return the wall clock time in the qualnet time format. NOTE: Interfaces that are running in real-time should not use this function to check the simulation time. The simulation time will not be the same as real time if the simulation was paused. Use the interface's time function instead.</p>
EXTERNAL_QueryRealTime	<p>This function will return the wall clock time in the qualnet time format. NOTE: Interfaces that are running in real-time should not use this function to check the simulation time. The simulation time will not be the same as real time if the simulation was paused. Use the interface's time function instead.</p>
EXTERNAL_QueryCPUTime	<p>This function will return the amount of Cpu time used by QualNet. The first call to this function will by an interface will return 0, and timing will begin from that point.</p>
EXTERNAL_Sleep	<p>This function will sleep for a minimum amount of time as indicated by the amount parameter. Depending on which platform it is called on the amount of time spent sleeping could be greater.</p>



Contact [QualNet Support](#) for questions pertaining to the QualNet API Reference. This document is confidential and proprietary.
It may not be reproduced or distributed without the expressed written consent of [Scalable Network Technologies](#).

QualNet® is a Registered Trademark of [Scalable Network Technologies](#).

QualNet 6.1 API Reference

FILEIO

This file describes data structures and functions used for reading from input files and printing to output files.

Constant / Data Structure Summary

Type	Name
CONSTANT	ANY_NODEID
	Optional macro values to use when calling IO_Read...() APIs. Defines any node id.
CONSTANT	ANY_ADDRESS
	Optional macro values to use when calling IO_Read...() APIs. Defines any node address.
CONSTANT	ANY_INSTANCE
	Optional macro values to use when calling IO_Read...() APIs. Defines any instance.
CONSTANT	MAX_INPUT_FILE_LINE_LENGTH
	Maximum input file line length. Evaluates (6 * MAX_STRING_LENGTH)
CONSTANT	MAX_ADDRESS_STRING_LENGTH
	Maximum length of address string.
CONSTANT	MAX_NUM_CACHED_FILES
	Max number of -FILE references in an input file. (Restriction is only for immediate children)
CONSTANT	MATCH_NODE_ID
	Defines the matching by node id.
CONSTANT	MATCH_NETWORK
	Defines the matching by network.
CONSTANT	MATCH_INTERFACE

	Defines the matching by interface.
CONSTANT	INPUT_ALLOCATION_UNIT
STRUCT	<p>Defines input allocation unit.</p> <p>NodeInput</p> <p>Definition of node input structure. typedef to NodeInput in include/main.h.</p>

Function / Macro Summary

Return Type	Summary
void	<p>IO_ConvertIpAddressToString(NodeAddress ipAddress, char* addressString)</p> <p>Parses IPv4 address into a dotted-decimal string.</p>
int	<p>IO_FindStringPos(const char s[], const char subString[])</p> <p>Returns the index of the first subString found in s.</p>
char*	<p>IO_GetToken(char* dst, const char* src, char ** next)</p> <p>Searches source buffer for the first %s-style token encountered, and copies it to dst.</p>
char*	<p>IO_GetDelimitedToken(char* dst, const char* src, const char* delim, char** next)</p> <p>Searches source buffer for the first delimited token encountered, and copies it to dst.</p>
const char*	<p>IO_Right(const char * s, unsigned count)</p> <p>Returns a pointer to the right side of the string of length "count" characters.</p>
char*	<p>IO_Chop(const char* s)</p> <p>Removes the last character of string.</p>
void	<p>IO_TrimNsbpSpaces(char* s)</p> <p>Changes nsbp characters for UTF-8 encoding to spaces.</p>

void	<u>IO_TrimLeft</u> (char* s)	Strips leading white space from a string (by memmove(jing string contents left).
void	<u>IO_TrimRight</u> (char* s)	Strips trailing white space from a string (by inserting early NULL).
void	<u>IO_CompressWhiteSpace</u> (char* s)	Compresses white space between words in the string to one space in a string. White space at the very beginning and very end of the string is also compressed to one space -- not stripped entirely.
BOOL	<u>IO_IsStringNonNegativeInteger</u> (const char* s)	Returns TRUE if every character in string is a digit. (Even white space will cause return of FALSE)
void	<u>IO_ConvertStringToLowercase</u> (char s[])	Runs tolower() on each character in string and converts the same to lowercase.
void	<u>IO_ConvertStringToUpperCase</u> (char s[])	Runs toupper() on each character in string and converts the same to uppercase.
BOOL	<u>IO_CaseInsensitiveStringsAreEqual</u> (const char[] s1, const char[] s2, char lengthToCompare)	Checks two strings are equal or not ignoring case.
BOOL	<u>IO_BankLine</u> (char s[])	Checks the blank line/string.
BOOL	<u>IO_CommentLine</u> (char s[])	Checks whether the line is a comment(i.e. starts with '#').
int	<u>IO_FindCaseInsensitiveStringPos</u> (const char s[], const char subString[])	Finds the case insensitive sub string position in a string.
int	<u>IO_FindCaseInsensitiveStringPos</u> (const char s[], const char subString[])	Finds the case insensitive sub string position in a string.

	<code>IO_SkipToken.</code> (char* token, char* tokenSep, char* skip)
	skip the first n tokens.
NodeInput *	<code>IO_CreateNodeInput</code> (NodeInput* nodeInput, const char* filename)
	Allocates a NodeInput datastructure that can then be passed to IO_ReadNodeInput Called for each file variable in the config file.
void	<code>IO_InitializeNodeInput</code> (NodeInput* nodeInput)
	Initializes a NodeInput structure
void	<code>IO_ReadNodeInput</code> (NodeInput* nodeInput, const char* filename)
	Reads an input file into a NodeInput struct. Calls IO_ReadFileParameters to first read in -FILE paramters. Then calls IO_ReadFileParameters to read the rest of the parameters.
void	<code>IO_ReadNodeInputEx</code> (NodeInput* nodeInput, const char* filename, const char* includeComment)
	Reads an input file into a NodeInput struct. The includeComment Flag facilitate whether to include the commented line lines in the nodeInput structure or not. The commented lines should only be included in Backward Compatibility for Old scenario exe. This exe is responsible for creating new config file and router model file. These new files contains changes according to current VERSION of QualNet. Calls IO_ReadFileParameters to first read in -FILE paramters. Then calls IO_ReadFileParameters to read the rest of the parameters.
BOOL	<code>IO_ConvertFile</code> (NodeInput* nodeInput, NodeInput* nodeOutput, char* version)
	Converts the contents of an old configuration file to the latest version.
void	<code>IO_ReadLine</code> (const NodeAddress nodeId, const NodeAddress interfaceAddress, const NodeInput* nodeInput, const char* index, BOOL* wasFound, char* readVal)
	This API is used to retrieve a whole line from input files.
void	<code>IO_ReadString</code> (const NodeAddress nodeId, const NodeAddress interfaceAddress, const NodeInput* nodeInput, const char* index, BOOL* wasFound, char* readVal)
	This API is used to retrieve a string parameter value from input files.
void	<code>IO_ReadBool</code> (const NodeAddress nodeId, const NodeAddress interfaceAddress, const NodeInput* nodeInput, const char* index, BOOL* wasFound, BOOL* readVal)
	This API is used to retrieve a boolean parameter value from input files.
void	<code>IO_ReadInt</code> (const NodeAddress nodeId, const NodeAddress interfaceAddress, const NodeInput* nodeInput, const char* index, BOOL* wasFound, int* readVal)

	This API is used to retrieve an integer parameter value from input files.
void	<code>IO_ReadDouble</code> (const NodeAddress nodeId, const NodeAddress interfaceAddress, const NodeInput* nodeInput, const char* index, BOOL* wasFound, double* readVal)
	This API is used to retrieve a double parameter value from input files.
void	<code>IO_ReadFloat</code> (const NodeAddress nodeId, const NodeAddress interfaceAddress, const NodeInput* nodeInput, const char* index, BOOL* wasFound, float* readVal)
	This API is used to retrieve a float parameter value from input files.
void	<code>IO_ReadTime</code> (const NodeAddress nodeId, const NodeAddress interfaceAddress, const NodeInput* nodeInput, const char* index, BOOL* wasFound, clocktype* readVal)
	This API is used to retrieve time parameter value from input files.
void	<code>IO_ReadCachedFile</code> (const NodeAddress nodeId, const NodeAddress interfaceAddress, const NodeInput* nodeInput, const char* index, BOOL* wasFound, NodeInput* parameterValue)
	This API is used to retrieve cached file parameter value from input files.
void	<code>IO_ReadStringInstance</code> (const NodeAddress nodeId, const NodeAddress interfaceAddress, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, char* parameterValue)
	This API is used to retrieve string parameter values from input files for a specific instance.
void	<code>IO_ReadBoolInstance</code> (const NodeAddress nodeId, const NodeAddress interfaceAddress, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, BOOL* parameterValue)
	This API is used to retrieve boolean parameter values from input files for a specific instance.
void	<code>IO_ReadIntInstance</code> (const NodeAddress nodeId, const NodeAddress interfaceAddress, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, int* parameterValue)
	This API is used to retrieve integer parameter values from input files for a specific instance.
void	<code>IO_ReadDoubleInstance</code> (const NodeAddress nodeId, const NodeAddress interfaceAddress, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, double* parameterValue)
	This API is used to retrieve double parameter values from input files for a specific instance.
void	<code>IO_ReadFloatInstance</code> (const NodeAddress nodeId, const NodeAddress interfaceAddress, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, float* parameterValue)

	This API is used to retrieve float parameter values from input files for a specific instance.
void	<u>IO_ReadTimeInstance</u> (const NodeAddress nodeId, const NodeAddress interfaceAddress, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, clocktype* parameterValue)
	This API is used to retrieve time parameter values from input files for a specific instance.
void	<u>IO_ReadCachedFileInstance</u> (const NodeAddress nodeId, const NodeAddress interfaceAddress, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, NodeInput* parameterValue)
	This API is used to retrieve file parameter values from input files for a specific instance.
void	<u>IO_ParseNodeIdHostOrNetworkAddress</u> (const char s[], NodeAddress* outputNodeAddress, int* numHostBits, BOOL* isNodeId)
	Parses a string for a nodeId, host address, or network address.
void	<u>IO_ParseNodeIdOrHostAddress</u> (const char s[], NodeAddress* outputNodeAddress, BOOL* isNodeId)
	Parses a string for a nodeId or host address.
void	<u>IO_ParseNetworkAddress</u> (const char s[], NodeAddress* outputNodeAddress, int* numHostBits)
	Parses a string for a network address.
void	<u>IO_FreeNodeInput</u> (NodeInput* nodeInput)
	Frees a NodeInput struct. (Currently unused.)
void	<u>IO_PrintStat</u> (Node* node, const char* layer, const char* protocol, NodeAddress interfaceAddress, int instanceId, const char* buf)
	Print out the relevant stat in "buf", along with the node id and the layer type generating this stat.
void	<u>IO_AppParseSourceAndDestStrings</u> (Node* node, const char* inputString, const char* sourceString, NodeAddress* sourceNodeId, NodeAddress* sourceAddr, const char* destString, NodeAddress* destNodeId, NodeAddress* destAddr)
	Application input parsing API. Parses the source and destination strings.
void	<u>IO_AppParseSourceString</u> (Node* node, const char* inputString, const char* sourceString, NodeAddress* sourceNodeId, NodeAddress* sourceAddr)
	Application input parsing API. Parses the source string.
void	<u>IO_AppParseDestString</u> (Node* node, const char* inputString, const char* destString, NodeAddress* destNodeId, NodeAddress* destAddr)

	<p>Application input parsing API. Parses the destination string.</p>
void	<pre>IO_AppParseHostString(Node* node, const char* inputString, const char* destString, NodeAddress* destNodeId, NodeAddress* destAddr)</pre>
	<p>Application input parsing API. Parses the host string.</p>
void	<pre>IO_AppForbidSameSourceAndDest(const char* inputString, NodeAddress sourceNodeId, NodeAddress destNodeId)</pre>
	<p>Application input checking API. Checks for the same source and destination node id. Calls abort() for same source and destination.</p>
BOOL	<pre>QualifierMatches(const NodeAddress nodeId, const NodeAddress interfaceAddress, const char* qualifier, int* matchType)</pre>
	<p>This is an auxiliary API used by the IO_Read...() set of APIs.</p>
void	<pre>IO_ReadBool(const NodeAddress nodeId, const in6_addr* interfaceAddress, const NodeInput* nodeInput, const char* parameterName, BOOL* wasFound, BOOL* parameterValue)</pre>
	<p>This API is used to retrieve boolean parameter values from input files. Overloaded API for Ipv6 compatibility.</p>
None	<pre>IO_ReadBool()</pre>
	<p>Reads boolean value for specified ATM address.</p>
void	<pre>IO_ReadBool(const NodeAddress nodeId, const Address* address, const NodeInput* nodeInput, const char* parameterName, BOOL* wasFound, BOOL* parameterValue)</pre>
	<p>This API is used to retrieve boolean parameter values from input files. Overloaded API for Ipv6 compatibility.</p>
void	<pre>IO_ReadString(const NodeId nodeId, const in6_addr* interfaceAddress, const NodeInput* nodeInput, const char* index, BOOL* wasFound, char* readVal)</pre>
	<p>This API is used to retrieve a string parameter value from input files. Overloaded API for Ipv6 compatibility.</p>
void	<pre>IO_ReadString(const NodeId nodeId, const AtmAddress* interfaceAddress, const NodeInput * nodeInput, const char * parameterName, BOOL * wasFound, char * parameterValue)</pre>
	<p>Reads string value for specified ATM address.</p>
void	<pre>IO_ReadString(const NodeId nodeId, const Address* address, const NodeInput* nodeInput, const char* index, BOOL* wasFound, char* readVal)</pre>
	<p>This API is used to retrieve a string parameter value from input files. Overloaded API for Ipv6 compatibility.</p>
void	<pre>IO_ReadInt(const NodeId nodeId, const in6_addr* interfaceAddress, const NodeInput* nodeInput, const char* index, BOOL* wasFound, int* readVal)</pre>

	This API is used to retrieve an integer parameter value from input files. Overloaded API for Ipv6 compatibility.
void	IO_ReadInt (const NodeId nodeId, const Address* address, const NodeInput* nodeInput, const char* index, BOOL* wasFound, int* readVal)
	This API is used to retrieve an integer parameter value from input files. Overloaded API for Ipv6 compatibility.
void	IO_ReadInt()
	Reads int value for specified ATM address.
void	IO_ReadDouble (const NodeId nodeId, const in6_addr* interfaceAddress, const NodeInput* nodeInput, const char* index, BOOL* wasFound, double* readVal)
	This API is used to retrieve a double parameter value from input files. Overloaded API for Ipv6 compatibility.
None	IO_ReadDouble()
	Reads double value for specified ATM address.
void	IO_ReadDouble (const NodeId nodeId, const Address* address, const NodeInput* nodeInput, const char* index, BOOL* wasFound, double* readVal)
	This API is used to retrieve a double parameter value from input files. Overloaded API for Ipv6 compatibility.
void	IO_ReadFloat (const NodeId nodeId, const in6_addr* interfaceAddress, const NodeInput* nodeInput, const char* index, BOOL* wasFound, float* readVal)
	This API is used to retrieve a float parameter value from input files. Overloaded API for Ipv6 compatibility.
void	IO_ReadFloat()
	Reads float value for specified ATM address.
void	IO_ReadFloat (const NodeId nodeId, const Address* address, const NodeInput* nodeInput, const char* index, BOOL* wasFound, float* readVal)
	This API is used to retrieve a float parameter value from input files. Overloaded API for Ipv6 compatibility.
void	IO_ReadTime (const NodeId nodeId, const in6_addr* interfaceAddress, const NodeInput* nodeInput, const char* index, BOOL* wasFound, clocktype* readVal)
	This API is used to retrieve time parameter value from input files. Overloaded API for Ipv6 compatibility.
void	IO_ReadTime()
	Reads time value for specified ATM address.

void	<code>IO_ReadTime</code> (const NodeId nodeId, const Address* address, const NodeInput* nodeInput, const char* index, BOOL* wasFound, clocktype* readVal)	This API is used to retrieve time parameter value from input files. Overloaded API for Ipv6 compatibility.
None	<code>IO_ReadBoolInstance</code> ()	Reads BOOL value for specified ATM address.
void	<code>IO_ReadStringInstance</code> (const NodeId nodeId, const in6_addr* interfaceAddress, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, char* parameterValue)	This API is used to retrieve string parameter values from input files for a specific instance. Overloaded API for Ipv6 compatibility.
void	<code>IO_ReadStringInstance</code> ()	Reads string value for specified ATM address.
void	<code>IO_ReadStringInstance</code> (const NodeId nodeId, const Address* address, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, char* parameterValue)	This API is used to retrieve string parameter values from input files for a specific instance. Overloaded API for Ipv6 compatibility.
void	<code>IO_ReadIntInstance</code> (const NodeId nodeId, const in6_addr* interfaceAddress, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, int* parameterValue)	This API is used to retrieve integer parameter values from input files for a specific instance. Overloaded API for Ipv6 compatibility.
void	<code>IO_ReadIntInstance</code> (const NodeId nodeId, const Address* address, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, int* parameterValue)	This API is used to retrieve integer parameter values from input files for a specific instance. Overloaded API for Ipv6 compatibility.
void	<code>IO_ReadDoubleInstance</code> (const NodeId nodeId, const in6_addr* interfaceAddress, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, double* parameterValue)	This API is used to retrieve double parameter values from input files for a specific instance. Overloaded API for Ipv6 compatibility.
void	<code>IO_ReadDoubleInstance</code> ()	Reads double value for specified ATM address.
void	<code>IO_ReadDoubleInstance</code> (const NodeId nodeId, const Address* address, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, double* parameterValue)	

	This API is used to retrieve double parameter values from input files for a specific instance. Overloaded API for Ipv6 compatibility.
void	<pre>IO_ReadFloatInstance(const NodeId nodeId, const in6_addr* interfaceAddress, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, float* parameterValue)</pre>
	This API is used to retrieve float parameter values from input files for a specific instance. Overloaded API for Ipv6 compatibility.
void	<pre>IO_ReadFloatInstance(const NodeId nodeId, const Address* address, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, float* parameterValue)</pre>
	This API is used to retrieve float parameter values from input files for a specific instance. Overloaded API for Ipv6 compatibility.
void	<pre>IO_ReadTimeInstance(const NodeId nodeId, const in6_addr* interfaceAddress, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, clocktype* parameterValue)</pre>
	This API is used to retrieve time parameter values from input files for a specific instance. Overloaded API for Ipv6 compatibility.
void	<pre>IO_ReadTimeInstance()</pre>
	Reads clocktype value for specified ATM address.
void	<pre>IO_ReadTimeInstance(const NodeId nodeId, const Address* address, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, clocktype* parameterValue)</pre>
	This API is used to retrieve time parameter values from input files for a specific instance. Overloaded API for Ipv6 compatibility.
void	<pre>IO_ReadCachedFile(const NodeAddress nodeId, const Address* address, const NodeInput* nodeInput, const char* index, BOOL* wasFound, NodeInput* parameterValue)</pre>
	This API is used to retrieve cached file parameter value from input files. Overloaded API for Ipv6 compatibility.
void	<pre>IO_ReadCachedFileInstance(const NodeAddress nodeId, const Address* address, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, NodeInput* parameterValue)</pre>
	This API is used to retrieve file parameter values from input files for a specific instance. Overloaded API for Ipv6 compatibility.
void	<pre>IO_PrintStat(Node* node, const char* layer, const char* protocol, const char* interfaceAddress, int instanceId, const char* buf)</pre>
	Print out the relevant stat in "buf", along with the node id and the layer type generating this stat. Overloaded API for Ipv6 compatibility.
void	<pre>IO_ParseNodeIdHostOrNetworkAddress(const char s[], in6_addr* ipAddress, BOOL* isIpAddr, NodeId* nodeId)</pre>

	Parses a string for a nodeId, host address, or network address. Overloaded API for Ipv6 compatibility.
void	IO_ParseNodeIdHostOrNetworkAddress (const char s[], ATM addr* atmAddress, BOOL* isAtmAddr, NodeId* nodeId)
	Parses a string for a nodeId, host address, or network address. Overloaded API for Ipv6 compatibility.
void	IO_ParseNodeIdOrHostAddress (const char s[], in6_addr* outputNodeAddress, BOOL* isNodeId)
	Parses a string for a nodeId or host address.
void	IO_ParseNetworkAddress (const char s[], unsigned int* tla, unsigned int* nla, unsigned int* sla)
	Parses a string for a network address. Overloaded API for Ipv6 compatibility.
void	IO_AppParseSourceAndDestStrings (Node* node, const char* inputString, const char* sourceString, NodeId* sourceNodeId, Address* sourceAddr, const char* destString, NodeId* destNodeId, Address* destAddr)
	Application input parsing API. Parses the source and destination strings. Overloaded for Ipv6 compatibility.
void	IO_AppParseSourceString (Node* node, const char* inputString, const char* sourceString, NodeAddress* sourceNodeId, Address* sourceAddr, NetworkType networkType)
	Application input parsing API. Parses the source string. Overloaded for Ipv6 compatibility.
void	IO_AppParseDestString (Node* node, const char* inputString, const char* destString, NodeAddress* destNodeId, Address* destAddr, NetworkType networkType)
	Application input parsing API. Parses the destination string. Overloaded for Ipv6 compatibility.
BOOL	QualifierMatches (const NodeId nodeId, const in6_addr interfaceAddress, const char* qualifier, int* matchType)
	This is an auxiliary API used by the IO_Read...() set of APIs. Overloaded for Ipv6 compatibility
BOOL	QualifierMatches (const NodeId nodeId, const AtmAddress* interfaceAddress, const char* qualifier, int* matchType)
	This is an auxiliary API used by the IO_Read...() set of APIs. Overloaded for Ipv6 compatibility
void	IO_ConvertIpv6StringToAddress() (char* interfaceAddr, in6_addr* ipAddress)
	Convert IPv6 address string to in6_addr structure. API for Ipv6 compatibility.
void	IO_ConvertIpAddressToString (in6_addr* ipAddress, char* interfaceAddr)
	Parses IPv6 address into a formatted string. Overloaded API for Ipv6 compatibility.
void	IO_ConvertIpAddressToString (Address* ipAddress, char* interfaceAddr)

	Parses IPv6 address into a formatted string. Overloaded API for Ipv6 compatibility. IO_ConvertStringToNodeAddress (char* addressString, NodeAddress* outputNodeAddress)
void	This API is used to covert a string parameter to NodeAdress. IO_CheckIsSameAddress (Address addr1, Address addr2)
BOOL	Compares IPv4 IPv6 address. API for Ipv6 compatibility.
void	IO_ReadString (Node* node node, const NodeInput* nodeInput, const char* index, BOOL* wasFound, char* readVal)
void	This API is used to retrieve a string parameter value from input files. IO_CacheFile (const NodeInput* nodeInput, const char* filename)
int	This API is used to read an auxiliary input file into a NodeInput struct Called for each file variable in the config file. IO_GetMaxLen (fileName char*)
int	This API is used to get the maximun length of a line in the file. IO_GetMaxLen (fp FILE*)
int	This API is used to get the maximun length of a line in the file. NI_GetMaxLen (nodeInput NodeInput*)
int	This API is used to get the maximun length of a line in nodeInput. NI_GetMaxLen (nodeInput const NodeInput*)
void	This API is used to get the maximun length of a line in nodeInput. IO_ParseNetworkAddress (const char s[], unsigned int* u_atmVal)
void	Parses a string for a network address. Overloaded API for ATM compatibility. IO_ConvertAddrToString (Address* address, char* addrStr)
void	Convert generic address to appropriate network type address string format. IO_ConvertAtmAddressToString (AtmAddress addr, char* addrStr)

	Convert Atm address to address string format.
void	<code>IO_InsertIntValue</code> (const char s[], const unsigned int val, unsigned int u_atmVal)
	Insert integer value for specific string in case of ATM
int	<code>IO_ReadCachedFileIndex</code> (NodeAddress nodeId, NodeAddress interfaceAddress, unsigned int nodeInput)
	Return Cached file index for the given parameter name
void	<code>IO_ReadString</code> (Node* node, const NodeId nodeId, int interfaceIndex, const NodeInput* nodeInput, const char* parameterName, BOOL* wasFound, char* parameterValue)
	This API is used to retrieve a string parameter value from input files.
void	<code>IO_ReadInt64</code> (Node* node, const NodeId nodeId, int interfaceIndex, const NodeInput* nodeInput, const char* parameterName, BOOL* wasFound, Int64* parameterValue)
	This API is used to retrieve a Int64 parameter value from input files.
void	<code>IO_ReadTime</code> (Node* node, const NodeId nodeId, int interfaceIndex, const NodeInput* nodeInput, const char* parameterName, BOOL* wasFound, clocktype* parameterValue)
	This API is used to retrieve a clocktype parameter value from input files.
void	<code>IO_ReadInt</code> (Node* node, const NodeId nodeId, int interfaceIndex, const NodeInput* nodeInput, const char* parameterName, BOOL* wasFound, int* parameterValue)
	This API is used to retrieve a Int parameter value from input files.
void	<code>IO_ReadDouble</code> (Node* node, const NodeId nodeId, int interfaceIndex, const NodeInput* nodeInput, const char* parameterName, BOOL* wasFound, double* parameterValue)
	This API is used to retrieve a double parameter value from input files.
void	<code>IO_ReadCachedFile</code> (Node* node, const NodeId nodeId, int interfaceIndex, const NodeInput* nodeInput, const char* parameterName, BOOL* wasFound, NodeInput* parameterValue)
	This API is used to retrieve a cached file parameter value from input files.
void	<code>IO_ReadLine</code> (Node* node, const NodeId nodeId, int interfaceIndex, const NodeInput* nodeInput, const char* parameterName, BOOL* wasFound, char* parameterValue)
	This API is used to retrieve a whole line from input files.
void	<code>IO_ReadStringInstance</code> (Node* node, const NodeId nodeId, int interfaceIndex, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, char* parameterValue)

	This API is used to retrieve string parameter values from input files for a specific instance.
void	<code>IO_ReadDoubleInstance</code> (Node* node, const NodeId nodeId, int interfaceIndex, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, double* parameterValue)
	This API is used to retrieve double parameter values from input files for a specific instance.
void	<code>IO_ReadIntInstance</code> (Node* node, const NodeId nodeId, int interfaceIndex, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, int* parameterValue)
	This API is used to retrieve int parameter values from input files for a specific instance.
void	<code>IO_ReadTimeInstance</code> (Node* node, const NodeId nodeId, int interfaceIndex, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, clocktype* parameterValue)
	This API is used to retrieve clocktype parameter values from input files for a specific instance.
void	<code>IO_ReadCachedFileInstance</code> (Node* node, const NodeId nodeId, int interfaceIndex, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, NodeInput* parameterValue)
	This API is used to retrieve cached file parameter values from input files for a specific instance.
void	<code>IO_ReadStringUsingIpAddress</code> (Node* node, int interfaceIndex, const NodeInput* nodeInput, const char* parameterName, BOOL* wasFound, char* parameterValue)
	This API is used to retrieve a string parameter value from input files using the ip-address.
void	<code>IO_ReadString</code> (const NodeAddress nodeId, NodeAddress ipv4Address, in6_addr* ipv6Address, const NodeInput* nodeInput, const char* parameterName, BOOL* wasFound, char* parameterValue)
	This API is used to retrieve a string parameter value from input files.
void	<code>IO_ReadString</code> (const NodeAddress nodeId, int interfaceIndex, const NodeAddress ipv4SubnetAddress, const in6_addr* ipv6SubnetAddress, const NodeInput* nodeInput, const char* parameterName, char* parameterValue, BOOL& wasFound, int& matchType)
	This API is used to retrieve a string parameter value from input files.

Constant / Data Structure Detail

Constant	ANY_NODEID 0xffffffff

	Optional macro values to use when calling IO_Read...() APIs. Defines any node id.
Constant	ANY_ADDRESS 0xffffffff Optional macro values to use when calling IO_Read...() APIs. Defines any node address.
Constant	ANY_INSTANCE 0xffffffff Optional macro values to use when calling IO_Read...() APIs. Defines any instance.
Constant	MAX_INPUT_FILE_LINE_LENGTH 6 * MAX_STRING_LENGTH Maximum input file line length. Evaluates (6 * MAX_STRING_LENGTH)
Constant	MAX_ADDRESS_STRING_LENGTH 80 Maximum length of address string.
Constant	MAX_NUM_CACHED_FILES 128 Max number of -FILE references in an input file. (Restriction is only for immediate children)
Constant	MATCH_NODE_ID 4 Defines the matching by node id.
Constant	MATCH_NETWORK 6 Defines the matching by network.
Constant	MATCH_INTERFACE 8 Defines the matching by interface.
Constant	INPUT_ALLOCATION_UNIT 500 Defines input allocation unit.
Structure	NodeInput Definition of node input structure. typedef to NodeInput in include/main.h.

Function / Macro Detail

Function / Macro	Format
IO_ConvertIpAddressToString Parses IPv4 address into a dotted-decimal string.	<p>void IO_ConvertIpAddressToString (NodeAddress ipAddress, char* addressString)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>ipAddress</code> - IPv4 address to be converted into • <code>addressString</code> - Storage for string. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
IO_FindStringPos Returns the index of the first subString found in s.	<p>int IO_FindStringPos (const char s[], const char subString[])</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>s[]</code> - Source string. • <code>subString[]</code> - Substring to search for. <p>Returns:</p> <ul style="list-style-type: none"> • <code>int</code> - Index of the first subString found in s. -1, if not found.
IO_GetToken Searches source buffer for the first %s-style token encountered, and copies it to dst.	<p>char* IO_GetToken (char* dst, const char* src, char ** next)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>dst</code> - Buffer to copy token too. If passed in as • <code>src</code> - Source string. • <code>next</code> - Storage for pointer to remainder of string. <p>Returns:</p> <ul style="list-style-type: none"> • <code>char*</code> - <code>dst</code>, if string was found and <code>dst</code> was passed in as non-NULL. Pointer to token in <code>src</code>, if string was found and <code>dst</code> was passed in as NULL. NULL, otherwise.
IO_GetDelimitedToken Searches source buffer for the first delimited token encountered, and copies it to dst.	<p>char* IO_GetDelimitedToken (char* dst, const char* src, const char* delim, char** next)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>dst</code> - Buffer to copy token too. If passed in as • <code>src</code> - Source string. • <code>delim</code> - Delimiter string.

	<ul style="list-style-type: none"> • <code>next</code> - Storage for pointer to remainder of string. <p>Returns:</p> <ul style="list-style-type: none"> • <code>char*</code> - <code>dst</code>, if string was found and <code>dst</code> was passed in as non-NULL. Pointer to token in <code>src</code>, if string was found and <code>dst</code> was passed in as NULL. <code>NULL</code>, otherwise.
IO_Right	<p><code>const char* IO_Right (const char * s, unsigned count)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>s</code> - String. • <code>count</code> - Number of characters on the right side. <p>Returns:</p> <ul style="list-style-type: none"> • <code>const char*</code> - A pointer to the right side of the string of length "count" characters. If <code>count</code> is 0, then a pointer to the string's terminating <code>NULL</code> is returned. If <code>count</code> is equal to or greater than the number of characters in the string, then the whole string is returned. A "character" is just a byte of <code>char</code> type that's not <code>NULL</code>. So, <code>\n</code> counts as a character.
IO_Chop	<p><code>char* IO_Chop (const char* s)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>s</code> - String. <p>Returns:</p> <ul style="list-style-type: none"> • <code>char*</code> - <code>s</code>. If the string has a <code>strlen()</code> of zero, then the string is returned unmodified. A "character" is just a byte of <code>char</code> type that's not <code>NULL</code>. So, <code>\n</code> counts as a character.
IO_TrimNsbpSpaces	<p><code>void IO_TrimNsbpSpaces (char* s)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>s</code> - String. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
IO_TrimLeft	<p><code>void IO_TrimLeft (char* s)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>s</code> - String. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
IO_TrimRight	<code>void IO_TrimRight (char* s)</code>

	<p>Strips trailing white space from a string (by inserting early NULL).</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>s</code> - String. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
IO_CompressWhiteSpace	<p>Compresses white space between words in the string to one space in a string. White space at the very beginning and very end of the string is also compressed to one space -- not stripped entirely.</p> <p>void <code>IO_CompressWhiteSpace</code> (char* s)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>s</code> - String. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
IO_IsStringNonNegativeInteger	<p>Returns TRUE if every character in string is a digit. (Even white space will cause return of FALSE)</p> <p>BOOL <code>IO_IsStringNonNegativeInteger</code> (const char* s)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>s</code> - String. <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - TRUE if every character is a digit. FALSE, otherwise.
IO_ConvertStringToLowerCase	<p>Runs tolower() on each character in string and converts the same to lowercase.</p> <p>void <code>IO_ConvertStringToLowerCase</code> (char s[])</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>s[]</code> - String. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
IO_ConvertStringToUpperCase	<p>Runs toupper() on each character in string and converts the same to uppercase.</p> <p>void <code>IO_ConvertStringToUpperCase</code> (char s[])</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>s[]</code> - String. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
IO_CaseInsensitiveStringsAreEqual	<p>Checks two strings are equal or not ignoring case.</p> <p>BOOL <code>IO_CaseInsensitiveStringsAreEqual</code> (const char[] s1, const char[] s2, char lengthToCompare)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>s1</code> - First string. • <code>s2</code> - Second string.

	<ul style="list-style-type: none"> • <code>lengthToCompare</code> - Length to compare. <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - Returns TRUE if strings are equal, FALSE otherwise.
IO_BankLine	<code>BOOL IO_BankLine (char s[])</code> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>s[]</code> - String. <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - Returns TRUE if the string is blank. FALSE, otherwise.
IO_CommentLine	<code>BOOL IO_CommentLine (char s[])</code> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>s[]</code> - String. <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - Returns TRUE if the line is a comment. FALSE, otherwise.
IO_FindCaseInsensitiveStringPos	<code>int IO_FindCaseInsensitiveStringPos (const char s[], const char subString[])</code> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>s[]</code> - String. • <code>subString[]</code> - Sub string <p>Returns:</p> <ul style="list-style-type: none"> • <code>int</code> - Returns the position of case insensitive sub string if found. -1, otherwise.
IO_FindCaseInsensitiveStringPos	<code>int IO_FindCaseInsensitiveStringPos (const char s[], const char subString[])</code> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>s[]</code> - String. • <code>subString[]</code> - Sub string <p>Returns:</p> <ul style="list-style-type: none"> • <code>int</code> - Returns the position of case insensitive sub string if found. -1, otherwise.
IO_SkipToken.	<code>IO_SkipToken. (char* token, char* tokenSep, char* skip)</code> <p>Parameters:</p>

<p>skip the first n tokens.</p>	<ul style="list-style-type: none"> • token - pointer to the input string, • tokenSep - pointer to the token separators, • skip - number of skips. <p>Returns:</p> <ul style="list-style-type: none"> • -
<p>IO_CreateNodeInput</p> <p>Allocates a NodeInput datastructure that can then be passed to IO_ReadNodeInput Called for each file variable in the config file.</p>	<p><code>NodeInput * IO_CreateNodeInput (NodeInput* nodeInput, const char* filename)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • nodeInput - Pointer to node input. • filename - Path to input file. <p>Returns:</p> <ul style="list-style-type: none"> • NodeInput * - None
<p>IO_InitializeNodeInput</p> <p>Initializes a NodeInput structure</p>	<p><code>void IO_InitializeNodeInput (NodeInput* nodeInput)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • nodeInput - A pointer to NodeInput structure. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
<p>IO_ReadNodeInput</p> <p>Reads an input file into a NodeInput struct. Calls IO_ReadFileParameters to first read in - FILE paramters. Then calls IO_ReadFileParameters to read the rest of the parameters.</p>	<p><code>void IO_ReadNodeInput (NodeInput* nodeInput, const char* filename)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • nodeInput - Pointer to node input. • filename - Path to input file. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
<p>IO_ReadNodeInputEx</p> <p>Reads an input file into a NodeInput struct. The includeComment Flag facilitate whether to include the commented line lines in the nodeInput structure or not. The commented lines should only be included in Backward Compatibility for Old scenario exe. This exe is</p>	<p><code>void IO_ReadNodeInputEx (NodeInput* nodeInput, const char* filename, const char* includeComment)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • nodeInput - Pointer to node input. • filename - Path to input file. • includeComment - When this flag is true it <p>Returns:</p>

<p>responsible for creating new config file and router model file. These new files contains changes according to current VERSION of QualNet. Calls IO_ReadFileParameters to first read in -FILE paramters. Then calls IO_ReadFileParameters to read the rest of the parameters.</p>	<ul style="list-style-type: none"> • void - None
<p>IO_ConvertFile</p> <p>Converts the contents of an old configuration file to the latest version.</p>	<p>BOOL IO_ConvertFile (NodeInput* nodeInput, NodeInput* nodeOutput, char* version)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • nodeInput - A pointer to node input. • nodeOutput - A pointer to node input. Goes through • version - Not used. <p>Returns:</p> <ul style="list-style-type: none"> • BOOL - Returns TRUE if able to convert. FALSE, otherwise. Either couldn't load the database or something else bad happened, so just copy the old into the new.
<p>IO_ReadLine</p> <p>This API is used to retrieve a whole line from input files.</p>	<p>void IO_ReadLine (const NodeAddress nodeId, const NodeAddress interfaceAddress, const NodeInput* nodeInput, const char* index, BOOL* wasFound, char* readVal)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • nodeId - nodeId. Can be ANY_NODEID. • interfaceAddress - IP address of interface. • nodeInput - Pointer to node input. • index - Parameter name. • wasFound - Storage for success of seach. • readVal - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
<p>IO_ReadString</p> <p>This API is used to retrieve a string parameter value from input files.</p>	<p>void IO_ReadString (const NodeAddress nodeId, const NodeAddress interfaceAddress, const NodeInput* nodeInput, const char* index, BOOL* wasFound, char* readVal)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • nodeId - nodeId. Can be ANY_NODEID. • interfaceAddress - IP address of interface. • nodeInput - Pointer to node input.

	<p>index - Parameter name.</p> <ul style="list-style-type: none"> • wasFound - Storage for success of seach. • readVal - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
IO_ReadBool	<p>This API is used to retrieve a boolean parameter value from input files.</p> <p>void IO_ReadBool (const NodeAddress nodeId, const NodeAddress interfaceAddress, const NodeInput* nodeInput, const char* index, BOOL* wasFound, BOOL* readVal)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • nodeId - nodeId. Can be ANY_NODEID. • interfaceAddress - IP address of interface. • nodeInput - Pointer to node input. • index - Parameter name. • wasFound - Storage for success of seach. • readVal - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
IO_ReadInt	<p>This API is used to retrieve an integer parameter value from input files.</p> <p>void IO_ReadInt (const NodeAddress nodeId, const NodeAddress interfaceAddress, const NodeInput* nodeInput, const char* index, BOOL* wasFound, int* readVal)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • nodeId - nodeId. Can be ANY_NODEID. • interfaceAddress - IP address of interface. • nodeInput - Pointer to node input. • index - Parameter name. • wasFound - Storage for success of seach. • readVal - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
IO_ReadDouble	<p>void IO_ReadDouble (const NodeAddress nodeId, const NodeAddress interfaceAddress, const NodeInput* nodeInput, const char* index, BOOL* wasFound, double* readVal)</p>

<p>This API is used to retrieve a double parameter value from input files.</p>	<p>Parameters:</p> <ul style="list-style-type: none"> • nodeId - nodeId. Can be ANY_NODEID. • interfaceAddress - IP address of interface. • nodeInput - Pointer to node input. • index - Parameter name. • wasFound - Storage for success of search. • readVal - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
<p>IO_ReadFloat</p> <p>This API is used to retrieve a float parameter value from input files.</p>	<p>void IO_ReadFloat (const NodeAddress nodeId, const NodeAddress interfaceAddress, const NodeInput* nodeInput, const char* index, BOOL* wasFound, float* readVal)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • nodeId - nodeId. Can be ANY_NODEID. • interfaceAddress - IP address of interface. • nodeInput - Pointer to node input. • index - Parameter name. • wasFound - Storage for success of search. • readVal - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
<p>IO_ReadTime</p> <p>This API is used to retrieve time parameter value from input files.</p>	<p>void IO_ReadTime (const NodeAddress nodeId, const NodeAddress interfaceAddress, const NodeInput* nodeInput, const char* index, BOOL* wasFound, clocktype* readVal)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • nodeId - nodeId. Can be ANY_NODEID. • interfaceAddress - IP address of interface. • nodeInput - Pointer to node input. • index - Parameter name. • wasFound - Storage for success of search. • readVal - Storage for parameter value.

	<p>Returns:</p> <ul style="list-style-type: none"> • void - None
IO_ReadCachedFile	<p>This API is used to retrieve cached file parameter value from input files.</p> <p>void IO_ReadCachedFile (const NodeAddress nodeId, const NodeAddress interfaceAddress, const NodeInput* nodeInput, const char* index, BOOL* wasFound, NodeInput* parameterValue)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • nodeId - nodeId. Can be ANY_NODEID. • interfaceAddress - IP address of interface. • nodeInput - Pointer to node input. • index - Parameter name. • wasFound - Storage for success of search. • parameterValue - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
IO_ReadStringInstance	<p>This API is used to retrieve string parameter values from input files for a specific instance.</p> <p>void IO_ReadStringInstance (const NodeAddress nodeId, const NodeAddress interfaceAddress, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, char* parameterValue)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • nodeId - nodeId. Can be ANY_NODEID. • interfaceAddress - IP address of interface. • nodeInput - Pointer to node input. • parameterName - Parameter name. • parameterInstanceNumber - Instance number. • fallbackIfNoInstanceMatch - Selects parameter without instance number. • wasFound - Storage for success of search. • parameterValue - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
IO_ReadBoolInstance	<p>void IO_ReadBoolInstance (const NodeAddress nodeId, const NodeAddress interfaceAddress, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, BOOL* parameterValue)</p>

<p>This API is used to retrieve boolean parameter values from input files for a specific instance.</p>	<p>Parameters:</p> <ul style="list-style-type: none"> • nodeId - nodeId. Can be ANY_NODEID. • interfaceAddress - IP address of interface. • nodeInput - Pointer to node input. • parameterName - Parameter name. • parameterInstanceNumber - Instance number. • fallbackIfNoInstanceMatch - Selects parameter without • wasFound - Storage for success of seach. • parameterValue - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
<p>IO_ReadIntInstance</p> <p>This API is used to retrieve integer parameter values from input files for a specific instance.</p>	<p>void IO_ReadIntInstance (const NodeAddress nodeId, const NodeAddress interfaceAddress, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, int* parameterValue)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • nodeId - nodeId. Can be ANY_NODEID. • interfaceAddress - IP address of interface. • nodeInput - Pointer to node input. • parameterName - Parameter name. • parameterInstanceNumber - Instance number. • fallbackIfNoInstanceMatch - Selects parameter without • wasFound - Storage for success of seach. • parameterValue - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
<p>IO_ReadDoubleInstance</p> <p>This API is used to retrieve double parameter values from input files for a specific instance.</p>	<p>void IO_ReadDoubleInstance (const NodeAddress nodeId, const NodeAddress interfaceAddress, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, double* parameterValue)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • nodeId - nodeId. Can be ANY_NODEID.

	<ul style="list-style-type: none"> • <code>interfaceAddress</code> - IP address of interface. • <code>nodeInput</code> - Pointer to node input. • <code>parameterName</code> - Parameter name. • <code>parameterInstanceNumber</code> - Instance number. • <code>fallbackIfNoInstanceMatch</code> - Selects parameter without • <code>wasFound</code> - Storage for success of search. • <code>parameterValue</code> - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
IO_ReadFloatInstance This API is used to retrieve float parameter values from input files for a specific instance.	<p><code>void IO_ReadFloatInstance (const NodeAddress nodeId, const NodeAddress interfaceAddress, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, float* parameterValue)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>nodeId</code> - nodeId. Can be ANY_NODEID. • <code>interfaceAddress</code> - IP address of interface. • <code>nodeInput</code> - Pointer to node input. • <code>parameterName</code> - Parameter name. • <code>parameterInstanceNumber</code> - Instance number. • <code>fallbackIfNoInstanceMatch</code> - Selects parameter without • <code>wasFound</code> - Storage for success of search. • <code>parameterValue</code> - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
IO_ReadTimeInstance This API is used to retrieve time parameter values from input files for a specific instance.	<p><code>void IO_ReadTimeInstance (const NodeAddress nodeId, const NodeAddress interfaceAddress, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, clocktype* parameterValue)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>nodeId</code> - nodeId. Can be ANY_NODEID. • <code>interfaceAddress</code> - IP address of interface.

	<p><code>nodeInput - Pointer to node input.</code></p> <ul style="list-style-type: none"> • <code>parameterName - Parameter name.</code> • <code>parameterInstanceNumber - Instance number.</code> • <code>fallbackIfNoInstanceMatch - Selects parameter without</code> • <code>wasFound - Storage for success of seach.</code> • <code>parameterValue - Storage for parameter value.</code> <p><code>Returns:</code></p> <ul style="list-style-type: none"> • <code>void - None</code>
IO_ReadCachedFileInstance	<p>This API is used to retrieve file parameter values from input files for a specific instance.</p> <p><code>void IO_ReadCachedFileInstance (const NodeAddress nodeId, const NodeAddress interfaceAddress, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, NodeInput* parameterValue)</code></p> <p><code>Parameters:</code></p> <ul style="list-style-type: none"> • <code>nodeId - nodeId. Can be ANY_NODEID.</code> • <code>interfaceAddress - IP address of interface.</code> • <code>nodeInput - Pointer to node input.</code> • <code>parameterName - Parameter name.</code> • <code>parameterInstanceNumber - Instance number.</code> • <code>fallbackIfNoInstanceMatch - Selects parameter without</code> • <code>wasFound - Storage for success of seach.</code> • <code>parameterValue - Storage for parameter value.</code> <p><code>Returns:</code></p> <ul style="list-style-type: none"> • <code>void - None</code>
IO_ParseNodeIdHostOrNetworkAddress	<p>Parses a string for a nodeId, host address, or network address.</p> <p><code>void IO_ParseNodeIdHostOrNetworkAddress (const char s[], NodeAddress* outputNodeAddress, int* numHostBits, BOOL* isNodeId)</code></p> <p><code>Parameters:</code></p> <ul style="list-style-type: none"> • <code>s[] - String to parse.</code> • <code>outputNodeAddress - Storage for nodeId or IP address.</code> • <code>numHostBits - Storage for number of host bits</code> • <code>isNodeId - Storage for whether the string is</code>

	<p>Returns:</p> <ul style="list-style-type: none"> • void - None
IO_ParseNodeIdOrHostAddress	<p>Parses a string for a nodeId or host address.</p> <p>void IO_ParseNodeIdOrHostAddress (const char s[], NodeAddress* outputNodeAddress, BOOL* isNodeId)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • s[] - String to parse. • outputNodeAddress - Storage for nodeId or IP address. • isNodeId - Storage for whether the string is <p>Returns:</p> <ul style="list-style-type: none"> • void - None
IO_ParseNetworkAddress	<p>Parses a string for a network address.</p> <p>void IO_ParseNetworkAddress (const char s[], NodeAddress* outputNodeAddress, int* numHostBits)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • s[] - String to parse. • outputNodeAddress - Storage for network address. • numHostBits - Storage for number of host bits <p>Returns:</p> <ul style="list-style-type: none"> • void - None
IO_FreeNodeInput	<p>Frees a NodeInput struct. (Currently unused.)</p> <p>void IO_FreeNodeInput (NodeInput* nodeInput)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • nodeInput - Pointer to node input. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
IO_PrintStat	<p>Print out the relevant stat in "buf", along with the node id and the layer type generating this stat.</p> <p>void IO_PrintStat (Node* node, const char* layer, const char* protocol, NodeAddress interfaceAddress, int instanceId, const char* buf)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - The node generating the stat. • layer - The layer generating the stat. • protocol - The protocol generating the stat. • interfaceAddress - Interface address.

	<ul style="list-style-type: none"> • <code>instanceId</code> - Instance id. • <code>buf</code> - String which has the statistic to <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
IO_AppParseSourceAndDestStrings	<p>Application input parsing API. Parses the source and destination strings.</p> <pre>void IO_AppParseSourceAndDestStrings (Node* node, const char* inputString, const char* sourceString, NodeAddress* sourceNodeId, NodeAddress* sourceAddr, const char* destString, NodeAddress* destNodeId, NodeAddress* destAddr)</pre> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - A pointer to Node. • <code>inputString</code> - The input string. • <code>sourceString</code> - The source string. • <code>sourceNodeId</code> - A pointer to NodeAddress. • <code>sourceAddr</code> - A pointer to NodeAddress. • <code>destString</code> - Const char pointer. • <code>destNodeId</code> - A pointer to NodeAddress. • <code>destAddr</code> - A pointer to NodeAddress. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
IO_AppParseSourceString	<p>Application input parsing API. Parses the source string.</p> <pre>void IO_AppParseSourceString (Node* node, const char* inputString, const char* sourceString, NodeAddress* sourceNodeId, NodeAddress* sourceAddr)</pre> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - A pointer to Node. • <code>inputString</code> - The input string. • <code>sourceString</code> - The source string. • <code>sourceNodeId</code> - A pointer to NodeAddress. • <code>sourceAddr</code> - A pointer to NodeAddress. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
IO_AppParseDestString	<pre>void IO_AppParseDestString (Node* node, const char* inputString, const char* destString, NodeAddress* destNodeId, NodeAddress* destAddr)</pre>

<p>Application input parsing API. Parses the destination string.</p>	<p>Parameters:</p> <ul style="list-style-type: none"> • node - A pointer to Node. • inputString - The input string. • destString - Const char pointer. • destNodeId - A pointer to NodeAddress. • destAddr - A pointer to NodeAddress. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
<p>IO_AppParseHostString</p> <p>Application input parsing API. Parses the host string.</p>	<p>void IO_AppParseHostString (Node* node, const char* inputString, const char* destString, NodeAddress* destNodeId, NodeAddress* destAddr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - A pointer to Node. • inputString - The input string. • destString - Const char pointer. • destNodeId - A pointer to NodeAddress. • destAddr - A pointer to NodeAddress. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
<p>IO_AppForbidSameSourceAndDest</p> <p>Application input checking API. Checks for the same source and destination node id. Calls abort() for same source and destination.</p>	<p>void IO_AppForbidSameSourceAndDest (const char* inputString, NodeAddress sourceNodeId, NodeAddress destNodeId)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • inputString - The input string. • sourceNodeId - Source node id, read from the • destNodeId - Destination node id, read from the <p>Returns:</p> <ul style="list-style-type: none"> • void - None
<p>QualifierMatches</p> <p>This is an auxiliary API used by the</p>	<p>BOOL QualifierMatches (const NodeAddress nodeId, const NodeAddress interfaceAddress, const char* qualifier, int* matchType)</p> <p>Parameters:</p>

<p>IO_Read...() set of APIs.</p>	<ul style="list-style-type: none"> • <code>nodeId</code> - nodeId to select for. • <code>interfaceAddress</code> - IP address to select for. • <code>qualifier</code> - String containing the • <code>matchType</code> - Stores the type of the match, <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - Returns TRUE if match found. FALSE, otherwise.
<p>IO_ReadBool</p> <p>This API is used to retrieve boolean parameter values from input files. Overloaded API for Ipv6 compatibility.</p>	<p>void IO_ReadBool (const NodeAddress nodeId, const in6_addr* interfaceAddress, const NodeInput* nodeInput, const char* parameterName, BOOL* wasFound, BOOL* parameterValue)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>nodeId</code> - nodeId. Can be ANY_NODEID. • <code>interfaceAddress</code> - IPv6 address of interface. • <code>nodeInput</code> - Pointer to node input. • <code>parameterName</code> - Parameter name. • <code>wasFound</code> - Storage for success of seach. • <code>parameterValue</code> - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
<p>IO_ReadBool</p> <p>Reads boolean value for specified ATM address.</p>	<p>None IO_ReadBool ()</p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> • <code>None</code> - None
<p>IO_ReadBool</p> <p>This API is used to retrieve boolean parameter values from input files. Overloaded API for Ipv6 compatibility.</p>	<p>void IO_ReadBool (const NodeAddress nodeId, const Address* address, const NodeInput* nodeInput, const char* parameterName, BOOL* wasFound, BOOL* parameterValue)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>nodeId</code> - nodeId. Can be ANY_NODEID. • <code>address</code> - Address of interface. • <code>nodeInput</code> - Pointer to node input. • <code>parameterName</code> - Parameter name. • <code>wasFound</code> - Storage for success of seach.

	<ul style="list-style-type: none"> • <code>parameterValue</code> - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
IO_ReadString	<p>This API is used to retrieve a string parameter value from input files. Overloaded API for Ipv6 compatibility.</p> <p>void IO_ReadString (const NodeId nodeId, const in6_addr* interfaceAddress, const NodeInput* nodeInput, const char* index, BOOL* wasFound, char* readVal)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>nodeId</code> - nodeId. Can be ANY_NODEID. • <code>interfaceAddress</code> - IP address of interface. • <code>nodeInput</code> - Pointer to node input. • <code>index</code> - Parameter name. • <code>wasFound</code> - Storage for success of seach. • <code>readVal</code> - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
IO_ReadString	<p>Reads string value for specified ATM address.</p> <p>void IO_ReadString (const NodeId nodeId, const AtmAddress* interfaceAddress, const NodeInput * nodeInput, const char * parameterName, BOOL * wasFound, char * parameterValue)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>nodeId</code> - NodeId for which parameter has • <code>interfaceAddress</code> - ATM Interface address • <code>nodeInput</code> - pointer to configuration inputs • <code>parameterName</code> - Parameter to be read • <code>wasFound</code> - Parameter found or not • <code>parameterValue</code> - Parameter's value if found. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
IO_ReadString	<p>This API is used to retrieve a string parameter value from input files. Overloaded API for</p> <p>void IO_ReadString (const NodeId nodeId, const Address* address, const NodeInput* nodeInput, const char* index, BOOL* wasFound, char* readVal)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>nodeId</code> - nodeId. Can be ANY_NODEID.

<p>Ipv6 compatibility.</p>	<ul style="list-style-type: none"> • <code>address</code> - IP address of interface. • <code>nodeInput</code> - Pointer to node input. • <code>index</code> - Parameter name. • <code>wasFound</code> - Storage for success of seach. • <code>readVal</code> - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
<p>IO_ReadInt</p> <p>This API is used to retrieve an integer parameter value from input files. Overloaded API for Ipv6 compatibility.</p>	<p>void IO_ReadInt (const NodeId nodeId, const in6_addr* interfaceAddress, const NodeInput* nodeInput, const char* index, BOOL* wasFound, int* readVal)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>nodeId</code> - nodeId. Can be ANY_NODEID. • <code>interfaceAddress</code> - IPv6 address of interface. • <code>nodeInput</code> - Pointer to node input. • <code>index</code> - Parameter name. • <code>wasFound</code> - Storage for success of seach. • <code>readVal</code> - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
<p>IO_ReadInt</p> <p>This API is used to retrieve an integer parameter value from input files. Overloaded API for Ipv6 compatibility.</p>	<p>void IO_ReadInt (const NodeId nodeId, const Address* address, const NodeInput* nodeInput, const char* index, BOOL* wasFound, int* readVal)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>nodeId</code> - nodeId. Can be ANY_NODEID. • <code>address</code> - Address of interface. • <code>nodeInput</code> - Pointer to node input. • <code>index</code> - Parameter name. • <code>wasFound</code> - Storage for success of seach. • <code>readVal</code> - Storage for parameter value. <p>Returns:</p>

	<ul style="list-style-type: none"> • void - None
IO_ReadInt	<p>void IO_ReadInt ()</p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> • void - None NOTE: Overloaded API IO_ReadInt()
IO_ReadDouble	<p>This API is used to retrieve a double parameter value from input files. Overloaded API for Ipv6 compatibility.</p> <p>void IO_ReadDouble (const NodeId nodeId, const in6_addr* interfaceAddress, const NodeInput* nodeInput, const char* index, BOOL* wasFound, double* readVal)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • nodeId - nodeId. Can be ANY_NODEID. • interfaceAddress - IPv6 address of interface. • nodeInput - Pointer to node input. • index - Parameter name. • wasFound - Storage for success of seach. • readVal - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
IO_ReadDouble	<p>Reads double value for specified ATM address.</p> <p>None IO_ReadDouble ()</p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> • None - None
IO_ReadDouble	<p>This API is used to retrieve a double parameter value from input files. Overloaded API for Ipv6 compatibility.</p> <p>void IO_ReadDouble (const NodeId nodeId, const Address* address, const NodeInput* nodeInput, const char* index, BOOL* wasFound, double* readVal)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • nodeId - nodeId. Can be ANY_NODEID. • address - Address of interface. • nodeInput - Pointer to node input. • index - Parameter name. • wasFound - Storage for success of seach. • readVal - Storage for parameter value.

	<p>Returns:</p> <ul style="list-style-type: none"> • void - None
IO_ReadFloat	<p>This API is used to retrieve a float parameter value from input files. Overloaded API for Ipv6 compatibility.</p> <p>void IO_ReadFloat (const NodeId nodeId, const in6_addr* interfaceAddress, const NodeInput* nodeInput, const char* index, BOOL* wasFound, float* readVal)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • nodeId - nodeId. Can be ANY_NODEID. • interfaceAddress - IPv6 address of interface. • nodeInput - Pointer to node input. • index - Parameter name. • wasFound - Storage for success of seach. • readVal - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
IO_ReadFloat	<p>Reads float value for specified ATM address.</p> <p>void IO_ReadFloat ()</p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> • void - None NOTE: Overloaded API IO_ReadFloat()
IO_ReadFloat	<p>This API is used to retrieve a float parameter value from input files. Overloaded API for Ipv6 compatibility.</p> <p>void IO_ReadFloat (const NodeId nodeId, const Address* address, const NodeInput* nodeInput, const char* index, BOOL* wasFound, float* readVal)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • nodeId - nodeId. Can be ANY_NODEID. • address - Address of interface. • nodeInput - Pointer to node input. • index - Parameter name. • wasFound - Storage for success of seach. • readVal - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none"> • void - None

IO_ReadTime	<p>This API is used to retrieve time parameter value from input files. Overloaded API for Ipv6 compatibility.</p>	<p>void IO_ReadTime (const NodeId nodeId, const in6_addr* interfaceAddress, const NodeInput* nodeInput, const char* index, BOOL* wasFound, clocktype* readVal)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • nodeId - nodeId. Can be ANY_NODEID. • interfaceAddress - IPv6 address of interface. • nodeInput - Pointer to node input. • index - Parameter name. • wasFound - Storage for success of seach. • readVal - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
IO_ReadTime	Reads time value for specified ATM address.	<p>void IO_ReadTime ()</p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> • void - None
IO_ReadTime	<p>This API is used to retrieve time parameter value from input files. Overloaded API for Ipv6 compatibility.</p>	<p>void IO_ReadTime (const NodeId nodeId, const Address* address, const NodeInput* nodeInput, const char* index, BOOL* wasFound, clocktype* readVal)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • nodeId - nodeId. Can be ANY_NODEID. • address - Address of interface. • nodeInput - Pointer to node input. • index - Parameter name. • wasFound - Storage for success of seach. • readVal - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
IO_ReadBoolInstance	Reads BOOL value for specified ATM	<p>None IO_ReadBoolInstance ()</p> <p>Parameters:</p> <p>Returns:</p>

address.	<ul style="list-style-type: none"> • None - None
IO_ReadStringInstance This API is used to retrieve string parameter values from input files for a specific instance. Overloaded API for Ipv6 compatibility.	<p>void IO_ReadStringInstance (const NodeId nodeId, const in6_addr* interfaceAddress, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, char* parameterValue)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • nodeId - nodeId. Can be ANY_NODEID. • interfaceAddress - IPv6 address of interface. • nodeInput - Pointer to node input. • parameterName - Parameter name. • parameterInstanceNumber - Instance number. • fallbackIfNoInstanceMatch - Selects parameter without • wasFound - Storage for success of seach. • parameterValue - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
IO_ReadStringInstance Reads string value for specified ATM address.	<p>void IO_ReadStringInstance ()</p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> • void - None NOTE: Overloaded API IO_ReadStringInstance()
IO_ReadStringInstance This API is used to retrieve string parameter values from input files for a specific instance. Overloaded API for Ipv6 compatibility.	<p>void IO_ReadStringInstance (const NodeId nodeId, const Address* address, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, char* parameterValue)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • nodeId - nodeId. Can be ANY_NODEID. • address - Address of interface. • nodeInput - Pointer to node input. • parameterName - Parameter name. • parameterInstanceNumber - Instance number. • fallbackIfNoInstanceMatch - Selects parameter without

	<ul style="list-style-type: none"> • <code>wasFound</code> - Storage for success of seach. • <code>parameterValue</code> - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
IO_ReadIntInstance	<p>This API is used to retrieve integer parameter values from input files for a specific instance. Overloaded API for Ipv6 compatibility.</p> <pre>void IO_ReadIntInstance (const NodeId nodeId, const in6_addr* interfaceAddress, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, int* parameterValue)</pre> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>nodeId</code> - nodeId. Can be ANY_NODEID. • <code>interfaceAddress</code> - IPv6 address of interface. • <code>nodeInput</code> - Pointer to node input. • <code>parameterName</code> - Parameter name. • <code>parameterInstanceNumber</code> - Instance number. • <code>fallbackIfNoInstanceMatch</code> - Selects parameter without • <code>wasFound</code> - Storage for success of seach. • <code>parameterValue</code> - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
IO_ReadIntInstance	<p>This API is used to retrieve integer parameter values from input files for a specific instance. Overloaded API for Ipv6 compatibility.</p> <pre>void IO_ReadIntInstance (const NodeId nodeId, const Address* address, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, int* parameterValue)</pre> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>nodeId</code> - nodeId. Can be ANY_NODEID. • <code>address</code> - Address of interface. • <code>nodeInput</code> - Pointer to node input. • <code>parameterName</code> - Parameter name. • <code>parameterInstanceNumber</code> - Instance number. • <code>fallbackIfNoInstanceMatch</code> - Selects parameter without • <code>wasFound</code> - Storage for success of seach. • <code>parameterValue</code> - Storage for parameter value.

	<p>Returns:</p> <ul style="list-style-type: none"> • void - None
IO_ReadDoubleInstance	<p>This API is used to retrieve double parameter values from input files for a specific instance. Overloaded API for Ipv6 compatibility.</p> <p>void IO_ReadDoubleInstance (const NodeId nodeId, const in6_addr* interfaceAddress, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, double* parameterValue)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • nodeId - nodeId. Can be ANY_NODEID. • interfaceAddress - IPv6 address of interface. • nodeInput - Pointer to node input. • parameterName - Parameter name. • parameterInstanceNumber - Instance number. • fallbackIfNoInstanceMatch - Selects parameter without • wasFound - Storage for success of seach. • parameterValue - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
IO_ReadDoubleInstance	<p>Reads double value for specified ATM address.</p> <p>void IO_ReadDoubleInstance ()</p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> • void - None NOTE: Overloaded API IO_ReadDoubleInstance()
IO_ReadDoubleInstance	<p>This API is used to retrieve double parameter values from input files for a specific instance. Overloaded API for Ipv6 compatibility.</p> <p>void IO_ReadDoubleInstance (const NodeId nodeId, const Address* address, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, double* parameterValue)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • nodeId - nodeId. Can be ANY_NODEID. • address - IPv6 address of interface. • nodeInput - Pointer to node input. • parameterName - Parameter name. • parameterInstanceNumber - Instance number.

	<p><code>fallbackIfNoInstanceMatch</code> - Selects parameter without</p> <ul style="list-style-type: none"> • <code>wasFound</code> - Storage for success of search. • <code>parameterValue</code> - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
IO_ReadFloatInstance	<p>This API is used to retrieve float parameter values from input files for a specific instance. Overloaded API for Ipv6 compatibility.</p> <p><code>void IO_ReadFloatInstance (const NodeId nodeId, const in6_addr* interfaceAddress, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, float* parameterValue)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>nodeId</code> - nodeId. Can be ANY_NODEID. • <code>interfaceAddress</code> - IPv6 address of interface. • <code>nodeInput</code> - Pointer to node input. • <code>parameterName</code> - Parameter name. • <code>parameterInstanceNumber</code> - Instance number. • <code>fallbackIfNoInstanceMatch</code> - Selects parameter without • <code>wasFound</code> - Storage for success of search. • <code>parameterValue</code> - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
IO_ReadFloatInstance	<p>This API is used to retrieve float parameter values from input files for a specific instance. Overloaded API for Ipv6 compatibility.</p> <p><code>void IO_ReadFloatInstance (const NodeId nodeId, const Address* address, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, float* parameterValue)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>nodeId</code> - nodeId. Can be ANY_NODEID. • <code>address</code> - Address of interface. • <code>nodeInput</code> - Pointer to node input. • <code>parameterName</code> - Parameter name. • <code>parameterInstanceNumber</code> - Instance number. • <code>fallbackIfNoInstanceMatch</code> - Selects parameter without • <code>wasFound</code> - Storage for success of search.

	<ul style="list-style-type: none"> • <code>parameterValue</code> - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
IO_ReadTimeInstance	<p>This API is used to retrieve time parameter values from input files for a specific instance. Overloaded API for Ipv6 compatibility.</p> <pre>void IO_ReadTimeInstance (const NodeId nodeId, const in6_addr* interfaceAddress, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, clocktype* parameterValue)</pre> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>nodeId</code> - nodeId. Can be ANY_NODEID. • <code>interfaceAddress</code> - IPv6 address of interface. • <code>nodeInput</code> - Pointer to node input. • <code>parameterName</code> - Parameter name. • <code>parameterInstanceNumber</code> - Instance number. • <code>fallbackIfNoInstanceMatch</code> - Selects parameter without • <code>wasFound</code> - Storage for success of seach. • <code>parameterValue</code> - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
IO_ReadTimeInstance	<p>Reads clocktype value for specified ATM address.</p> <pre>void IO_ReadTimeInstance ()</pre> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None NOTE: Overloaded API IO_ReadTimeInstance()
IO_ReadTimeInstance	<p>This API is used to retrieve time parameter values from input files for a specific instance. Overloaded API for Ipv6 compatibility.</p> <pre>void IO_ReadTimeInstance (const NodeId nodeId, const Address* address, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, clocktype* parameterValue)</pre> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>nodeId</code> - nodeId. Can be ANY_NODEID. • <code>address</code> - Address of interface. • <code>nodeInput</code> - Pointer to node input. • <code>parameterName</code> - Parameter name.

	<p>parameterInstanceNumber - Instance number.</p> <ul style="list-style-type: none"> • fallbackIfNoInstanceMatch - Selects parameter without • wasFound - Storage for success of search. • parameterValue - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
IO_ReadCachedFile	<p>This API is used to retrieve cached file parameter value from input files. Overloaded API for Ipv6 compatibility.</p> <p>void IO_ReadCachedFile (const NodeAddress nodeId, const Address* address, const NodeInput* nodeInput, const char* index, BOOL* wasFound, NodeInput* parameterValue)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • nodeId - nodeId. Can be ANY_NODEID. • address - Address of interface. • nodeInput - Pointer to node input. • index - Parameter name. • wasFound - Storage for success of search. • parameterValue - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
IO_ReadCachedFileInstance	<p>This API is used to retrieve file parameter values from input files for a specific instance. Overloaded API for Ipv6 compatibility.</p> <p>void IO_ReadCachedFileInstance (const NodeAddress nodeId, const Address* address, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, NodeInput* parameterValue)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • nodeId - nodeId. Can be ANY_NODEID. • address - Address of interface. • nodeInput - Pointer to node input. • parameterName - Parameter name. • parameterInstanceNumber - Instance number. • fallbackIfNoInstanceMatch - Selects parameter without • wasFound - Storage for success of search. • parameterValue - Storage for parameter value.

	<p>Returns:</p> <ul style="list-style-type: none"> • void - None
IO_PrintStat	<p>Print out the relevant stat in "buf", along with the node id and the layer type generating this stat. Overloaded API for Ipv6 compatibility.</p> <p>void IO_PrintStat (Node* node, const char* layer, const char* protocol, const char* interfaceAddress, int instanceId, const char* buf)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - The node generating the stat. • layer - The layer generating the stat. • protocol - The protocol generating the stat. • interfaceAddress - The Interface address the stat. • instanceId - Instance id. • buf - String which has the statistic to <p>Returns:</p> <ul style="list-style-type: none"> • void - None
IO_ParseNodeIdHostOrNetworkAddress	<p>Parses a string for a nodeId, host address, or network address. Overloaded API for Ipv6 compatibility.</p> <p>void IO_ParseNodeIdHostOrNetworkAddress (const char s[], in6_addr* ipAddress, BOOL* isIpAddr, NodeId* nodeId)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • s[] - String to parse. • ipAddress - Storage for ipv6address. • isIpAddr - Storage for whether the string is • nodeId - Storage for nodeId. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
IO_ParseNodeIdHostOrNetworkAddress	<p>Parses a string for a nodeId, host address, or network address. Overloaded API for Ipv6 compatibility.</p> <p>void IO_ParseNodeIdHostOrNetworkAddress (const char s[], ATM addr* atmAddress, BOOL* isAtmAddr, NodeId* nodeId)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • s[] - String to parse. • atmAddress - Storage for ATMAddress. • isAtmAddr - Storage for whether the string is • nodeId - Storage for nodeId. <p>Returns:</p>

	<ul style="list-style-type: none"> • void - None
IO_ParseNodeIdOrHostAddress	<p>Parses a string for a nodeId or host address.</p> <p>void <code>IO_ParseNodeIdOrHostAddress</code> (const char s[], in6_addr* outputNodeAddress, BOOL* isNodeId)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • s[] - String to parse. • outputNodeAddress - Storage for ipv6address. • isNodeId - Storage for whether the string is <p>Returns:</p> <ul style="list-style-type: none"> • void - None
IO_ParseNetworkAddress	<p>Parses a string for a network address. Overloaded API for Ipv6 compatibility.</p> <p>void <code>IO_ParseNetworkAddress</code> (const char s[], unsigned int* tla, unsigned int* nla, unsigned int* sla)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • s[] - String to parse. • tla - Storage for tla • nla - Storage for nla. • sla - Storage for sla. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
IO_AppParseSourceAndDestStrings	<p>Application input parsing API. Parses the source and destination strings. Overloaded for Ipv6 compatibility.</p> <p>void <code>IO_AppParseSourceAndDestStrings</code> (Node* node, const char* inputString, const char* sourceString, NodeId* sourceNodeId, Address* sourceAddr, const char* destString, NodeId* destNodeId, Address* destAddr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - A pointer to Node. • inputString - The input string. • sourceString - The source string. • sourceNodeId - A pointer to NodeId. • sourceAddr - A pointer to Address. • destString - Const char pointer. • destNodeId - A pointer to NodeId. • destAddr - A pointer to Address. <p>Returns:</p>

	<ul style="list-style-type: none"> • void - None
IO_AppParseSourceString	<p>void IO_AppParseSourceString (Node* node, const char* inputString, const char* sourceString, NodeAddress* sourceNodeId, Address* sourceAddr, NetworkType networkType)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - A pointer to Node. • inputString - The input string. • sourceString - The source string. • sourceNodeId - A pointer to NodeAddress. • sourceAddr - A pointer to Address. • networkType - used when sourceString <p>Returns:</p> <ul style="list-style-type: none"> • void - None
IO_AppParseDestString	<p>void IO_AppParseDestString (Node* node, const char* inputString, const char* destString, NodeAddress* destNodeId, Address* destAddr, NetworkType networkType)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - A pointer to Node. • inputString - The input string. • destString - Const char pointer. • destNodeId - A pointer to NodeAddress. • destAddr - A pointer to Address. • networkType - used when sourceString <p>Returns:</p> <ul style="list-style-type: none"> • void - None
QualifierMatches	<p>BOOL QualifierMatches (const NodeId nodeId, const in6_addr interfaceAddress, const char* qualifier, int* matchType)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • nodeId - nodeId to select for. • interfaceAddress - IPv6 address to select for. • qualifier - String containing the • matchType - Stores the type of the match,

	<p>Returns:</p> <ul style="list-style-type: none"> • BOOL - Returns TRUE if match found. FALSE, otherwise.
QualifierMatches	<p>BOOL QualifierMatches (const NodeId nodeId, const AtmAddress* interfaceAddress, const char* qualifier, int* matchType)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • nodeId - nodeId to select for. • interfaceAddress - ATM address to select for. • qualifier - String containing the • matchType - Stores the type of the match, <p>Returns:</p> <ul style="list-style-type: none"> • BOOL - Returns TRUE if match found. FALSE, otherwise.
IO_ConvertIpv6StringToAddress()	<p>void IO_ConvertIpv6StringToAddress() (char* interfaceAddr, in6_addr* ipAddress)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • interfaceAddr - Storage for ipv6address string • ipAddress - Storage for ipv6address. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
IO_ConvertIpAddressToString	<p>void IO_ConvertIpAddressToString (in6_addr* ipAddress, char* interfaceAddr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • ipAddress - Storage for ipv6address. • interfaceAddr - Storage for ipv6address string <p>Returns:</p> <ul style="list-style-type: none"> • void - None
IO_ConvertIpAddressToString	<p>void IO_ConvertIpAddressToString (Address* ipAddress, char* interfaceAddr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • ipAddress - IP address info • interfaceAddr - Storage for ipv6address string <p>Returns:</p>

	<ul style="list-style-type: none"> • void - None
IO_ConvertStringToNodeAddress	<p>This API is used to convert a string parameter to NodeAddress.</p> <p>void IO_ConvertStringToNodeAddress (char* addressString, NodeAddress* outputNodeAddress)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • addressString - IP address string info • outputNodeAddress - Storage for IP address <p>Returns:</p> <ul style="list-style-type: none"> • void - None
IO_CheckIsSameAddress	<p>Compares IPv4 IPv6 address. API for Ipv6 compatibility.</p> <p>BOOL IO_CheckIsSameAddress (Address addr1, Address addr2)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • addr1 - Storage for IPv4 IPv6 address • addr2 - Storage for IPv4 IPv6 address <p>Returns:</p> <ul style="list-style-type: none"> • BOOL - None
IO_ReadString	<p>This API is used to retrieve a string parameter value from input files.</p> <p>void IO_ReadString (Node* node node, const NodeInput* nodeInput, const char* index, BOOL* wasFound, char* readVal)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Node pointer for which string is • nodeInput - Pointer to node input. • index - Parameter name. • wasFound - Storage for success of search. • readVal - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
IO_CacheFile	<p>This API is used to read an auxiliary input file into a NodeInput struct. Called for each file variable in the config file.</p> <p>void IO_CacheFile (const NodeInput* nodeInput, const char* filename)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • nodeInput - Pointer to node input. • filename - Path to input file. <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL

IO_GetMaxLen	<p>This API is used to get the maximum length of a line in the file.</p> <p>int IO_GetMaxLen (fileName char*)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>char*</code> - Pointer to the name of the file. <p>Returns:</p> <ul style="list-style-type: none"> • <code>int</code> - Integer with the largest line length.
IO_GetMaxLen	<p>This API is used to get the maximum length of a line in the file.</p> <p>int IO_GetMaxLen (fp FILE*)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>FILE*</code> - Pointer to a file stream. <p>Returns:</p> <ul style="list-style-type: none"> • <code>int</code> - Integer with the largest line length.
NI_GetMaxLen	<p>This API is used to get the maximum length of a line in nodeInput.</p> <p>int NI_GetMaxLen (nodeInput NodeInput*)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>NodeInput*</code> - Pointer to a node input. <p>Returns:</p> <ul style="list-style-type: none"> • <code>int</code> - Integer with the largest line length.
NI_GetMaxLen	<p>This API is used to get the maximum length of a line in nodeInput.</p> <p>int NI_GetMaxLen (nodeInput const NodeInput*)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>const NodeInput*</code> - Pointer to a node input. <p>Returns:</p> <ul style="list-style-type: none"> • <code>int</code> - Integer with the largest line length.
IO_ParseNetworkAddress	<p>Parses a string for a network address. Overloaded API for ATM compatibility.</p> <p>void IO_ParseNetworkAddress (const char s[], unsigned int* u_atmVal)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>s[]</code> - String to parse. • <code>u_atmVal</code> - Storage for icd, aid, ptpt <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
IO_ConvertAddrToString	void IO_ConvertAddrToString (Address* address, char* addrStr)

	<p>Convert generic address to appropriate network type address string format.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • address - generic address • addrStr - address string <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
IO_ConvertAtmAddressToString	<p>Convert Atm address to address string format.</p> <p>void IO_ConvertAtmAddressToString (AtmAddress addr, char* addrStr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • addr - Atm address • addrStr - address string <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
IO_InsertIntValue	<p>Insert integer value for specific string in case of ATM</p> <p>void IO_InsertIntValue (const char s[], const unsigned int val, unsigned int u_atmVal)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • s[] - character array • val - value to be inserted • u_atmVal - atm_value need to be checked <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
IO_ReadCachedFileIndex	<p>Return Cached file index for the given parameter name</p> <p>int IO_ReadCachedFileIndex (NodeAddress nodeId, NodeAddress interfaceAddress, unsigned int nodeInput)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • nodeId - node Id • interfaceAddress - Interface Address for the given node • nodeInput - atm_value need to be checked <p>Returns:</p> <ul style="list-style-type: none"> • int - None
IO_ReadString	<p>This API is used to retrieve a string parameter</p> <p>void IO_ReadString (Node* node, const NodeId nodeId, int interfaceIndex, const NodeInput* nodeInput, const char* parameterName, BOOL* wasFound, char* parameterValue)</p> <p>Parameters:</p>

<p>value from input files.</p>	<ul style="list-style-type: none"> • node - node structure pointer. • nodeId - nodeId. • interfaceIndex - interface Index. • nodeInput - Pointer to node input. • parameterName - Parameter name. • wasFound - Storage for success of seach. • parameterValue - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
<p>IO_ReadInt64</p> <p>This API is used to retrieve a Int64 parameter value from input files.</p>	<p>void IO_ReadInt64 (Node* node, const NodeId nodeId, int interfaceIndex, const NodeInput* nodeInput, const char* parameterName, BOOL* wasFound, Int64* parameterValue)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node structure pointer. • nodeId - nodeId. • interfaceIndex - interface Index. • nodeInput - Pointer to node input. • parameterName - Parameter name. • wasFound - Storage for success of seach. • parameterValue - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
<p>IO_ReadTime</p> <p>This API is used to retrieve a clocktype parameter value from input files.</p>	<p>void IO_ReadTime (Node* node, const NodeId nodeId, int interfaceIndex, const NodeInput* nodeInput, const char* parameterName, BOOL* wasFound, clocktype* parameterValue)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node structure pointer. • nodeId - nodeId. • interfaceIndex - interface Index. • nodeInput - Pointer to node input. • parameterName - Parameter name.

	<ul style="list-style-type: none"> • <code>wasFound</code> - Storage for success of seach. • <code>parameterValue</code> - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
IO_ReadInt	<pre>void IO_ReadInt (Node* node, const NodeId nodeId, int interfaceIndex, const NodeInput* nodeInput, const char* parameterName, BOOL* wasFound, int* parameterValue)</pre> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node structure pointer. • <code>nodeId</code> - nodeId. • <code>interfaceIndex</code> - interface Index. • <code>nodeInput</code> - Pointer to node input. • <code>parameterName</code> - Parameter name. • <code>wasFound</code> - Storage for success of seach. • <code>parameterValue</code> - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
IO_ReadDouble	<pre>void IO_ReadDouble (Node* node, const NodeId nodeId, int interfaceIndex, const NodeInput* nodeInput, const char* parameterName, BOOL* wasFound, double* parameterValue)</pre> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node structure pointer. • <code>nodeId</code> - nodeId. • <code>interfaceIndex</code> - interface Index. • <code>nodeInput</code> - Pointer to node input. • <code>parameterName</code> - Parameter name. • <code>wasFound</code> - Storage for success of seach. • <code>parameterValue</code> - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None

IO_ReadCachedFile <p>This API is used to retrieve a cached file parameter value from input files.</p>	<p>void IO_ReadCachedFile (Node* node, const NodeId nodeId, int interfaceIndex, const NodeInput* nodeInput, const char* parameterName, BOOL* wasFound, NodeInput* parameterValue)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node structure pointer. • nodeId - nodeId. • interfaceIndex - interface Index. • nodeInput - Pointer to node input. • parameterName - Parameter name. • wasFound - Storage for success of seach. • parameterValue - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
IO_ReadLine <p>This API is used to retrieve a whole line from input files.</p>	<p>void IO_ReadLine (Node* node, const NodeId nodeId, int interfaceIndex, const NodeInput* nodeInput, const char* parameterName, BOOL* wasFound, char* parameterValue)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node structure pointer. • nodeId - nodeId. • interfaceIndex - interface Index. • nodeInput - Pointer to node input. • parameterName - Parameter name. • wasFound - Storage for success of seach. • parameterValue - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
IO_ReadStringInstance <p>This API is used to retrieve string parameter values from input files for a specific instance.</p>	<p>void IO_ReadStringInstance (Node* node, const NodeId nodeId, int interfaceIndex, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, char* parameterValue)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node structure pointer. • nodeId - nodeId.

	<ul style="list-style-type: none"> • <code>interfaceIndex</code> - interface Index. • <code>nodeInput</code> - Pointer to node input. • <code>parameterName</code> - Parameter name. • <code>parameterInstanceNumber</code> - Instance number. • <code>fallbackIfNoInstanceMatch</code> - Selects parameter without • <code>wasFound</code> - Storage for success of seach. • <code>parameterValue</code> - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
IO_ReadDoubleInstance	<p>This API is used to retrieve double parameter values from input files for a specific instance.</p> <pre>void IO_ReadDoubleInstance (Node* node, const NodeId nodeId, int interfaceIndex, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, double* parameterValue)</pre> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node structure pointer. • <code>nodeId</code> - nodeId. • <code>interfaceIndex</code> - interface Index. • <code>nodeInput</code> - Pointer to node input. • <code>parameterName</code> - Parameter name. • <code>parameterInstanceNumber</code> - Instance number. • <code>fallbackIfNoInstanceMatch</code> - Selects parameter without • <code>wasFound</code> - Storage for success of seach. • <code>parameterValue</code> - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
IO_ReadIntInstance	<p>This API is used to retrieve int parameter values from input files for a specific instance.</p> <pre>void IO_ReadIntInstance (Node* node, const NodeId nodeId, int interfaceIndex, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, int* parameterValue)</pre> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node structure pointer. • <code>nodeId</code>

	<ul style="list-style-type: none"> - nodeId. • interfaceIndex - interface Index. • nodeInput - Pointer to node input. • parameterName - Parameter name. • parameterInstanceNumber - Instance number. • fallbackIfNoInstanceMatch - Selects parameter without • wasFound - Storage for success of seach. • parameterValue - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
IO_ReadTimeInstance	<p>void IO_ReadTimeInstance (Node* node, const NodeId nodeId, int interfaceIndex, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, clocktype* parameterValue)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node structure pointer. • nodeId - nodeId. • interfaceIndex - interface Index. • nodeInput - Pointer to node input. • parameterName - Parameter name. • parameterInstanceNumber - Instance number. • fallbackIfNoInstanceMatch - Selects parameter without • wasFound - Storage for success of seach. • parameterValue - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
IO_ReadCachedFileInstance	<p>void IO_ReadCachedFileInstance (Node* node, const NodeId nodeId, int interfaceIndex, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, NodeInput* parameterValue)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node structure pointer.

	<ul style="list-style-type: none"> • nodeId - nodeId. • interfaceIndex - interface Index. • nodeInput - Pointer to node input. • parameterName - Parameter name. • parameterInstanceNumber - Instance number. • fallbackIfNoInstanceMatch - Selects parameter without • wasFound - Storage for success of seach. • parameterValue - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
IO_ReadStringUsingIpAddress	<p>This API is used to retrieve a string parameter value from input files using the ip-address.</p> <pre>void IO_ReadStringUsingIpAddress (Node* node, int interfaceIndex, const NodeInput* nodeInput, const char* parameterName, BOOL* wasFound, char* parameterValue)</pre> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node structure pointer. • interfaceIndex - interface Index. • nodeInput - Pointer to node input. • parameterName - Parameter name. • wasFound - Storage for success of seach. • parameterValue - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
IO_ReadString	<p>This API is used to retrieve a string parameter value from input files.</p> <pre>void IO_ReadString (const NodeAddress nodeId, NodeAddress ipv4Address, in6_addr* ipv6Address, const NodeInput* nodeInput, const char* parameterName, BOOL* wasFound, char* parameterValue)</pre> <p>Parameters:</p> <ul style="list-style-type: none"> • nodeId - nodeId. • ipv4Address - IP address of an interface • ipv6Address - IPv6 address of an interface • nodeInput - Pointer to node input. • parameterName - Parameter name.

	<ul style="list-style-type: none"> • <code>wasFound</code> - Storage for success of search. • <code>parameterValue</code> - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
IO_ReadString	<p>This API is used to retrieve a string parameter value from input files.</p> <pre>void IO_ReadString (const NodeAddress nodeId, int interfaceIndex, const NodeAddress ipv4SubnetAddress, const in6_addr* ipv6SubnetAddress, const NodeInput* nodeInput, const char* parameterName, char* parameterValue, BOOL& wasFound, int& matchType)</pre> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>nodeId</code> - nodeId. • <code>interfaceIndex</code> - interface index • <code>ipv4SubnetAddress</code> - IPv4 subnet address • <code>ipv6SubnetAddress</code> - IPv6 subnet address • <code>nodeInput</code> - Pointer to node input. • <code>parameterName</code> - Parameter name. • <code>parameterValue</code> - Storage for parameter value. • <code>wasFound</code> - Storage for success of search. • <code>matchType</code> - Storage for matchType. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None



Contact [QualNet Support](#) for questions pertaining to the QualNet API Reference. This document is confidential and proprietary.
It may not be reproduced or distributed without the expressed written consent of [Scalable Network Technologies](#).

[QualNet](#)® is a Registered Trademark of [Scalable Network Technologies](#).

Copyright © 2001-2012 [Scalable Network Technologies, Inc.](#). All rights reserved.

QualNet 6.1 API Reference

GUI

This file describes data structures and functions for interfacing with the QualNet GUI and the other graphical tools.

Constant / Data Structure Summary

Type	Name
CONSTANT	GUI_DEFAULT_STEP
	The default interval before waiting for the Animator handshake/STEP.
CONSTANT	GUI_DEFAULT_ICON
	Icon used in case none is specified for a node.
CONSTANT	MAX_LAYERS
	By default, there are 8 layers, but users may add more
CONSTANT	GUI_DEFAULT_DATA_TYPE
	Default value to use for data types.
CONSTANT	GUI_EMULATION_DATA_TYPE
	Default value to use for data types.
CONSTANT	GUI_DEFAULT_LINK_TYPE
	Default value to use for link types.
CONSTANT	GUI_DEFAULT_NODE_TYPE
	Default value to use for node types.
CONSTANT	GUI_DEFAULT_INTERFACE
	Default interface for GUI commands.
CONSTANT	GUI_WIRELESS_LINK_TYPE

	Used to distinguish wireless and wired links.
CONSTANT	GUI_ATM_LINK_TYPE
	Used to distinguish ATM links from other types.
CONSTANT	GUI_COVERAGE_LINK_TYPE
	Used by Stats Manager
CONSTANT	GUI_MAX_COMMAND_LENGTH
	Maximum length for a single interchange with Animator.
ENUMERATION	GuiLayers
	Layer in protocol stack. Allows animation filtering.
ENUMERATION	GuiEvents
	Semantic events to be animated.
ENUMERATION	GuiStatisticsEvents
	Statistics events recognized by Animator.
ENUMERATION	GuiMetrics
	Types of statistical metrics.
ENUMERATION	GuiDataTypes
	The numeric data types supported for dynamic statistics.
ENUMERATION	GuiEffects
	Animation effects that can be assigned to an event.
ENUMERATION	GuiColors
	Colors that can be assigned to Animator effects.
ENUMERATION	GuiSubnetTypes

	Types of subnets recognized by the Animator. GuiVisObjCommands
ENUMERATION	Commands for displaying visualization objects GuiVisShapes
ENUMERATION	Shape selections for GUI_DRAW_SHAPE command GuiCommands
ENUMERATION	Coded commands sent from Animator to Simulator. GuiReplies
ENUMERATION	Coded commands sent from Simulator to Animator. GuiReply
STRUCT	Structure containing message sent to Animator. MetricData
STRUCT	Class to identify a specific dynamic statistic. MetricLayerData
STRUCT	Contains a list of the metrics collected at a layer of the protocol stack. GuiCommand
	Structure containing command received from Animator.

Function / Macro Summary

Return Type	Summary
void	<p>GUI_HandleHTLInput(const char * args, PartitionData * partition)</p> <p>Called from GUI_EXTERNAL_ReceiveCommand() if command type is GUI_USER_DEFINED. Created so that GUI Human In the loop commands can also be given through a file, instead of giving it through the GUI. Will serve for good unit testing of GUI HTL</p>

	commands
void	<code>GUI_Initialize</code> (NodeInput* nodeInput, int numNodes, int coordinateSystem, Coordinates origin, Coordinates dimensions, clocktype maxClock)
	Initializes the GUI in order to start the animation. The terrain map should give the path (either absolute, or relative to <code>QUALNET_HOME</code>) of a file to represent the terrain.
void	<code>GUI_SetEffect</code> (GuiEvents event, GuiLayers layer, int type, GuiEffects effect, GuiColors color)
	This function will allow the protocol designer to specify the effect to use to display certain events.
void	<code>GUI_InitNode</code> (Node* node, NodeInput* nodeInput, clocktype time)
	Provides the initial location and orientation of the node, the transmission range (for wireless nodes), a node type, and optional icon and label.
void	<code>GUI_InitWirelessInterface</code> (Node* node, int interfaceIndex)
	Provides the initial location and orientation of the node, the transmission range (for wireless nodes), a node type, and optional icon and label.
void	<code>GUI_InitializeInterfaces</code> (NodeInput* nodeInput)
	Sets the IP address associated with one of the node's interfaces.
void	<code>GUI_SetInterfaceAddress</code> (NodeId nodeID, NodeAddress interfaceAddress, int interfaceIndex, clocktype time)
	Sets the IP address associated with one of the node's interfaces.
void	<code>GUI_SetSubnetMask</code> (NodeId nodeID, NodeAddress interfaceAddress, int interfaceIndex, clocktype time)
	Sets the Subnet mask associated with one of the node's interfaces.
void	<code>GUI_SetInterfaceName</code> (NodeId nodeID, char* interfaceAddress, int interfaceIndex, clocktype time)
	Sets the Interface name associated with one of the node's interfaces.
void	<code>GUI_MoveNode</code> (NodeId nodeID, Coordinates position, clocktype time)
	Moves the node to a new position.
void	<code>GUI_SetNodeOrientation</code> (NodeId nodeID, Orientation orientation, clocktype time)
	Changes the orientation of a node.
void	<code>GUI_SetNodeIcon</code> (NodeId nodeID, char* iconFile, clocktype time)

	Changes the icon associated with a node.
void	GUI_SetNodeLabel (NodeId nodeID, char* label, clocktype time)
	Changes the label (the node name) of a node.
void	GUI_SetNodeRange (NodeId nodeID, int interfaceIndex, double range, clocktype time)
	Changes the transmission range of a node
void	GUI_SetNodeType (NodeId nodeID, int type, clocktype time)
	Changes the (symbolic) type of a node
void	GUI_SetPatternIndex (Node* node, int interfaceIndex, int index, clocktype time)
	Sets the antenna pattern to one of a previously specified antenna pattern file.
void	GUI_SetPatternAndAngle (node node*, int interfaceIndex, int index, int angleInDegrees, clocktype time)
	For steerable antennas, it sets the pattern to use, and also an angle relative to the node's current orientation.
void	GUI_AddLink (NodeId sourceID, NodeId destID, GuiLayers layer, int type, NodeAddress subnetAddress, int numHostBits, clocktype time)
	Adds a link (one hop on a route) between two nodes. In a wired topology, this could be a static route; in wireless, a dynamic one.
void	GUI_AddLink (NodeId sourceID, NodeId destID, GuiLayers layer, int type, int tla, int nla, int sla, clocktype time)
	Adds an IPv6 link (one hop on a route) between two nodes. In a wired topology, this could be a static route; in wireless, a dynamic one.
void	GUI_AddLink (NodeId sourceID, NodeId destID, GuiLayers layer, int type, IPv6subnetAddress ip6_addr, IPv6subnetPrefixLen unsigned int, clocktype time)
	Adds an IPv6 link (one hop on a route) between two nodes. In a wired topology, this could be a static route; in wireless, a dynamic one.
void	GUI_DeleteLink (NodeId sourceID, NodeId destID, GuiLayers layer, int type, clocktype time)
	Removes link of a specific type.
void	GUI_DeleteLink (NodeId sourceID, NodeId destID, GuiLayers layer, clocktype time)
	Removes the aforementioned link, no matter the "type."

void	GUI_Broadcast (NodeId nodeID, GuiLayers layer, int type, int interfaceIndex, clocktype time)
	Indicates a broadcast.
void	GUI_EndBroadcast (NodeId nodeID, GuiLayers layer, int type, int interfaceIndex, clocktype time)
	Indicates the end of a broadcast.
void	GUI_Multicast (NodeId nodeID, GuiLayers layer, int type, int interfaceIndex, clocktype time)
	Indicates a multicast. (Probably need to add a destination address.)
void	GUI_Uncast (NodeId sourceID, NodeId destID, GuiLayers layer, int type, int sendingInterfaceIndex, int receivingInterfaceIndex, clocktype time)
	Sends a unicast packet/frame/signal to a destination. Will probably be drawn as a temporary line between source and destination, followed by a signal (at the receiver) indicating success or failure.
void	GUI_Receive (NodeId sourceID, NodeId destID, GuiLayers layer, int type, int sendingInterfaceIndex, int receivingInterfaceIndex, clocktype time)
	Shows a successful receipt at a destination.
void	GUI_Drop (NodeId sourceID, NodeId destID, GuiLayers layer, int type, int sendingInterfaceIndex, int receivingInterfaceIndex, clocktype time)
	Shows a packet/frame/signal being dropped by a node.
void	GUI_Collision (NodeId nodeID, GuiLayers layer, clocktype time)
	Shows a node detecting a collision.
void	GUI_CreateSubnet (GuiSubnetTypes type, NodeAddress subnetAddress, int numHostBits, const char* nodeList, clocktype time)
	Creates a subnet. Normally done at startup.
void	GUI_CreateSubnet (GuiSubnetTypes type, in6_addr IPv6subnetAddress, unsigned int IPv6subnetPrefixLen, const char* nodeList, clocktype time)
	Creates a IPv6 subnet. Normally done at startup.
void	GUI_CreateSubnet (GuiSubnetTypes type, IPv6subnetAddress ip6_addr, IPv6subnetPrefixLen unsigned int, const char* nodeList, clocktype time)
	Creates a IPv6 subnet. Normally done at startup.
void	GUI_CreateHierarchy (int componentID, char* nodeList)

	Since the GUI supports hierarchical design, this function informs the GUI of the contents of a hierarchical component.
void	<code>GUI_MoveHierarchy(int hierarchyId, Coordinates centerCoordinates, Orientation orientation, clocktype time)</code> Moves the center point of a hierarchy to a new position.
void	<code>GUI_CreateWeatherPattern(int patternID, char* inputLine)</code> Sends the input line describing a weather pattern to the GUI.
void	<code>GUI_MoveWeatherPattern(int patternID, Coordinates coordinates, clocktype time)</code> Moves the first point of a weather pattern to a new position.
void	<code>GUI_AddApplication(NodeId sourceID, NodeId destID, char* appName, int uniqueId, clocktype time)</code> Shows label beside the client and the server as app link is setup.
void	<code>GUI_DeleteApplication(NodeId sourceID, NodeId destID, char* appName, int uniqueId, clocktype time)</code> Deletes the labels shown by AddApplication.
void	<code>GUI_AddInterfaceQueue(NodeId nodeID, GuiLayers layer, int interfaceIndex, unsigned priority, int queueSize, clocktype time)</code> Creates a queue for a node, interface and priority.
void	<code>GUI_QueueInsertPacket(NodeId nodeID, GuiLayers layer, int interfaceIndex, unsigned priority, int packetSize, clocktype time)</code> Inserting one packet to a queue for a node, interface and priority
void	<code>GUI_QueueDropPacket(NodeId nodeID, GuiLayers layer, int interfaceIndex, unsigned priority, clocktype time)</code> Dropping one packet from a queue for a node, interface and priority.
void	<code>GUI_QueueDequeuePacket(NodeId nodeID, GuiLayers layer, int interfaceIndex, unsigned priority, int packetSize, clocktype time)</code> Dequeuing one packet from a queue for a node, interface and priority
int	<code>GUI_DefineMetric(char* name, NodeId nodeID, GuiLayers layer, int linkID, GuiDataTypes datatype, GuiMetrics metricType)</code>

	This function defines a metric by giving it a name and a description. The system will assign a number to this data item. Future references to the data should use the number rather than the name. The link ID will be used to associate a metric with a particular application link, or MAC interface, etc.
void	GUI_SendIntegerData (NodeId nodeID, int metricID, int value, clocktype time) Sends data for an integer metric.
void	GUI_SendUnsignedData (NodeId nodeID, int metricID, unsigned value, clocktype time) Sends data for an unsigned metric.
void	GUI_SendRealData (NodeId nodeID, int metricID, double value, clocktype time) Sends data for a floating point metric.
bool	GUI_isAnimateOrInteractive () Returns true if the GUI was activated on the command line.
void	GUI_EXTERNAL_Bootstrap (int argc, char** argv, NodeInput* nodeInput, int numberOfProcessors, int thisPartitionId) Creates a connection to the GUI
void	GUI_EXTERNAL_Registration (PartitionData* partitionData, EXTERNAL_InterfaceList* list) Registers the GUI as an external interface
void	GUI_CreateReply (GuiReplies replyType, std msg) Function used to replace newline characters in a string being sent to the GUI.

Constant / Data Structure Detail

Constant	GUI_DEFAULT_STEP 1 econds The default interval before waiting for the Animator handshake/STEP.
Constant	GUI_DEFAULT_ICON "" Icon used in case none is specified for a node.

Constant	<code>MAX_LAYERS 12</code>
	By default, there are 8 layers, but users may add more
Constant	<code>GUI_DEFAULT_DATA_TYPE 0</code>
	Default value to use for data types.
Constant	<code>GUI_EMULATION_DATA_TYPE 1</code>
	Default value to use for data types.
Constant	<code>GUI_DEFAULT_LINK_TYPE 0</code>
	Default value to use for link types.
Constant	<code>GUI_DEFAULT_NODE_TYPE 0</code>
	Default value to use for node types.
Constant	<code>GUI_DEFAULT_INTERFACE 0</code>
	Default interface for GUI commands.
Constant	<code>GUI_WIRELESS_LINK_TYPE 1</code>
	Used to distinguish wireless and wired links.
Constant	<code>GUI_ATM_LINK_TYPE 2</code>
	Used to distinguish ATM links from other types.
Constant	<code>GUI_COVERAGE_LINK_TYPE 3</code>
	Used by Stats Manager
Constant	<code>GUI_MAX_COMMAND_LENGTH 1024</code>
	Maximum length for a single interchange with Animator.
Enumeration	<code>GuiLayers</code>

	Layer in protocol stack. Allows animation filtering.
Enumeration	<p>GuiEvents</p> <p>Semantic events to be animated.</p>
Enumeration	<p>GuiStatisticsEvents</p> <p>Statistics events recognized by Animator.</p>
Enumeration	<p>GuiMetrics</p> <p>Types of statistical metrics.</p>
Enumeration	<p>GuiDataTypes</p> <p>The numeric data types supported for dynamic statistics.</p>
Enumeration	<p>GuiEffects</p> <p>Animation effects that can be assigned to an event.</p>
Enumeration	<p>GuiColors</p> <p>Colors that can be assigned to Animator effects.</p>
Enumeration	<p>GuiSubnetTypes</p> <p>Types of subnets recognized by the Animator.</p>
Enumeration	<p>GuiVisObjCommands</p> <p>Commands for displaying visualization objects</p>
Enumeration	<p>GuiVisShapes</p> <p>Shape selections for GUI_DRAW_SHAPE command</p>
Enumeration	GuiCommands

	Coded commands sent from Animator to Simulator.
Enumeration	GuiReplies
	Coded commands sent from Simulator to Animator.
Enumeration	GuiReply
	Structure containing message sent to Animator.
Structure	MetricData
	Class to identify a specific dynamic statistic.
Structure	MetricLayerData
	Contains a list of the metrics collected at a layer of the protocol stack.
Structure	GuiCommand
	Structure containing command received from Animator.

Function / Macro Detail

Function / Macro	Format
GUI_HandleHITLInput	<p>Called from <code>GUI_EXTERNAL_ReceiveCommand()</code> if command type is <code>GUI_USER_DEFINED</code>. Created so that GUI Human In the loop commands can also be given through a file, instead of giving it through the GUI. Will serve for good unit testing of GUI HITL commands</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>args</code> - the command itself • <code>partition</code> - the partition pointer <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
GUI_Initialize	<p>Initializes the GUI in order to start the animation. The terrain map should give the path (either absolute, or relative to <code>QUALNET_HOME</code>) of an file to represent</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>nodeInput</code> - configuration file • <code>numNodes</code> - the number of nodes in the simulation

<p>the terrain.</p>	<ul style="list-style-type: none"> • coordinateSystem - LATLONALT or CARTESIAN • origin - Southwest corner • dimensions - Northeast corner, or size • maxClock - length of the simulation <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
<p>GUI_SetEffect</p> <p>This function will allow the protocol designer to specify the effect to use to display certain events.</p>	<p>void GUI_SetEffect (GuiEvents event, GuiLayers layer, int type, GuiEffects effect, GuiColors color)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • event - the type of event for the new effect • layer - the protocol layer • type - special key to distinguish similar events • effect - the effect to use • color - optional color for the effect <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
<p>GUI_InitNode</p> <p>Provides the initial location and orientation of the node, the transmission range (for wireless nodes), a node type, and optional icon and label.</p>	<p>void GUI_InitNode (Node* node, NodeInput* nodeInput, clocktype time)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - the node • nodeInput - configuration file • time - the current simulation time <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
<p>GUI_InitWirelessInterface</p> <p>Provides the initial location and orientation of the node, the transmission range (for wireless nodes), a node type, and optional icon and label.</p>	<p>void GUI_InitWirelessInterface (Node* node, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - the node • interfaceIndex - the interface to initialize <p>Returns:</p>

	<ul style="list-style-type: none"> • void - NULL
GUI_InitializeInterfaces	<p>Sets the IP address associated with one of the node's interfaces.</p> <p>void GUI_InitializeInterfaces (NodeInput* nodeInput)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • nodeInput - configuration file <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
GUI_SetInterfaceAddress	<p>Sets the IP address associated with one of the node's interfaces.</p> <p>void GUI_SetInterfaceAddress (NodeId nodeID, NodeAddress interfaceAddress, int interfaceIndex, clocktype time)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • nodeID - the node's ID • interfaceAddress - new IP address • interfaceIndex - interface Address to change • time - the current simulation time <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
GUI_SetSubnetMask	<p>Sets the Subnet mask associated with one of the node's interfaces.</p> <p>void GUI_SetSubnetMask (NodeId nodeID, NodeAddress interfaceAddress, int interfaceIndex, clocktype time)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • nodeID - the node's ID • interfaceAddress - new Subnet mask • interfaceIndex - Subnet mask to change • time - the current simulation time <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
GUI_SetInterfaceName	<p>Sets the Interface name associated with one of the node's interfaces.</p> <p>void GUI_SetInterfaceName (NodeId nodeID, char* interfaceAddress, int interfaceIndex, clocktype time)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • nodeID - the node's ID • interfaceAddress - new Interface name • interfaceIndex - interface Name to change • time - the current simulation time

	<p>Returns:</p> <ul style="list-style-type: none"> • <code>void - NULL</code>
GUI_MoveNode	<p><code>void GUI_MoveNode (NodeId nodeID, Coordinates position, clocktype time)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>nodeID</code> - the node's ID • <code>position</code> - the new position • <code>time</code> - the current simulation time <p>Returns:</p> <ul style="list-style-type: none"> • <code>void - NULL</code>
GUI_SetNodeOrientation	<p><code>void GUI_SetNodeOrientation (NodeId nodeID, Orientation orientation, clocktype time)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>nodeID</code> - the node's ID • <code>orientation</code> - the new orientation • <code>time</code> - the current simulation time <p>Returns:</p> <ul style="list-style-type: none"> • <code>void - NULL</code>
GUI_SetNodeIcon	<p><code>void GUI_SetNodeIcon (NodeId nodeID, char* iconFile, clocktype time)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>nodeID</code> - the node's ID • <code>iconFile</code> - the path to the image file, may be the • <code>time</code> - the current simulation time <p>Returns:</p> <ul style="list-style-type: none"> • <code>void - NULL</code>
GUI_SetNodeLabel	<p><code>void GUI_SetNodeLabel (NodeId nodeID, char* label, clocktype time)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>nodeID</code> - the node's ID • <code>label</code> - a string to label the node

	<ul style="list-style-type: none"> • <code>time</code> - the current simulation time <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
GUI_SetNodeRange	<p>Changes the transmission range of a node</p> <p>void GUI_SetNodeRange (NodeId nodeID, int interfaceIndex, double range, clocktype time)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>nodeID</code> - the node's ID • <code>interfaceIndex</code> - which of the node's interfaces to use • <code>range</code> - the new transmission range in meters • <code>time</code> - the current simulation time <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
GUI_SetNodeType	<p>Changes the (symbolic) type of a node</p> <p>void GUI_SetNodeType (NodeId nodeID, int type, clocktype time)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>nodeID</code> - the node's ID • <code>type</code> - user defined type, used with <code>GUI_SetEffect</code> • <code>time</code> - the current simulation time <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
GUI_SetPatternIndex	<p>Sets the antenna pattern to one of a previously specified antenna pattern file.</p> <p>void GUI_SetPatternIndex (Node* node, int interfaceIndex, int index, clocktype time)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - the node pointer • <code>interfaceIndex</code> - which of the node's interfaces to use • <code>index</code> - index into the node's antenna pattern file • <code>time</code> - the current simulation time <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
GUI_SetPatternAndAngle	<p>void GUI_SetPatternAndAngle (node node*, int interfaceIndex, int index, int angleInDegrees, clocktype time)</p> <p>Parameters:</p>

<p>For steerable antennas, it sets the pattern to use, and also an angle relative to the node's current orientation.</p>	<ul style="list-style-type: none"> • <code>node*</code> - the node pointer • <code>interfaceIndex</code> - which of the node's interfaces to use • <code>index</code> - index into the node's antenna pattern file • <code>angleInDegrees</code> - angle to rotate the pattern • <code>time</code> - the current simulation time <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
<p>GUI_AddLink</p> <p>Adds a link (one hop on a route) between two nodes. In a wired topology, this could be a static route; in wireless, a dynamic one.</p>	<p>void GUI_AddLink (NodeId sourceID, NodeId destID, GuiLayers layer, int type, NodeAddress subnetAddress, int numHostBits, clocktype time)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>sourceID</code> - the source node for the link • <code>destID</code> - the destination node • <code>layer</code> - the protocol layer associated w/ the link • <code>type</code> - a user-defined type for the link • <code>subnetAddress</code> - subnet address for network links • <code>numHostBits</code> - subnet size for network links • <code>time</code> - the current simulation time <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
<p>GUI_AddLink</p> <p>Adds an IPv6 link (one hop on a route) between two nodes. In a wired topology, this could be a static route; in wireless, a dynamic one.</p>	<p>void GUI_AddLink (NodeId sourceID, NodeId destID, GuiLayers layer, int type, int tla, int nla, int sla, clocktype time)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>sourceID</code> - the source node for the link • <code>destID</code> - the destination node • <code>layer</code> - the protocol layer associated w/ the link • <code>type</code> - a user-defined type for the link • <code>tla</code> - TLA field of IPv6 address • <code>nla</code> - NLA field of IPv6 address • <code>sla</code> - SLA field of IPv6 address

	<ul style="list-style-type: none"> • <code>time</code> - the current simulation time <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
GUI_AddLink	<p>Adds an IPv6 link (one hop on a route) between two nodes. In a wired topology, this could be a static route; in wireless, a dynamic one.</p> <p><code>void GUI_AddLink (NodeId sourceID, NodeId destID, GuiLayers layer, int type, IPv6subnetAddress ip6_addr, IPv6subnetPrefixLen unsigned int, clocktype time)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>sourceID</code> - the source node for the link • <code>destID</code> - the destination node • <code>layer</code> - the protocol layer associated w/ the link • <code>type</code> - a user-defined type for the link • <code>ip6_addr</code> - IPv6 address • <code>unsigned int</code> - IPv6 address prefix length • <code>time</code> - the current simulation time <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
GUI_DeleteLink	<p>Removes link of a specific type.</p> <p><code>void GUI_DeleteLink (NodeId sourceID, NodeId destID, GuiLayers layer, int type, clocktype time)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>sourceID</code> - the source node for the link • <code>destID</code> - the destination node • <code>layer</code> - the protocol layer associated w/ the link • <code>type</code> - type of link being deleted • <code>time</code> - the current simulation time <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
GUI_DeleteLink	<p>Removes the aforementioned link, no matter the "type."</p> <p><code>void GUI_DeleteLink (NodeId sourceID, NodeId destID, GuiLayers layer, clocktype time)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>sourceID</code> - the source node for the link • <code>destID</code> - the destination node

	<ul style="list-style-type: none"> • <code>layer</code> - the protocol layer associated w/ the link • <code>time</code> - the current simulation time <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
GUI_Broadcast	<p><code>void GUI_Broadcast (NodeId nodeID, GuiLayers layer, int type, int interfaceIndex, clocktype time)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>nodeID</code> - the node's ID • <code>layer</code> - the protocol layer associated w/ event • <code>type</code> - a user-defined type for the link • <code>interfaceIndex</code> - which of the node's interfaces to use • <code>time</code> - the current simulation time <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
GUI_EndBroadcast	<p><code>void GUI_EndBroadcast (NodeId nodeID, GuiLayers layer, int type, int interfaceIndex, clocktype time)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>nodeID</code> - the node's ID • <code>layer</code> - the protocol layer associated w/ event • <code>type</code> - a user-defined type for the link • <code>interfaceIndex</code> - which of the node's interfaces to use • <code>time</code> - the current simulation time <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
GUI_Multicast	<p><code>void GUI_Multicast (NodeId nodeID, GuiLayers layer, int type, int interfaceIndex, clocktype time)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>nodeID</code> - the node's ID • <code>layer</code> - the protocol layer associated w/ event • <code>type</code> - a user-defined type for the link • <code>interfaceIndex</code> - which of the node's interfaces to use

	<ul style="list-style-type: none"> • <code>time</code> - the current simulation time <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
GUI_Uncast	<p>Sends a unicast packet/frame/signal to a destination. Will probably be drawn as a temporary line between source and destination, followed by a signal (at the receiver) indicating success or failure.</p> <p><code>void GUI_Uncast (NodeId sourceID, NodeId destID, GuiLayers layer, int type, int sendingInterfaceIndex, int receivingInterfaceIndex, clocktype time)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>sourceID</code> - the source node • <code>destID</code> - the destination node • <code>layer</code> - protocol layer associated w/ the event • <code>type</code> - a user-defined type • <code>sendingInterfaceIndex</code> - sender's interface to use • <code>receivingInterfaceIndex</code> - receiver's interface to use • <code>time</code> - the current simulation time <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
GUI_Receive	<p>Shows a successful receipt at a destination.</p> <p><code>void GUI_Receive (NodeId sourceID, NodeId destID, GuiLayers layer, int type, int sendingInterfaceIndex, int receivingInterfaceIndex, clocktype time)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>sourceID</code> - the source node • <code>destID</code> - the destination node • <code>layer</code> - protocol layer associated w/ the event • <code>type</code> - a user-defined type • <code>sendingInterfaceIndex</code> - sender's interface to use • <code>receivingInterfaceIndex</code> - receiver's interface to use • <code>time</code> - the current simulation time <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
GUI_Drop	<p><code>void GUI_Drop (NodeId sourceID, NodeId destID, GuiLayers layer, int type, int sendingInterfaceIndex, int receivingInterfaceIndex, clocktype time)</code></p>

	<p>Shows a packet/frame/signal being dropped by a node.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>sourceID</code> - the source node • <code>destID</code> - the destination node • <code>layer</code> - protocol layer associated w/ the event • <code>type</code> - a user-defined type • <code>sendingInterfaceIndex</code> - sender's interface to use • <code>receivingInterfaceIndex</code> - receiver's interface to use • <code>time</code> - the current simulation time <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
GUI_Collision	<p>Shows a node detecting a collision.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>nodeID</code> - the node's ID • <code>layer</code> - the protocol layer associated w/ event • <code>time</code> - the current simulation time <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
GUI_CreateSubnet	<p>Creates a subnet. Normally done at startup.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>type</code> - GUI_WIRED/WIRELESS/SATELLITE_NETWORK • <code>subnetAddress</code> - base address for the subnet • <code>numHostBits</code> - number of host bits for subnet mask • <code>nodeList</code> - the rest of the .config file SUBNET line • <code>time</code> - the current simulation time <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
GUI_CreateSubnet	<p>Parameters:</p> <ul style="list-style-type: none"> • <code>type</code> - GuiSubnetTypes type, in6_addr IPv6subnetAddress, unsigned int IPv6subnetPrefixLen, const

	<p>Creates a IPv6 subnet. Normally done at startup.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>type</code> - GUI_WIRED/WIRELESS/SATELLITE_NETWORK • <code>IPv6subnetAddress</code> - base address for the subnet • <code>IPv6subnetPrefixLen</code> - number of network bits present • <code>nodeList</code> - the rest of the .config file SUBNET line • <code>time</code> - the current simulation time <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
GUI_CreateSubnet	<p>Creates a IPv6 subnet. Normally done at startup.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>type</code> - GUI_WIRED/WIRELESS/SATELLITE_NETWORK • <code>ip6_addr</code> - IPv6 address • <code>unsigned int</code> - IPv6 address prefix length • <code>nodeList</code> - the rest of the .config file SUBNET line • <code>time</code> - the current simulation time <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
GUI_CreateHierarchy	<p>Since the GUI supports hierarchical design, this function informs the GUI of the contents of a hierarchical component.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>componentID</code> - an identifier for the hierarchy • <code>nodeList</code> - the rest of the .config file COMPONENT line <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
GUI_MoveHierarchy	<p>Moves the center point of a hierarchy to a new position.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>hierarchyId</code> - the hierarchy's ID

	<ul style="list-style-type: none"> • <code>centerCoordinates</code> - the new position • <code>orientation</code> - the new orientation • <code>time</code> - the current simulation time <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
GUI_CreateWeatherPattern	<p>Sends the input line describing a weather pattern to the GUI.</p> <p><code>void GUI_CreateWeatherPattern (int patternID, char* inputLine)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>patternID</code> - the weather pattern ID • <code>inputLine</code> - the .weather file line <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
GUI_MoveWeatherPattern	<p>Moves the first point of a weather pattern to a new position.</p> <p><code>void GUI_MoveWeatherPattern (int patternID, Coordinates coordinates, clocktype time)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>patternID</code> - the weather pattern ID • <code>coordinates</code> - the new position • <code>time</code> - the current simulation time <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
GUI_AddApplication	<p>Shows label beside the client and the server as app link is setup.</p> <p><code>void GUI_AddApplication (NodeId sourceID, NodeId destID, char* appName, int uniqueId, clocktype time)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>sourceID</code> - the source node • <code>destID</code> - the destination node • <code>appName</code> - the application name, e.g. "CBR" • <code>uniqueId</code> - unique label for this application session • <code>time</code> - the current simulation time <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
GUI_DeleteApplication	<code>void GUI_DeleteApplication (NodeId sourceID, NodeId destID, char* appName, int uniqueId, clocktype time)</code>

	<p>Deletes the labels shown by AddApplication.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>sourceID</code> - the source node • <code>destID</code> - the destination node • <code>appName</code> - the application name, e.g. "CBR" • <code>uniqueID</code> - unique label for this application session • <code>time</code> - the current simulation time <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
GUI_AddInterfaceQueue Creates a queue for a node, interface and priority.	<p>void GUI_AddInterfaceQueue (NodeId nodeID, GuiLayers layer, int interfaceIndex, unsigned priority, int queueSize, clocktype time)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>nodeID</code> - the node's ID • <code>layer</code> - protocol layer associated w/ the event • <code>interfaceIndex</code> - associated interface of node • <code>priority</code> - priority of queue • <code>queueSize</code> - maximum size in bytes • <code>time</code> - the current simulation time <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
GUI_QueueInsertPacket Inserting one packet to a queue for a node, interface and priority	<p>void GUI_QueueInsertPacket (NodeId nodeID, GuiLayers layer, int interfaceIndex, unsigned priority, int packetSize, clocktype time)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>nodeID</code> - the node's ID • <code>layer</code> - protocol layer associated w/ the event • <code>interfaceIndex</code> - associated interface of node • <code>priority</code> - priority of queue • <code>packetSize</code> - size of packet • <code>time</code> - the current simulation time

	<p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
GUI_QueueDropPacket	<p>Dropping one packet from a queue for a node, interface and priority.</p> <p>void GUI_QueueDropPacket (NodeId nodeID, GuiLayers layer, int interfaceIndex, unsigned priority, clocktype time)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • nodeID - the node's ID • layer - protocol layer associated w/ the event • interfaceIndex - associated interface of node • priority - priority of queue • time - the current simulation time <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
GUI_QueueDequeuePacket	<p>Dequeuing one packet from a queue for a node, interface and priority</p> <p>void GUI_QueueDequeuePacket (NodeId nodeID, GuiLayers layer, int interfaceIndex, unsigned priority, int packetSize, clocktype time)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • nodeID - the node's ID • layer - protocol layer associated w/ the event • interfaceIndex - associated interface of node • priority - priority of queue • packetSize - size of packet • time - the current simulation time <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
GUI_DefineMetric	<p>This function defines a metric by giving it a name and a description. The system will assign a number to this data item. Future references to the data should use the number rather than the name. The link ID will be used to associate a metric with a particular application link, or MAC interface, etc.</p> <p>int GUI_DefineMetric (char* name, NodeId nodeID, GuiLayers layer, int linkID, GuiDataTypes datatype, GuiMetrics metrictype)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • name - the name of the metric • nodeID - the node's ID • layer - protocol layer associated w/ the event • linkID - e.g., an application session ID

	<ul style="list-style-type: none"> • <code>datatype</code> - real/unsigned/integer • <code>metrictype</code> - cumulative/average, etc. <p>Returns:</p> <ul style="list-style-type: none"> • <code>int</code> - an identifier associated with the metric name and layer
GUI_SendIntegerData	<p>Sends data for an integer metric.</p> <p>void GUI_SendIntegerData (NodeId nodeID, int metricID, int value, clocktype time)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>nodeID</code> - the node's ID • <code>metricID</code> - the value returned by DefineMetric • <code>value</code> - the current value of the metric • <code>time</code> - the current simulation time <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
GUI_SendUnsignedData	<p>Sends data for an unsigned metric.</p> <p>void GUI_SendUnsignedData (NodeId nodeID, int metricID, unsigned value, clocktype time)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>nodeID</code> - the node's ID • <code>metricID</code> - the value returned by DefineMetric • <code>value</code> - the current value of the metric • <code>time</code> - the current simulation time <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
GUI_SendRealData	<p>Sends data for a floating point metric.</p> <p>void GUI_SendRealData (NodeId nodeID, int metricID, double value, clocktype time)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>nodeID</code> - the node's ID • <code>metricID</code> - the value returned by DefineMetric • <code>value</code> - the current value of the metric • <code>time</code> - the current simulation time <p>Returns:</p>

	void - NULL
GUI_isAnimateOrInteractive	<p>bool GUI_isAnimateOrInteractive ()</p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> • bool - True if the GUI is enabled.
GUI_EXTERNAL_Bootstrap	<p>void GUI_EXTERNAL_Bootstrap (int argc, char** argv, NodeInput* nodeInput, int numberOfProcessors, int thisPartitionId)</p> <p>Creates a connection to the GUI</p> <p>Parameters:</p> <ul style="list-style-type: none"> • argc - number of command line parameters • argv - command line parameters • nodeInput - the contents of the .config file • numberOfProcessors - the number of processors in use • thisPartitionId - the ID of this partition <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
GUI_EXTERNAL_Registration	<p>void GUI_EXTERNAL_Registration (PartitionData* partitionData, EXTERNAL_InterfaceList* list)</p> <p>Registers the GUI as an external interface</p> <p>Parameters:</p> <ul style="list-style-type: none"> • partitionData - the partition to register with • list - the list to add oneself to <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
GUI_CreateReply	<p>void GUI_CreateReply (GuiReplies replyType, std msg)</p> <p>Function used to replace newline characters in a string being sent to the GUI.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • replyType - the type of reply • msg - string* <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL



Contact [QualNet Support](#) for questions pertaining to the QualNet API Reference. This document is confidential and proprietary.
It may not be reproduced or distributed without the expressed written consent of [Scalable Network Technologies](#).

[QualNet](#)® is a Registered Trademark of [Scalable Network Technologies, Inc.](#)

Copyright © 2001-2012 [Scalable Network Technologies, Inc.](#). All rights reserved.

QualNet 6.1 API Reference

IP

This file contains data structures and prototypes of functions used by IP.

Constant / Data Structure Summary

Type	Name
CONSTANT	IPVERSION4 Version of IP
CONSTANT	IPTOS_LOWDELAY Type of service (low delay)
CONSTANT	IPTOS_THROUGHPUT Type of service (throughput)
CONSTANT	IPTOS_RELIABILITY Type of service (reliability)
CONSTANT	IPTOS_MINCOST Type of service (minimum cost)
CONSTANT	IPTOS_ECT Bits 6 and 7 in the IPv4 TOS octet are designated as the ECN field. Bit 6 is designated as the ECT bit.
CONSTANT	IPTOS_CE Bits 6 and 7 in the IPv4 TOS octet are designated as the ECN field. Bit 7 is designated as the CE bit.
CONSTANT	IPTOS_DSCP_MAX Bits 0 to 5 in the IPv4 TOS octet are designated as DSCP field.The range for this 6-bit field is < 0 - 63 >.
CONSTANT	IPTOS_DSCP_MIN

	Bits 0 to 5 in the IPv4 TOS octet are designated as DSCP field.The range for this 6-bit field is < 0 - 63 >.
CONSTANT	<u>IPTOS_PREC_EFINTERNETCONTROL</u>
CONSTANT	IP precedence 'EF classe internet control' <u>IPTOS_PREC_NETCONTROL</u>
CONSTANT	IP precedence 'net control' <u>IPTOS_PREC_INTERNETCONTROL</u>
CONSTANT	IP precedence 'internet control' <u>IPTOS_PREC_CRITIC_ECP</u>
CONSTANT	IP precedence 'critic ecp' <u>IPTOS_PREC_FLASHOVERRIDE</u>
CONSTANT	IP precedence 'flash override' <u>IPTOS_PREC_FLASH</u>
CONSTANT	IP precedence 'flash' <u>IPTOS_PREC_IMMEDIATE</u>
CONSTANT	IP precedence 'immediate' <u>IPTOS_PREC_PRIORITY</u>
CONSTANT	IP precedence 'priority' <u>IPTOS_PREC_ROUTINE</u>
CONSTANT	IP precedence 'routing' <u>IPTOS_PREC_INTERNETCONTROL_MIN_DELAY_SET</u>
CONSTANT	IP precedence 'internet control'with the 'minimize delay' bit set <u>IPTOS_PREC_CRITIC_ECP_MIN_DELAY_SET</u>

	IP precedence 'critic ecn' with the 'minimize delay' bit set
CONSTANT	IPOPT_CONTROL
	IP option 'control'
CONSTANT	IPOPT_RESERVED1
	IP option 'reserved1'.
CONSTANT	IPOPT_DEBMEAS
	IP option 'debmeas'
CONSTANT	IPOPT_RESERVED2
	IP option 'reserved2'
CONSTANT	IPOPT_EOL
	IP option 'end of option list'.
CONSTANT	IPOPT_NOP
	IP option 'no operation'.
CONSTANT	IPOPT_RR
	IP option 'record packet route'.
CONSTANT	IPOPT_TS
	IP option 'timestamp'.
CONSTANT	IPOPT_SECURITY
	IP option ' provide s,c,h,tcc'.
CONSTANT	IPOPT_LSRR
	IP option 'loose source route'.
CONSTANT	IPOPT_SATID

	IP option 'satnet id'.
CONSTANT	IPOPT_SSRR
	IP option 'strict source route '.
CONSTANT	IPOPT_TRCRT
	IP option 'Traceroute'.
CONSTANT	IPOPT_OPTVAL
	Offset to IP option 'option ID'
CONSTANT	IPOPT_OLEN
	Offset to IP option 'option length'
CONSTANT	IPOPT_OFFSET
	Offset to IP option 'offset within option'
CONSTANT	IPOPT_MINOFF
	Offset to IP option 'min value of above'
CONSTANT	IPOPT_TS_TSONLY
	Flag bits for ipt_flg (timestamps only);
CONSTANT	IPOPT_TS_TSANDADDR
	Flag bits for ipt_flg (timestamps and addresses);
CONSTANT	IPOPT_TS_PRESPEC
	Flag bits for ipt_flg (specified modules only);
CONSTANT	IPOPT_SECUR_UNCLASS
	'unclass' bits for security in IP option field
CONSTANT	IPOPT_SECUR_CONFID

	'confid' bits for security in IP option field
CONSTANT	IPOPT_SECUR_EFTO
	'efto' bits for security in IP option field
CONSTANT	IPOPT_SECUR_MMMM
	'mmmm' bits for security in IP option field
CONSTANT	IPOPT_SECUR_RESTR
	'restr' bits for security in IP option field
CONSTANT	IPOPT_SECUR_SECRET
	'secreat' bits for security in IP option field
CONSTANT	IPOPT_SECUR_TOPSECRET
	'top secret' bits for security in IP option field
CONSTANT	MAXTTL
	Internet implementation parameters (maximum time to live (seconds))
CONSTANT	IPDEFTTL
	Internet implementation parameters (default ttl, from RFC 1340)
CONSTANT	IPFRAGTTL
	Internet implementation parameters (time to live for frags, slowhz)
CONSTANT	IPTTLDEC
	Internet implementation parameters (subtracted when forwarding)
CONSTANT	IPDEFTOS
	Internet implementation parameters (default TOS)
CONSTANT	IP_MSS
	Internet implementation parameters (default maximum segment size)

CONSTANT	IPPROTO_IP
	IP protocol numbers.
CONSTANT	IPPROTO_ICMP
	IP protocol numbers for ICMP.
CONSTANT	IPPROTO_IGMP
	IP protocol numbers for IGMP.
CONSTANT	IPPROTO_IPIP
	IP protocol numbers for IP tunneling.
CONSTANT	IPPROTO_TCP
	IP protocol numbers for TCP .
CONSTANT	IPPROTO_UDP
	IP protocol numbers for UDP
CONSTANT	IPPROTO_IPV6
	IP protocol number for DUAL-IP.
CONSTANT	IPPROTO_RSVP
	IP protocol numbers for RSVP.
CONSTANT	IPPROTO_MOBILE_IP
	IP protocol numbers for MOBILE_IP.
CONSTANT	IPPROTO_CES_HAIE
	IP protocol numbers.
CONSTANT	IPPROTO_ESP
	IP protocol numbers for IPSEC.
CONSTANT	IPPROTO_AH

	IP protocol numbers for IPSEC.
CONSTANT	IPPROTO_ISAKMP
	IP protocol numbers for IPSEC.
CONSTANT	IPPROTO_CES_ISAKMP
	IP protocol numbers for IPSEC.
CONSTANT	IPPROTO_IAHCP
	IP protocol numbers.
CONSTANT	IPPROTO OSPF
	IP protocol numbers for OSPF .
CONSTANT	IPPROTO_PIM
	IP protocol numbers for PIM .
CONSTANT	IPPROTO_RPIM
	IP protocol numbers for PIM .
CONSTANT	IPPROTO_IGRP
	IP protocol numbers for IGRP .
CONSTANT	IPPROTO_EIGRP
	IP protocol numbers for EIGRP .
CONSTANT	IPPROTO_BELLMANFORD
	IP protocol numbers for BELLMANFORD.
CONSTANT	IPPROTO_IPIP_RED
	IP protocol numbers for IP_RED.
CONSTANT	IPPROTO_FISHEYE

	IP protocol numbers for FISHEYE .
CONSTANT	IPPROTO_FSRL
	IP protocol numbers for LANMAR .
CONSTANT	IPPROTO_ANODR
	IP protocol numbers for ANODR .
CONSTANT	IPPROTO_SECURE_NEIGHBOR
	IP protocol numbers for secure neighbor discovery .
CONSTANT	IPPROTO_SECURE_COMMUNITY
	IP protocol numbers for secure routing community
CONSTANT	IPPROTO_NETWORK_CES_CLUSTER
	IP protocol numbers for clustering protocol.
CONSTANT	IPPROTO_ROUTING_CES_ROSPF
	IP protocol numbers for OSPF protocol.
CONSTANT	IPPROTO_IPIP_ROUTING_CES_MALSR
	IP protocol numbers for MALSR IP encapsulation.
CONSTANT	IPPROTO_IPIP_ROUTING_CES_ROSPF
	IP protocol numbers for OSPF IP encapsulation.
CONSTANT	IPPROTO_NETWORK_CES_REGION
	IP protocol numbers for RAP election protocol.
CONSTANT	IPPROTO_MPR
	IP protocol numbers for MPR
CONSTANT	IPPROTO_IPIP_ROUTING_CES_SRW

	IP protocol numbers for ROUTING_CES_SRW IP encapsulation. IPPROTO_IPIP_SDR
CONSTANT	IP protocol numbers for SDR IP encapsulation. IPPROTO_IPIP_SDR
CONSTANT	IP protocol numbers for SDR IP encapsulation. IPPROTO_MULTICAST_CES_SRW_MOSPF
CONSTANT	IP protocol numbers for MULTICAST_CES_SRW_MOSPF IP encapsulation. IPPROTO_CES_HSLS
CONSTANT	IP protocol numbers for HSLS protocol. IPPROTO_AODV
CONSTANT	IP protocol numbers for AODV . IPPROTO_DYMO
CONSTANT	IP protocol numbers for DYMO . IPPROTO_MAODV
CONSTANT	IP protocol numbers for MAODV. IPPROTO_DSR
CONSTANT	IP protocol numbers for DSR . IPPROTO_ODMRP
CONSTANT	IP protocol numbers for ODMRP . IPPROTO_LAR1
CONSTANT	IP protocol numbers for LAR1. IPPROTO_STAR

	IP protocol numbers for STAR.
CONSTANT	IPPROTO_DAWN
	IP protocol numbers for DAWN.
CONSTANT	IPPROTO_EPLRS
	IP protocol numbers for EPLRS protocol.
CONSTANT	IPPROTO_EPLRS
	IP protocol numbers for EPLRS protocol.
CONSTANT	IPPROTO_CES_EPLRS_MPR
	IP protocol numbers for EPLRS MPR protocol.
CONSTANT	IPPROTO_DVMRP
	IP protocol numbers for DVMRP.
CONSTANT	IPPROTO_GSM
	IP protocol numbers for GSM.
CONSTANT	IPPROTO_EXTERNAL
	IP protocol for external interface.
CONSTANT	IPPROTO_INTERNET_GATEWAY
	IP protocol numbers for Internet gateway for emulated nodes
CONSTANT	IPPROTO_EXATA_VIRTUAL_LAN
	IP protocol numbers for Internet gateway for emulated nodes
CONSTANT	IPPROTO_NDP
	IP protocol numbers for NDP.
CONSTANT	IPPROTO_BRP
	IP protocol numbers for BRP .

CONSTANT	<u>IP_MIN_HEADER_SIZE</u>	Minimum IP header size in bytes
CONSTANT	<u>IP_MAX_HEADER_SIZE</u>	Maximum IP header size in bytes
CONSTANT	<u>IP_FRAGMENT_HOLD_TIME</u>	Fragmented packets hold time.
CONSTANT	<u>IP_MIN_MULTICAST_ADDRESS</u>	Used to determine whether an IP address is multicast.
CONSTANT	<u>IP_MAX_MULTICAST_ADDRESS</u>	Used to determine whether an IP address is multicast.
CONSTANT	<u>MULTICAST_DEFAULT_INTERFACE_ADDRESS</u>	Default multicast interface address (224.0.0.0).
CONSTANT	<u>IP_MIN_RESERVED_MULTICAST_ADDRESS</u>	Minimum reserve multicast address (224.0.0.0).
CONSTANT	<u>IP_MAX_RESERVED_MULTICAST_ADDRESS</u>	Maximum reserve multicast address (224.0.0.255).
CONSTANT	<u>MULTICAST_DEFAULT_NUM_HOST_BITS</u>	Multicast default num host bit
CONSTANT	<u>NETWORK_UNREACHABLE</u>	Network unreachable.
CONSTANT	<u>DEFAULT_INTERFACE</u>	Default interface index
CONSTANT	<u>NETWORK_IP_REASS_BUFF_TIMER</u>	

	Max time data can stored in assembly buffer
CONSTANT	MAX_IP_FRAGMENTS_SIMPLE_CASE
	Max size of fragment allowed.
CONSTANT	SMALL_REASSEMBLY_BUFFER_SIZE
	Size of reassemble buffer
CONSTANT	REASSEMBLY_BUFFER_EXPANSION_MULTIPLIER
	Multiplier used for reassemble buffer expansion
ENUMERATION	BackplaneType
	NetworkIp backplane type(either CENTRAL or DISTRIBUTED)
STRUCT	IpHeaderType
	IpHeaderType is 20 bytes,just like in the BSD code.
STRUCT	ip_timestamp
	Time stamp option structure.
STRUCT	ip_traceroute
	TraceRoute option structure.
STRUCT	NetworkIpBackplaneInfo
	Structure maintaining IP Back plane Information
STRUCT	ipHeaderSizeInfo
	Structure maintaining IP header size Information
STRUCT	NetworkMulticastForwardingTableRow
	Structure of an entity of multicast forwarding table.
STRUCT	NetworkMulticastForwardingTable

	Structure of multicast forwarding table NetworkIpMulticastGroupEntry
STRUCT	Structure for Multicast Group Entry IpPerHopBehaviorInfoType
STRUCT	Structure to maintain DS priority queue mapping IpMftcParameter
STRUCT	Variables of the structure define a unique condition class IpMultiFieldTrafficConditionerInfo
STRUCT	Structure used to store traffic condition. IpOptionsHeaderType
STRUCT	Structure of optional header for IP source route NetworkIpStatsType
STRUCT	Structure used to keep track of all stats of network layer. NetworkForwardingTableRow
STRUCT	Structure of an entity of forwarding table. NetworkForwardingTable
STRUCT	Structure of forwarding table. IpInterfaceInfoType
STRUCT	Structure for maintaining IP interface informations. This struct must be allocated by new, not MEM_malloc. All member variables MUST be initialized in the constructor. ip_frag_data
STRUCT	QualNet typedefs struct ip_frag_data to IpFragData. is a simple queue to hold fragmented packets. Ipv6FragQueue

	Ipv6 fragment queue structure. FragmetedMsg
STRUCT	QualNet typedefs struct fragmeted_msg_struct to ip6q. struct fragmeted_msg_struct is a simple fragmented packets msg hold structure.
STRUCT	NetworkDataIp;
STRUCT	Main structure of network layer. IpReassemblyBufferType
STRUCT	Structure of reassembly buffer IpReassemblyBufferListCellType
STRUCT	Structure of reassembly buffer cell listing IpReassemblyBufferListType
STRUCT	Structure of reassembly buffer list

Function / Macro Summary

Return Type	Summary
MACRO	IPOPT_COPIED(o) IP option 'copied'.
MACRO	IPOPT_CLASS(o) IP option 'class'
MACRO	IPOPT_NUMBER(o) IP option 'number'
MACRO	IpHeaderSize(ipHeader) Returns IP header ip_hl field * 4, which is the size of the IP header in bytes.
MACRO	SetIpHeaderSize(IpHeader, Size)

	Sets IP header ip_hl field (header length) to Size divided by 4 FragmentOffset(ipHeader)
MACRO	Starting position of this fragment in actual packet. SetFragmentOffset(ipHeader, offset)
MACRO	To set offset of fragment.
void	IpHeaderSetVersion() (UInt32* ip_v_hl_tos_len, unsigned int version) Set the value of version number for IpHeaderType
void	IpHeaderSetHLen() (UInt32* ip_v_hl_tos_len, unsigned int hlen) Set the value of header length for IpHeaderType
void	IpHeaderSetTOS() (UInt32* ip_v_hl_tos_len, unsigned int ipTos) Set the value of Type of Service for IpHeaderType
void	IpHeaderSetIpLength() (UInt32* ip_v_hl_tos_len, unsigned int ipLen) Set the value of ip length for IpHeaderType
void	IpHeaderSetIpFragOffset() (UInt16* ipFragment, UInt16 offset) Set the value of ip_fragment_offset for IpHeaderType
void	IpHeaderSetIpReserved() (UInt16* ipFragment, UInt16 ipReserved) Set the value of ipReserved for IpHeaderType
void	IpHeaderSetIpDontFrag() (UInt16* ipFragment, UInt16 dontFrag) Set the value of ip_dont_fragment for IpHeaderType
void	IpHeaderSetIpMoreFrag() (UInt16* ipFragment, UInt16 moreFrag) Set the value of ip_more_fragment for IpHeaderType
unsigned int	IpHeaderGetVersion() (UInt32 ip_v_hl_tos_len)

	Returns the value of version number for IpHeaderType IpHeaderGetHLen() (UInt32 ip_v_hl_tos_len)
unsigned int	Returns the value of header length for IpHeaderType IpHeaderGetTOS() (UInt32 ip_v_hl_tos_len)
unsigned int	Returns the value of Type of Service for IpHeaderType IpHeaderGetTpLength() (UInt32 ip_v_hl_tos_len)
unsigned int	Returns the value of ip length for IpHeaderType IpHeaderGetIpFragOffset() (UInt16 ipFragment)
UInt16	Returns the value of ip_fragment_offset for IpHeaderType IpHeaderGetIpDontFrag() (UInt16 ipFragment)
BOOL	Returns the value of ip_dont_fragment for IpHeaderType IpHeaderGetIpMoreFrag() (UInt16 ipFragment)
BOOL	Returns the value of ip_more_fragment for IpHeaderType IpHeaderGetIpReserved() (UInt16 ipFragment)
void	Returns the value of ipReserved for IpHeaderType Ip_timestampSetFlag() (unsigned char flgOflw, unsigned char flag) Set the value of flag for ip_timestamp_str Ip_timestampSetOvflw() (unsigned char flgOflw, unsigned char ovflw)
void	Set the value of ovflw for ip_timestamp_str Ip_timestampGetFlag() (unsigned char flgOflw)
unsigned char	Returns the value of flag for ip_timestamp_str Ip_timestampGetOvflw() (unsigned char flgOflw)
unsigned char	

	Returns the value of overflow counter for ip_timestamp_str ConvertNumHostBitsToSubnetMask (int numHostBits)
NodeAddress	To generate subnetmask using number of host bit ConvertSubnetMaskToNumHostBits (NodeAddress subnetMask)
int	To generate number of host bit using subnetmask. MaskIpAddress (NodeAddress address, NodeAddress mask)
NodeAddress	To mask a ip address. MaskIpAddressWithNumHostBits (NodeAddress address, int numHostBits)
NodeAddress	To mask a ip address. CalcBroadcastIpAddress (NodeAddress address, int numHostBits)
BOOL	To generate broadcast address. IsIpAddressInSubnet (NodeAddress address, NodeAddress subnetAddress, int numHostbits)
void	To check if a ip address belongs to a subnet. NetworkIpAddHeader (Node* node, Message* msg, NodeAddress sourceAddress, NodeAddress destinationAddress, TosType priority, unsigned char protocol, unsigned ttl)
IpOptionsHeaderType*	Add an IP packet header to a message. Just calls AddIpHeader. FindAnIpOptionField (const IpHeaderType* ipHeader, const int optionKey)
	Searches the IP header for the option field with option code that matches optionKey, and returns a pointer to the option field header.
void	NetworkIpPreInit (Node* node) IP initialization required before any of the other layers are initialized. This is mainly for MAC initialization, which requires certain IP structures be pre-initialized.
void	NetworkIpInit (Node* node, const NodeInput* nodeInput) Initialize IP variables, and all network-layer IP protocols..
void	NetworkIpLayer (Node* node, Message* msg)

	Handle IP layer events, incoming messages and messages sent to itself (timers, etc.).
void	<p><code>NetworkIpFinalize</code>(Node* node)</p> <p>Finalize function for the IP model. Finalize functions for all network-layer IP protocols are called here.</p>
void	<p><code>NetworkIpReceivePacketFromTransportLayer</code>(Node* node, Message* msg, NodeAddress sourceAddress, NodeAddress destinationAddress, int outgoingInterface, TosType priority, unsigned char protocol, BOOL isEcnCapable)</p> <p>Called by transport layer protocols (UDP, TCP) to send UDP datagrams and TCP segments using IP. Simply calls <code>NetworkIpSendRawMessage()</code>.</p>
void	<p><code>NetworkIpSendRawMessage</code>(Node* node, Message* msg, NodeAddress sourceAddress, NodeAddress destinationAddress, int outgoingInterface, TosType priority, unsigned char protocol, unsigned ttl)</p> <p>Called by <code>NetworkIpReceivePacketFromTransportLayer()</code> to send UDP datagrams, TCP segments using IP. Also called by network-layer routing protocols (AODV, OSPF, etc.) to send IP packets. This function adds an IP header and calls <code>RoutePacketAndSendToMac()</code>.</p>
void	<p><code>NetworkIpSendRawMessageWithDelay</code>(Node* node, Message* msg, NodeAddress sourceAddress, NodeAddress destinationAddress, int outgoingInterface, TosType priority, unsigned char protocol, unsigned ttl, clocktype delay)</p> <p>Same as <code>NetworkIpSendRawMessage()</code>, but schedules event after a simulation delay.</p>
void	<p><code>NetworkIpSendRawMessageToMacLayer</code>(Node* node, Message* msg, NodeAddress sourceAddress, NodeAddress destinationAddress, TosType priority, unsigned char protocol, unsigned ttl, int interfaceIndex, NodeAddress nextHop)</p> <p>Called by network-layer routing protocols (AODV, OSPF, etc.) to add an IP header to payload data, and with the resulting IP packet, calls <code>NetworkIpSendPacketOnInterface()</code>.</p>
void	<p><code>NetworkIpSendRawMessageToMacLayerWithDelay</code>(Node* node, Message* msg, NodeAddress sourceAddress, NodeAddress destinationAddress, TosType priority, unsigned char protocol, unsigned ttl, int interfaceIndex, NodeAddress nextHop, clocktype delay)</p> <p>Same as <code>NetworkIpSendRawMessageToMacLayer()</code>, but schedules the event after a simulation delay by calling <code>NetworkIpSendPacketOnInterfaceWithDelay()</code>.</p>
void	<p><code>NetworkIpSendPacketToMacLayer</code>(Node* node, Message* msg, int interfaceIndex, NodeAddress nextHop)</p> <p>This function is called once the outgoing interface index and next hop address to which to route an IP packet are known.</p>
void	<p><code>NetworkIpSendPacketOnInterface</code>(Node* node, Message* msg, int incomingInterface, int outgoingInterface, NodeAddress nextHop)</p> <p>This function is called once the outgoing interface index and next hop address to which to route an IP packet are known. This</p>

	queues an IP packet for delivery to the MAC layer. This function calls QueueUpIpFragmentForMacLayer(). This function is used to initiate fragmentation if required, but since fragmentation has been disabled, all it does is assert false if the IP packet is too big before calling the next function.
void	<p><code>NetworkIpSendPacketToMacLayerWithDelay</code>(Node* node, Message* msg, int interfaceIndex, NodeAddress nextHop, clocktype delay)</p> <p>Same as NetworkIpSendPacketOnInterface(), but schedules event after a simulation delay.</p>
void	<p><code>NetworkIpSendPacketOnInterfaceWithDelay</code>(Node* node, Message* msg, int incomingInterface, int outgoingInterface, NodeAddress nextHop, clocktype delay)</p> <p>Same as NetworkIpSendPacketOnInterface(), but schedules event after a simulation delay.</p>
void	<p><code>NetworkIpSendRawPacketOnInterfaceWithDelay</code>(Node* node, Message* msg, int incomingInterface, int outgoingInterface, NodeAddress nextHop, clocktype delay)</p> <p>Same as NetworkIpSendPacketOnInterface(), but schedules event after a simulation delay and denotes raw packet.</p>
void	<p><code>NetworkIpSendPacketToMacLayerWithNewStrictSourceRoute</code>(Node* node, Message* msg, NodeAddress[] newRouteAddresses, int numNewRouteAddresses, BOOL removeExistingRecordedRoute)</p> <p>Tacks on a new source route to an existing IP packet and then sends the packet to the MAC layer.</p>
void	<p><code>NetworkIpReceivePacketFromMacLayer</code>(Node* node, Message* msg, NodeAddress previousHopNodeId, int interfaceIndex)</p> <p>IP received IP packet from MAC layer. Updates the Stats database and then calls NetworkIpReceivePacket.</p>
void	<p><code>NetworkIpReceivePacket</code>(Node* node, Message* msg, NodeAddress previousHopNodeId, int interfaceIndex)</p> <p>IP received IP packet. Determine whether the packet is to be delivered to this node, or needs to be forwarded. ipHeader->ip_ttl is decremented here, instead of the way BSD TCP/IP does it, which is to decrement TTL right before forwarding the packet. QualNet's alternative method suits its network-layer ad hoc routing protocols, which may do their own forwarding.</p>
void	<p><code>NetworkIpNotificationOfPacketDrop</code>(Node* node, Message* msg, NodeAddress nextHopNodeAddress, int interfaceIndex)</p> <p>Invoke callback functions when a packet is dropped.</p>
MacLayerStatusEventHandlerFunctionType	<p><code>NetworkIpGetMacLayerStatusEventHandlerFunction</code>(Node* node, int interfaceIndex)</p> <p>Get the status event handler function pointer.</p>
void	<p><code>NetworkIpSetMacLayerStatusEventHandlerFunction</code>(Node* node, MacLayerStatusEventHandlerFunctionType StatusEventHandlerPtr, int interfaceIndex)</p> <p>Allows the MAC layer to send status messages (e.g., packet drop, link failure) to a network-layer routing protocol for routing</p>

	optimization.
void	<p>NetworkIpSneakPeekAtMacPacket(Node* node, const Message* msg, int interfaceIndex, NodeAddress prevHop)</p> <p>Called Directly by the MAC layer, this allows a routing protocol to "sneak a peek" or "tap" messages it would not normally see from the MAC layer. This function will possibly unfragment such packets and call the function registered by the routing protocol to do the "Peek".</p>
PromiscuousMessagePeekFunctionType	<p>NetworkIpGetPromiscuousMessagePeekFunction(Node* node, int interfaceIndex)</p> <p>Returns the network-layer function which will promiscuously inspect packets. See NetworkIpSneakPeekAtMacPacket().</p>
void	<p>NetworkIpSetPromiscuousMessagePeekFunction(Node* node, PromiscuousMessagePeekFunctionType PeekFunctionPtr, int interfaceIndex)</p> <p>Sets the network-layer function which will promiscuously inspect packets. See NetworkIpSneakPeekAtMacPacket().</p>
void	<p>NetworkIpReceiveMacAck(Node* node, int interfaceIndex, const Message* msg, NodeAddress nextHop)</p> <p>MAC received an ACK, so call ACK handler function.</p>
MacLayerAckHandlerType	<p>NetworkIpGetMacLayerAckHandler(Node* node, int interfaceIndex)</p> <p>Get MAC layer ACK handler</p>
void	<p>NetworkIpSetMacLayerAckHandler(Node* node, MacLayerAckHandlerType macAckHandlerPtr, int interfaceIndex)</p> <p>Set MAC layer ACK handler</p>
void	<p>SendToUdp(Node* node, Message* msg, TosType priority, NodeAddress sourceAddress, NodeAddress destinationAddress, int incomingInterfaceIndex)</p> <p>Sends a UDP packet to UDP in the transport layer. The source IP address, destination IP address, and priority of the packet are also sent.</p>
void	<p>SendToTcp(Node* node, Message* msg, TosType priority, NodeAddress sourceAddress, NodeAddress destinationAddress, BOOL aCongestionExperienced)</p> <p>Sends a TCP packet to TCP in the transport layer. The source IP address, destination IP address, and priority of the packet are also sent..</p>
void	<p>SendToRsvp(Node* node, Message* msg, TosType priority, NodeAddress sourceAddress, NodeAddress destinationAddress, int interfaceIndex, unsigned ttl)</p> <p>Sends a RSVP packet to RSVP in the transport layer. The source IP address, destination IP address, and priority of the packet are also sent.</p>
void	<p>NetworkIpRemoveIpHeader(Node* node, Message* msg, NodeAddress* sourceAddress, NodeAddress* destinationAddress, TosType* priority, unsigned char* protocol, unsigned* ttl)</p>

	Removes the IP header from a message while also returning all the fields of the header.
void	<code>AddIpOptionField</code> (Node* node, Message* msg, int optionCode, int optionSize)
void	Inserts an option field in the header of an IP packet.
RouterFunctionType	Retrieves a copy of the source and recorded route from the options field in the header.
void	<code>NetworkIpGetRouterFunction</code> (Node* node, int interfaceIndex)
	Get the router function pointer.
void	<code>NetworkIpSetRouterFunction</code> (Node* node, RouterFunctionType RouterFunctionPtr, int interfaceIndex)
	Allows a routing protocol to set the "routing function" (one of its functions) which is called when a packet needs to be routed. NetworkIpSetRouterFunction() allows a routing protocol to define the routing function. The routing function is called by the network layer to ask the routing protocol to route the packet. The routing function is given the packet and its destination. The routing protocol can route the packet and set "PacketWasRouted" to TRUE; or not route the packet and set to FALSE. If the packet, was not routed, then the network layer will try to use the forwarding table or the source route in the IP header. This function will also be given packets for the local node the routing protocols can look at packets for protocol reasons. In this case, the message should not be modified and PacketWasRouted must be set to FALSE.
void	<code>NetworkIpAddUnicastRoutingProtocolType</code> (Node* node, NetworkRoutingProtocolType routingProtocolType, int interfaceIndex)
	Add unicast routing protocol type to interface.
void	<code>NetworkIpAddUnicastIntraRegionRoutingProtocolType</code> (Node* node, NetworkRoutingProtocolType routingProtocolType, int interfaceIndex)
	Add unicast intra region routing protocol type to interface.
void*	<code>NetworkIpGetRoutingProtocol</code> (Node* node, NetworkRoutingProtocolType routingProtocolType)
	Get routing protocol structure associated with routing protocol running on this interface.
NetworkRoutingProtocolType	<code>NetworkIpGetUnicastRoutingProtocolType</code> (Node* node, int interfaceIndex)
	Get unicast routing protocol type on this interface.
void	<code>NetworkIpSetHsrpOnInterface</code> (Node* node, int interfaceIndex)

	To enable hsrp on a interface NetworkIpIsHsrpEnabled (Node* node, int interfaceIndex)
BOOL	To test if any interface is hsrp enabled.
void	NetworkIpAddNewInterface (Node* node, NodeAddress interfaceIpAddress, int numHostBits, int* newInterfaceIndex, const NodeInput* nodeInput) Add new interface to node.
void	NetworkIpInitCpuQueueConfiguration (Node* node, const NodeInput* nodeInput) Initializes cpu queue parameters during startup.
void	NetworkIpInitInputQueueConfiguration (Node* node, const NodeInput* nodeInput, int interfaceIndex) Initializes input queue parameters during startup.
void	NetworkIpInitOutputQueueConfiguration (Node* node, const NodeInput* nodeInput, int interfaceIndex) Initializes queue parameters during startup.
void	NetworkIpCreateQueues (Node* node, const NodeInput* nodeInput, int interfaceIndex) Initializes input and output queue parameters during startup
void	NetworkIpSchedulerParameterInit (Scheduler* schedulerPtr, const int numPriorities, Queue* queue) Initialize the scheduler parameters and also allocate memory for queues if require.
void	NetworkIpSchedulerInit (Node* node, const NodeInput* nodeInput, int interfaceIndex, Scheduler* schedulerPtr, const char* schedulerTypeString) Call initialization function for appropriate scheduler.
void	NetworkIpCpuQueueInsert (Node* node, Message* msg, NodeAddress nextHopAddress, NodeAddress destinationAddress, int outgoingInterface, int networkType, BOOL* queueIsFull, int incomingInterface) Calls the cpu packet scheduler for an interface to retrieve an IP packet from a queue associated with the interface. The dequeued packet, since it's already been routed, has an associated next-hop IP address. The packet's priority value is also returned.
void	NetworkIpInputQueueInsert (Node* node, int incomingInterface, Message* msg, NodeAddress nextHopAddress, NodeAddress destinationAddress, int outgoingInterface, int networkType, BOOL* queueIsFull) Calls input packet scheduler for an interface to retrieve an IP packet from a queue associated with the interface. The dequeued

	packet, since it's already been routed, has an associated next-hop IP address. The packet's priority value is also returned.
void	<p>NetworkIpOutputQueueInsert(Node* node, int interfaceIndex, Message* msg, NodeAddress nextHopAddress, NodeAddress destinationAddress, int networkType, BOOL* queueIsFull)</p> <p>Calls output packet scheduler for an interface to retrieve an IP packet from a queue associated with the interface. The dequeued packet, since it's already been routed, has an associated next-hop IP address. The packet's priority value is also returned. Called by QueueUpIpFragmentForMacLayer().</p>
BOOL	<p>NetworkIpInputQueueDequeuePacket(Node* node, int incomingInterface, Message** msg, NodeAddress* nextHopAddress, int* outgoingInterface, int* networkType, QueuePriorityType* priority)</p> <p>Calls the packet scheduler for an interface to retrieve an IP packet from the input queue associated with the interface.</p>
BOOL	<p>NetworkIpOutputQueueDequeuePacket(Node* node, int interfaceIndex, Message** msg, NodeAddress* nextHopAddress, int* networkType, QueuePriorityType* priority)</p> <p>Calls the packet scheduler for an interface to retrieve an IP packet from a queue associated with the interface. The dequeued packet, since it's already been routed, has an associated next-hop IP address. The packet's priority value is also returned. This function is called by MAC_OutputQueueDequeuePacket() (mac/mac.pc), which itself is called from mac/mac_802_11.pc and other MAC protocol source files. This function will assert false if the scheduler cannot return an IP packet for whatever reason.</p>
BOOL	<p>NetworkIpOutputQueueDequeuePacketForAPriority(Node* node, int interfaceIndex, QueuePriorityType priority, Message** msg, NodeAddress* nextHopAddress, int* networkType, int posInQueue)</p> <p>Same as NetworkIpOutputQueueDequeuePacket(), except the packet dequeued is requested by a specific priority, instead of leaving the priority decision up to the packet scheduler. This function is called by MAC_OutputQueueDequeuePacketForAPriority() (mac/mac.pc), which itself is called from mac/mac_802_11.pc and other MAC protocol source files. This function will assert false if the scheduler cannot return an IP packet for whatever reason.</p>
BOOL	<p>NetworkIpOutputQueueDequeuePacketWithIndex(Node* node, int interfaceIndex, int msgIndex, Message** msg, NodeAddress* nextHopAddress, int* networkType)</p> <p>Same as NetworkIpOutputQueueDequeuePacket(), except the packet dequeued is requested by a specific index. This function is called by MAC_OutputQueueDequeuePacketForAPriority() (mac/mac.pc), which itself is called from mac/mac_802_11.pc and other MAC protocol source files. This function will assert false if the scheduler cannot return an IP packet for whatever reason.</p>
BOOL	<p>NetworkIpInputQueueTopPacket(Node* node, int incomingInterface, Message** msg, NodeAddress* nextHopAddress, int* outgoingInterface, int* networkType, QueuePriorityType* priority)</p> <p>Same as NetworkIpInputQueueDequeuePacket(), except the packet is not actually dequeued. Note that the message containing the packet is not copied; the contents may (inadvertently or not) be directly modified.</p>
BOOL	<p>NetworkIpOutputQueueTopPacket(Node* node, int interfaceIndex, Message** msg, NodeAddress* nextHopAddress, int* networkType, QueuePriorityType* priority)</p> <p>Same as NetworkIpOutputQueueDequeuePacket(), except the packet is not actually dequeued. Note that the message containing the packet is not copied; the contents may (inadvertently or not) be directly modified. This function is called by MAC_OutputQueueTopPacket() (mac/mac.pc), which itself is called from mac/mac_802_11.pc and other MAC protocol source files. This function will assert false if the scheduler cannot return an IP packet for whatever reason.</p>

BOOL	<code>NetworkIpOutputQueuePeekWithIndex</code> (Node* node, int interfaceIndex, int msgIndex, Message** msg, NodeAddress* nextHopAddress, QueuePriorityType* priority)	Same as NetworkIpOutputQueueDequeuePacket(), except the packet is not actually dequeued. Note that the message containing the packet is not copied; the contents may (inadvertently or not) be directly modified. This function is called by MAC_OutputQueueTopPacket() (mac/mac.pc), which itself is called from mac/mac_802_11.pc and other MAC protocol source files. This function will assert false if the scheduler cannot return an IP packet for whatever reason.
BOOL	<code>NetworkIpOutputQueueTopPacketForAPriority</code> (Node* node, int interfaceIndex, QueuePriorityType priority, Message** msg, NodeAddress* nextHopAddress, int posInQueue)	Same as NetworkIpOutputQueueDequeuePacketForAPriority(), except the packet is not actually dequeued. Note that the message containing the packet is not copied; the contents may (inadvertently or not) be directly modified. This function is called by MAC_OutputQueueTopPacketForAPriority() (mac/mac.pc), which itself is called from mac/mac_802_11.pc and other MAC protocol source files. This function will assert false if the scheduler cannot return an IP packet for whatever reason.
BOOL	<code>NetworkIpInputQueueIsEmpty</code> (Node* node, int incomingInterface)	Calls the packet scheduler for an interface to determine whether the interface's input queue is empty
BOOL	<code>NetworkIpOutputQueueIsEmpty</code> (Node* node, int interfaceIndex)	Calls the packet scheduler for an interface to determine whether the interface's output queue is empty.
int	<code>NetworkIpOutputQueueNumberInQueue</code> (Node* node, int interfaceIndex, BOOL specificPriorityOnly, QueuePriorityType priority)	Calls the packet scheduler for an interface to determine how many packets are in a queue. There may be multiple queues on an interface, so the priority of the desired queue is also provided.
NodeAddress	<code>NetworkIpOutputQueueDropPacket</code> (Node* node, int interfaceIndex, Message** msg)	Drop a packet from the queue.
void	<code>NetworkIpDeleteOutboundPacketsToANode</code> (Node* node, const NodeAddress nextHopAddress, const NodeAddress destinationAddress, const BOOL returnPacketsToRoutingProtocol)	Deletes all packets in the queue going (probably broken), to the specified next hop address. There is option to return all such packets back to the routing protocols via the usual mechanism (callback).
unsigned	<code>GetQueueNumberFromPriority</code> (TosType userTos, int numQueues)	Get queue number through which a given user priority will be forwarded.
QueuePriorityType	<code>ReturnPriorityForPHB</code> (Node* node, TosType tos)	Returns the priority queue corresponding to the DS/TOS field.

void	<code>NetworkGetInterfaceAndNextHopFromForwardingTable</code> (Node* node, NodeAddress destinationAddress, int* interfaceIndex, NodeAddress* nextHopAddress)	Do a lookup on the routing table with a destination IP address to obtain a route (index of an outgoing interface and a next hop Ip address).
void	<code>NetworkGetInterfaceAndNextHopFromForwardingTable</code> (Node* node, int currentInterface, NodeAddress destinationAddress, int* interfaceIndex, NodeAddress* nextHopAddress)	Do a lookup on the routing table with a destination IP address to obtain a route (index of an outgoing interface and a next hop Ip address).
void	<code>NetworkGetInterfaceAndNextHopFromForwardingTable</code> (Node* node, NodeAddress destinationAddress, int* interfaceIndex, NodeAddress* nextHopAddress, BOOL testType, NetworkRoutingProtocolType type)	Do a lookup on the routing table with a destination IP address to obtain a route (index of an outgoing interface and a next hop Ip address).
void	<code>NetworkGetInterfaceAndNextHopFromForwardingTable</code> (Node* node, int operatingInterface, NodeAddress destinationAddress, int* interfaceIndex, NodeAddress* nextHopAddress, BOOL testType, NetworkRoutingProtocolType type)	Do a lookup on the routing table with a destination IP address to obtain a route (index of an outgoing interface and a next hop Ip address).
int	<code>NetworkIpGetInterfaceIndexForNextHop</code> (Node* node, NodeAddress nextHopAddress)	This function looks at the network address of each of a node's network interfaces. When nextHopAddress is matched to a network, the interface index corresponding to the network is returned. (used by NetworkUpdateForwardingTable() and ospfv2.pc)
int	<code>NetworkGetInterfaceIndexForDestAddress</code> (Node* node, NodeAddress destAddress)	Get interface for the destination address.
NetworkRoutingAdminDistanceType	<code>NetworkRoutingGetAdminDistance</code> (Node* node, NetworkRoutingProtocolType type)	Get the administrative distance of a routing protocol. These values don't quite match those recommended by Cisco.
void	<code>NetworkInitForwardingTable</code> (Node* node)	Initialize the IP fowarding table, allocate enough memory for number of rows.
void	<code>NetworkUpdateForwardingTable</code> (Node* node, NodeAddress destAddress, NodeAddress destAddressMask, NodeAddress nextHopAddress, int outgoingInterfaceIndex, int cost, NetworkRoutingProtocolType type)	Update or add entry to IP routing table. Search the routing table for an entry with an exact match for destAddress, destAddressMask, and routing protocol. Update this entry with the specified nextHopAddress (the outgoing interface is

	automatically determined from the nextHopAddress -- see code). If no matching entry found, then add a new route.
void	NetworkRemoveForwardingTableEntry (Node* node, NodeAddress destAddress, NodeAddress destAddressMask, NodeAddress nextHopAddress, int outgoingInterfaceIndex)
	Remove single entries in the routing table
void	NetworkEmptyForwardingTable (Node* node, NetworkRoutingProtocolType type)
	Remove entries in the routing table corresponding to a given routing protocol.
void	NetworkPrintForwardingTable (Node* node)
	Display all entries in node's routing table.
int	NetworkGetMetricForDestAddress (Node* node, NodeAddress destAddress)
	Get the cost metric for a destination from the forwarding table.
void	NetworkIpSetRouteUpdateEventFunction (Node* node, NetworkRouteUpdateEventType routeUpdateFunctionPtr)
	Set a callback function when a route changes from forwarding table.
NetworkRouteUpdateEventType	NetworkIpGetRouteUpdateEventFunction (Node* node)
	Print packet headers when packet tracing is enabled.
NodeAddress	NetworkIpGetInterfaceAddress (Node* node, int interfaceIndex)
	Get the interface address on this interface
char*	NetworkIpGetInterfaceName (Node* node, int interfaceIndex)
	To get the interface name associated with the interface
NodeAddress	NetworkIpGetInterfaceNetworkAddress (Node* node, int interfaceIndex)
	To get network address associated with interface
NodeAddress	NetworkIpGetInterfaceSubnetMask (Node* node, int interfaceIndex)
	To retrieve subnet mask of the node interface
int	NetworkIpGetInterfaceNumHostBits (Node* node, int interfaceIndex)

	Get the number of host bits on this interface
NodeAddress	NetworkIpGetInterfaceBroadcastAddress (Node* node, int interfaceIndex)
	Get broadcast address on this interface
BOOL	IsOutgoingBroadcast (Node* node, NodeAddress destAddress, int* outgoingInterface, NodeAddress* outgoingBroadcastAddress)
	Checks whether IP packet's destination address is broadcast
BOOL	NetworkIpIsMyIP (Node* node, NodeAddress ipAddress)
	In turn calls IsMyPacket()
void	NetworkIpConfigurationError (Node* node, char [] parameterName, int interfaceIndex)
	Prints out the IP configuration error
void	NetworkPrintIpHeader (Message* msg)
	To print the IP header
void	NetworkIpAddToMulticastGroupList (Node* node, NodeAddress groupAddress)
	Add a specified node to a multicast group
void	NetworkIpRemoveFromMulticastGroupList (Node* node, NodeAddress groupAddress)
	To remove specified node from a multicast group
void	NetworkIpPrintMulticastGroupList (Node* node)
	To print the multicast group list
BOOL	NetworkIpIsPartOfMulticastGroup (Node* node, NodeAddress groupAddress)
	check if a node is part of specified multicast group
void	NetworkIpJoinMulticastGroup (Node* node, NodeAddress mcastAddr, clocktype delay)
	To join a multicast group
void	NetworkIpJoinMulticastGroup (Node* node, NodeAddress mcastAddr, clocktype delay)

	To join a multicast group <code>NetworkIpLeaveMulticastGroup</code> (Node* node, NodeAddress mcastAddr, clocktype delay)
void	To leave a multicast group <code>NetworkIpLeaveMulticastGroup</code> (Node* node, NodeAddress mcastAddr, clocktype delay)
void	To leave a multicast group <code>NetworkIpSetMulticastTimer</code> (Node* node, long eventType, NodeAddress mcastAddr, clocktype delay)
void	To set a multicast timer to join or leave multicast groups <code>NetworkIpSetMulticastRoutingProtocol</code> (Node* node, void* multicastRoutingProtocol, int interfaceIndex)
void *	Assign a multicast routing protocol to an interface <code>NetworkIpGetMulticastRoutingProtocol</code> (Node* node, NetworkRoutingProtocolType routingProtocolType)
void	To get the Multicast Routing Protocol structure <code>NetworkIpAddMulticastRoutingProtocolType</code> (Node* node, NetworkRoutingProtocolType multicastProtocolType, int interfaceIndex)
void	Assign a multicast protocol type to an interface <code>NetworkIpSetMulticastRouterFunction</code> (Node* node, MulticastRouterFunctionType routerFunctionPtr, int interfaceIndex)
MulticastRouterFunctionType	Set a multicast router function to an interface <code>NetworkIpGetMulticastRouterFunction</code> (Node* node, int interfaceIndex)
void	Get the multicast router function for an interface <code>NetworkIpUpdateMulticastRoutingProtocolAndRouterFunction</code> (Node* node, NetworkRoutingProtocolType routingProtocolType, int interfaceIndex)
void	Assign multicast routing protocol structure and router function to an interface. We are only allocating the multicast routing protocol structure and router function once by using pointers to the original structures. <code>NetworkIpUpdateUnicastRoutingProtocolAndRouterFunction</code> (Node* node, NetworkRoutingProtocolType routingProtocolType, int interfaceIndex)
	Assign unicast routing protocol structure and router function to an interface. We are only allocating the unicast routing protocol structure and router function once by using pointers to the original structures.

int	NetworkIpGetInterfaceIndexFromAddress (Node* node, NodeAddress address) Get the interface index from an IP address.
int	NetworkIpGetInterfaceIndexFromSubnetAddress (Node* node, NodeAddress address) Get the interface index from an IP subnet address.
BOOL	NetworkIpIsMulticastAddress (Node* node, NodeAddress address) Check if an address is a multicast address.
void	NetworkInitMulticastForwardingTable (Node* node) initialize the multicast fowarding table, allocate enough memory for number of rows, used by ip
void	NetworkEmptyMulticastForwardingTable (Node* node) empty out all the entries in the multicast forwarding table. basically set the size of table back to 0.
LinkedList*	NetworkGetOutgoingInterfaceFromMulticastForwardingTable (Node* node, NodeAddress sourceAddress, NodeAddress groupAddress) get the interface Id node that lead to the (source, multicast group) pair.
void	NetworkUpdateMulticastForwardingTable (Node* node, NodeAddress sourceAddress, NodeAddress multicastGroupAddress, int interfaceIndex) update entry with(sourceAddress,multicastGroupAddress) pair. search for the row with(sourceAddress,multicastGroupAddress) and update its interface.
void	NetworkPrintMulticastForwardingTable (Node* node) display all entries in multicast forwarding table of the node.
void	NetworkPrintMulticastOutgoingInterface (Node* node, list* list) Print mulitcast outgoing interfaces.
BOOL	NetworkInMulticastOutgoingInterface (Node* node, List* list, int interfaceIndex) Determine if interface is in multicast outgoing interface list.
void	NetworkIpPrintTraceXML (Node* node, Message* msg)

	Print packet trace information in XML format.
void	<code>RouteThePacketUsingLookupTable</code> (Node* node, Message* msg, int incomingInterface)
int	Tries to route and send the packet using the node's forwarding table. <code>GetNetworkIPFragUnit</code> (Node* node, int interfaceIndex)
void	Returns the network ip fragmentation unit. <code>NetworkIpUserProtocolInit</code> (Node* node, const NodeInput* nodeInput, const char* routingProtocolString, NetworkRoutingProtocolType* routingProtocolType, void** routingProtocolData)
void	Initialization of user protocol(disabled) <code>NetworkIpUserHandleProtocolEvent</code> (Node* node, Message* msg)
void	Event handler function of user protocol(disabled) <code>NetworkIpUserHandleProtocolPacket</code> (Node* node, Message* msg, unsigned char ipProtocol, NodeAddress sourceAddress, NodeAddress destinationAddress, int ttl)
void	Process a user protocol generated control packet(disabled) <code>NetworkIpUserProtocolFinalize</code> (Node* node, int userProtocolNumber)
void	Finalization of user protocol(disabled) <code>Atm_RouteThePacketUsingLookupTable</code> (Node* node, NodeAddress* destAddr, int* outIntf, NodeAddress* nextHop)
void	Routing packet received at ATM node <code>RouteThePacketUsingMulticastForwardingTable</code> (Node* node, Message* msg, int incomingInterface)
int	Tries to route the multicast packet using the multicast forwarding table. <code>NETWORKIpRoutingInit</code> (Node * node, const NodeInput *nodeInput nodeInput)
Int64	Initialization function for network layer. Initializes IP. <code>NetworkIpGetBandwidth</code> (Node* node, int interfaceIndex)
clocktype	getting the bandwidth information <code>NetworkIpGetPropDelay</code> (Node* node, int interfaceIndex)

	getting the propagation delay information
BOOL	NetworkIpInterfaceIsEnabled (Node* node, int interfaceIndex) To check the interface is enabled or not?
BOOL	NetworkIpIsWiredNetwork (Node* node, int interfaceIndex) Determines if an interface is a wired interface.
BOOL	NetworkIpIsPointToPointNetwork (Node* node, int interfaceIndex) Determines if an interface is a point-to-point.
BOOL	IsIPV4MulticastEnabledOnInterface (Node* node, int interfaceIndex) To check if IPV4 Multicast is enabled on interface?
BOOL	IsIPV4RoutingEnabledOnInterface (Node* node, int interfaceIndex) To check if IPV4 Routing is enabled on interface?
NetworkProtocolType	NetworkIpGetNetworkProtocolType (Node* node, NodeAddress nodeId) Get Network Protocol Type for the node
NetworkType	ResolveNetworkTypeFromSrcAndDestNodeId (Node* node, NodeId sourceNodeId, NodeId destNodeId) Resolve the NetworkType from source and destination node id's.
BOOL	NetworkIpIsWiredBroadcastNetwork (Node* node, int interfaceIndex) Determines if an interface is a wired interface.
ip_traceroute*	FindTraceRouteOption (const IpHeaderType* ipHeader) Searches the IP header for the Traceroute option field , and returns a pointer to traceroute header.

Constant / Data Structure Detail

Constant	IPVERSION4 4

Version of IP	
Constant	IPTOS_LOWDELAY 0x10 Type of service (low delay)
Constant	IPTOS_THROUGHPUT 0x08 Type of service (throughput)
Constant	IPTOS_RELIABILITY 0x04 Type of service (reliability)
Constant	IPTOS_MINCOST 0x02 Type of service (minimum cost)
Constant	IPTOS_ECT 0x02 Bits 6 and 7 in the IPv4 TOS octet are designated as the ECN field. Bit 6 is designated as the ECT bit.
Constant	IPTOS_CE 0x01 Bits 6 and 7 in the IPv4 TOS octet are designated as the ECN field. Bit 7 is designated as the CE bit.
Constant	IPTOS_DSCP_MAX 0x3f Bits 0 to 5 in the IPv4 TOS octet are designated as DSCP field.The range for this 6-bit field is < 0 - 63 >.
Constant	IPTOS_DSCP_MIN 0x00 Bits 0 to 5 in the IPv4 TOS octet are designated as DSCP field.The range for this 6-bit field is < 0 - 63 >.
Constant	IPTOS_PREC_EFINTERNETCONTROL 0xb8 IP precedence 'EF classe internet control'
Constant	IPTOS_PREC_NETCONTROL 0xe0

	IP precedence 'net control'
Constant	IPTOS_PREC_INTERNETCONTROL 0xc0 IP precedence 'internet control'
Constant	IPTOS_PREC_CRITIC_ECP 0xa0 IP precedence 'critic ecp'
Constant	IPTOS_PREC_FLASH OVERRIDE 0x80 IP precedence 'flash override'
Constant	IPTOS_PREC_FLASH 0x60 IP precedence 'flash'
Constant	IPTOS_PREC_IMMEDIATE 0x40 IP precedence 'immediate'
Constant	IPTOS_PREC_PRIORITY 0x20 IP precedence 'priority'
Constant	IPTOS_PREC_ROUTINE 0x00 IP precedence 'routing'
Constant	IPTOS_PREC_INTERNETCONTROL_MIN_DELAY_SET 0xd0 IP precedence 'internet control' with the 'minimize delay' bit set
Constant	IPTOS_PREC_CRITIC_ECP_MIN_DELAY_SET 0xb0 IP precedence 'critic ecp' with the 'minimize delay' bit set
Constant	IPOPT_CONTROL 0x00 IP option 'control'
Constant	IPOPT_RESERVED1 0x20

	IP option 'reserved1'.
Constant	IPOPT_DEBMEAS 0x40 IP option 'debmeas'
Constant	IPOPT_RESERVED2 0x60 IP option 'reserved2'
Constant	IPOPT_EOL 0 IP option 'end of option list'.
Constant	IPOPT_NOP 1 IP option 'no operation'.
Constant	IPOPT_RR 7 IP option 'record packet route'.
Constant	IPOPT_TS 68 IP option 'timestamp'.
Constant	IPOPT_SECURITY 130 IP option ' provide s,c,h,tcc'.
Constant	IPOPT_LSRR 131 IP option 'loose source route'.
Constant	IPOPT_SATID 136 IP option 'satnet id'.
Constant	IPOPT_SSRR 137

	IP option 'strict source route'.
Constant	IPOPT_TRCRT 82 IP option 'Traceroute'.
Constant	IPOPT_OPTVAL 0 Offset to IP option 'option ID'
Constant	IPOPT_OLEN 1 Offset to IP option 'option length'
Constant	IPOPT_OFFSET 2 Offset to IP option 'offset within option'
Constant	IPOPT_MINOFF 4 Offset to IP option 'min value of above'
Constant	IPOPT_TS_TSONLY 0 Flag bits for ipt_flg (timestamps only);
Constant	IPOPT_TS_TSANDADDR 1 Flag bits for ipt_flg (timestamps and addresses);
Constant	IPOPT_TS_PRESPEC 3 Flag bits for ipt_flg (specified modules only);
Constant	IPOPT_SECUR_UNCLASS 0x0000 'unclass' bits for security in IP option field
Constant	IPOPT_SECUR_CONFID 0xf135 'confid' bits for security in IP option field
Constant	IPOPT_SECUR_EFTO 0x789a

	'efto' bits for security in IP option field
Constant	IPOPT_SECUR_MMMM 0xbc4d
	'mmmm' bits for security in IP option field
Constant	IPOPT_SECUR_RESTR 0xaf13
	'restr' bits for security in IP option field
Constant	IPOPT_SECUR_SECRET 0xd788
	'secreat' bits for security in IP option field
Constant	IPOPT_SECUR_TOPSECRET 0x6bc5
	'top secret' bits for security in IP option field
Constant	MAXTTL 255
	Internet implementation parameters (maximum time to live (seconds))
Constant	IPDEFTTL 64
	Internet implementation parameters (default ttl, from RFC 1340)
Constant	IPFRAGTTL 60
	Internet implementation parameters (time to live for frags, slowhz)
Constant	IPTTLDEC 1
	Internet implementation parameters (subtracted when forwarding)
Constant	IPDEFTOS 0x10
	Internet implementation parameters (default TOS)
Constant	IP_MSS 576

	Internet implementation parameters (default maximum segment size)
Constant	IPPROTO_IP 0 IP protocol numbers.
Constant	IPPROTO_ICMP 1 IP protocol numbers for ICMP.
Constant	IPPROTO_IGMP 2 IP protocol numbers for IGMP.
Constant	IPPROTO_IPIP 4 IP protocol numbers for IP tunneling.
Constant	IPPROTO_TCP 6 IP protocol numbers for TCP .
Constant	IPPROTO_UDP 17 IP protocol numbers for UDP
Constant	IPPROTO_IPV6 41 IP protocol number for DUAL-IP.
Constant	IPPROTO_RSVP 46 IP protocol numbers for RSVP.
Constant	IPPROTO_MOBILE_IP 48 IP protocol numbers for MOBILE_IP.
Constant	IPPROTO_CES_HAIEP 49 IP protocol numbers.
Constant	IPPROTO_ESP 50

	IP protocol numbers for IPSEC.
Constant	IPPROTO_AH 51
	IP protocol numbers for IPSEC.
Constant	IPPROTO_ISAKMP 52
	IP protocol numbers for IPSEC.
Constant	IPPROTO_CES_ISAKMP 53
	IP protocol numbers for IPSEC.
Constant	IPPROTO_IAHEP 54
	IP protocol numbers.
Constant	IPPROTO OSPF 89
	IP protocol numbers for OSPF .
Constant	IPPROTO_PIM 103
	IP protocol numbers for PIM .
Constant	IPPROTO_RPIM 104
	IP protocol numbers for PIM .
Constant	IPPROTO_IGRP 100
	IP protocol numbers for IGRP .
Constant	IPPROTO_EIGRP 88
	IP protocol numbers for EIGRP .
Constant	IPPROTO_BELLMANFORD 150

	IP protocol numbers for BELLMANFORD.
Constant	IPPROTO_IPIP_RED 150 IP protocol numbers for IP_RED.
Constant	IPPROTO_FISHEYE 160 IP protocol numbers for FISHEYE .
Constant	IPPROTO_FSRL 161 IP protocol numbers for LANMAR .
Constant	IPPROTO_ANODR 162 IP protocol numbers for ANODR .
Constant	IPPROTO_SECURE_NEIGHBOR 163 IP protocol numbers for secure neighbor discovery .
Constant	IPPROTO_SECURE_COMMUNITY 164 IP protocol numbers for secure routing community
Constant	IPPROTO_NETWORK_CES_CLUSTER 165 IP protocol numbers for clustering protocol.
Constant	IPPROTO_ROUTING_CES_ROSPF 167 IP protocol numbers for ROSPF protocol.
Constant	IPPROTO_IPIP_ROUTING_CES_MALSR 168 IP protocol numbers for MALSR IP encapsulation.
Constant	IPPROTO_IPIP_ROUTING_CES_ROSPF 169 IP protocol numbers for ROSPF IP encapsulation.
Constant	IPPROTO_NETWORK_CES_REGION 170

	IP protocol numbers for RAP election protocol.
Constant	IPPROTO_MPR 171 IP protocol numbers for MPR
Constant	IPPROTO_IPIP_ROUTING_CES_SRW 173 IP protocol numbers for ROUTING_CES_SRW IP encapsulation.
Constant	IPPROTO_IPIP_SDR 174 IP protocol numbers for SDR IP encapsulation.
Constant	IPPROTO_IPIP_SDR 175 IP protocol numbers for SDR IP encapsulation.
Constant	IPPROTO_MULTICAST_CES_SRW_MOSPF 177 IP protocol numbers for MULTICAST_CES_SRW_MOSPF IP encapsulation.
Constant	IPPROTO_CES_HSLS 178 IP protocol numbers for HSLS protocol.
Constant	IPPROTO_AODV 123 IP protocol numbers for AODV .
Constant	IPPROTO_DYMO 132 IP protocol numbers for DYMO .
Constant	IPPROTO_MAODV 124 IP protocol numbers for MAODV.
Constant	IPPROTO_DSR 135

	IP protocol numbers for DSR .
Constant	IPPROTO_ODMRP 145 IP protocol numbers for ODMRP .
Constant	IPPROTO_LAR1 110 IP protocol numbers for LAR1.
Constant	IPPROTO_STAR 136 IP protocol numbers for STAR.
Constant	IPPROTO_DAWN 120 IP protocol numbers for DAWN.
Constant	IPPROTO_EPLRS 174 IP protocol numbers for EPLRS protocol.
Constant	IPPROTO_EPLRS 174 IP protocol numbers for EPLRS protocol.
Constant	IPPROTO_CES_EPLRS_MPR 179 IP protocol numbers for EPLRS MPR protocol.
Constant	IPPROTO_DVMRP 200 IP protocol numbers for DVMRP.
Constant	IPPROTO_GSM 202 IP protocol numbers for GSM.
Constant	IPPROTO_EXTERNAL 233 IP protocol for external interface.
Constant	IPPROTO_INTERNET_GATEWAY 240

	IP protocol numbers for Internet gateway for emulated nodes
Constant	IPPROTO_EXATA_VIRTUAL_LAN 241 IP protocol numbers for Internet gateway for emulated nodes
Constant	IPPROTO_NDP 255 IP protocol numbers for NDP.
Constant	IPPROTO_BRP 251 IP protocol numbers for BRP .
Constant	IP_MIN_HEADER_SIZE 20 Minimum IP header size in bytes
Constant	IP_MAX_HEADER_SIZE 60 Maximum IP header size in bytes
Constant	IP_FRAGMENT_HOLD_TIME 60 * SECOND Fragmented packets hold time.
Constant	IP_MIN_MULTICAST_ADDRESS 0xE0000000 Used to determine whether an IP address is multicast.
Constant	IP_MAX_MULTICAST_ADDRESS 0xFFFFFFFF Used to determine whether an IP address is multicast.
Constant	MULTICAST_DEFAULT_INTERFACE_ADDRESS 3758096384u Default multicast interface address (224.0.0.0).
Constant	IP_MIN_RESERVED_MULTICAST_ADDRESS 0xE0000000

	Minimum reserve multicast address (224.0.0.0).
Constant	IP_MAX_RESERVED_MULTICAST_ADDRESS 0xE00000FF
	Maximum reserve multicast address (224.0.0.255).
Constant	MULTICAST_DEFAULT_NUM_HOST_BITS 27
	Multicast default num host bit
Constant	NETWORK_UNREACHABLE -2
	Network unreachable.
Constant	DEFAULT_INTERFACE 0
	Default interface index
Constant	NETWORK_IP_REASS_BUFF_TIMER (15 * SECOND)
	Max time data can stored in assembly buffer
Constant	MAX_IP_FRAGMENTS_SIMPLE_CASE 64
	Max size of fragment allowed.
Constant	SMALL_REASSEMBLY_BUFFER_SIZE 2048
	Size of reassemble buffer
Constant	REASSEMBLY_BUFFER_EXPANSION_MULTIPLIER 8
	Multiplier used for reassemble buffer expansion
Enumeration	BackplaneType
	NetworkIp backplane type(either CENTRAL or DISTRIBUTED)
Structure	IpHeaderType
	IpHeaderType is 20 bytes,just like in the BSD code.
Structure	ip_timestamp

	Time stamp option structure.
Structure	ip_traceroute
	TraceRoute option structure.
Structure	NetworkIpBackplaneInfo
	Structure maintaining IP Back plane Information
Structure	ipHeaderSizeInfo
	Structure maintaining IP header size Information
Structure	NetworkMulticastForwardingTableRow
	Structure of an entity of multicast forwarding table.
Structure	NetworkMulticastForwardingTable
	Structure of multicast forwarding table
Structure	NetworkIpMulticastGroupEntry
	Structure for Multicast Group Entry
Structure	IpPerHopBehaviorInfoType
	Structure to maintain DS priority queue mapping
Structure	IpMftcParameter
	Variables of the structure define a unique condition class
Structure	IpMultiFieldTrafficConditionerInfo
	Structure used to store traffic condition.
Structure	IpOptionsHeaderType

	Structure of optional header for IP source route
Structure	NetworkIpStatsType Structure used to keep track of all stats of network layer.
Structure	NetworkForwardingTableRow Structure of an entity of forwarding table.
Structure	NetworkForwardingTable Structure of forwarding table.
Structure	IpInterfaceInfoType Structure for maintaining IP interface informations. This struct must be allocated by new, not MEM_malloc. All member variables MUST be initialized in the constructor.
Structure	ip_frag_data QualNet typedefs struct ip_frag_data to IpFragData. is a simple queue to hold fragmented packets.
Structure	Ipv6FragQueue Ipv6 fragment queue structure.
Structure	FragmetedMsg QualNet typedefs struct fragmeted_msg_struct to ip6q. struct fragmeted_msg_struct is a simple fragmented packets msg hold structure.
Structure	NetworkDataIp; Main structure of network layer.
Structure	IpReassemblyBufferType Structure of reassembly buffer
Structure	IpReassemblyBufferListCellType Structure of reassembly buffer cell listing

Structure	IpReassemblyBufferListType
Structure of reassembly buffer list	

Function / Macro Detail

Function / Macro	Format
IPOPT_COPIED(o)	IP option 'copied'.
IPOPT_CLASS(o)	IP option 'class'
IPOPT_NUMBER(o)	IP option 'number'
IpHeaderSize(ipHeader)	Returns IP header ip_hl field * 4, which is the size of the IP header in bytes.
SetIpHeaderSize(IpHeader, Size)	Sets IP header ip_hl field (header length) to Size divided by 4
FragmentOffset(ipHeader)	Starting position of this fragment in actual packet.
SetFragmentOffset(ipHeader, offset)	To set offset of fragment.
IpHeaderSetVersion() Set the value of version number for IpHeaderType	<p>void IpHeaderSetVersion() (UInt32* ip_v_hl_tos_len, unsigned int version)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • ip_v_hl_tos_len - The variable containing the value of ip_v • version - Input value for set operation <p>Returns:</p> <ul style="list-style-type: none"> • void - None
IpHeaderSetHLen() Set the value of header length for IpHeaderType	<p>void IpHeaderSetHLen() (UInt32* ip_v_hl_tos_len, unsigned int hlen)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • ip_v_hl_tos_len - The variable containing the value of ip_v • hlen - Input value for set operation

	<p>Returns:</p> <ul style="list-style-type: none"> • void - None
IpHeaderSetTOS()	<p>void IpHeaderSetTOS() (UInt32* ip_v_hl_tos_len, unsigned int ipTos)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • ip_v_hl_tos_len - The variable containing the value of ip_v • ipTos - Input value for set operation <p>Returns:</p> <ul style="list-style-type: none"> • void - None
IpHeaderSetIpLength()	<p>void IpHeaderSetIpLength() (UInt32* ip_v_hl_tos_len, unsigned int ipLen)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • ip_v_hl_tos_len - The variable containing the value of ip_v • ipLen - Input value for set operation <p>Returns:</p> <ul style="list-style-type: none"> • void - None
IpHeaderSetIpFragOffset()	<p>void IpHeaderSetIpFragOffset() (UInt16* ipFragment, UInt16 offset)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • ipFragment - The variable containing the value of • offset - Input value for set operation <p>Returns:</p> <ul style="list-style-type: none"> • void - None
IpHeaderSetIpReserved()	<p>void IpHeaderSetIpReserved() (UInt16* ipFragment, UInt16 ipReserved)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • ipFragment - The variable containing the value of • ipReserved - Input value for set operation <p>Returns:</p> <ul style="list-style-type: none"> • void - None
IpHeaderSetIpDontFrag()	<p>void IpHeaderSetIpDontFrag() (UInt16* ipFragment, UInt16 dontFrag)</p> <p>Parameters:</p>

<p>Set the value of ip_dont_fragment for IpHeaderType</p>	<ul style="list-style-type: none"> • <code>ipFragment</code> - The variable containing the value of • <code>dontFrag</code> - Input value for set operation <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
<p>IpHeaderSetIpMoreFrag()</p> <p>Set the value of ip_more_fragment for IpHeaderType</p>	<p><code>void IpHeaderSetIpMoreFrag() (UInt16* ipFragment, UInt16 moreFrag)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>ipFragment</code> - The variable containing the value of • <code>moreFrag</code> - Input value for set operation <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
<p>IpHeaderGetVersion()</p> <p>Returns the value of version number for IpHeaderType</p>	<p><code>unsigned int IpHeaderGetVersion() (UInt32 ip_v_hl_tos_len)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>ip_v_hl_tos_len</code> - The variable containing the value of ip_v <p>Returns:</p> <ul style="list-style-type: none"> • <code>unsigned int</code> - None
<p>IpHeaderGetHLen()</p> <p>Returns the value of header length for IpHeaderType</p>	<p><code>unsigned int IpHeaderGetHLen() (UInt32 ip_v_hl_tos_len)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>ip_v_hl_tos_len</code> - The variable containing the value of ip_v <p>Returns:</p> <ul style="list-style-type: none"> • <code>unsigned int</code> - None
<p>IpHeaderGetTOS()</p> <p>Returns the value of Type of Service for IpHeaderType</p>	<p><code>unsigned int IpHeaderGetTOS() (UInt32 ip_v_hl_tos_len)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>ip_v_hl_tos_len</code> - The variable containing the value of ip_v <p>Returns:</p> <ul style="list-style-type: none"> • <code>unsigned int</code> - None
<p>IpHeaderGetIpLength()</p>	<p><code>unsigned int IpHeaderGetIpLength() (UInt32 ip_v_hl_tos_len)</code></p> <p>Parameters:</p>

Returns the value of ip length for IpHeaderType	<ul style="list-style-type: none"> • <code>ip_v_hl_tos_len</code> - The variable containing the value of ip_v <p>Returns:</p> <ul style="list-style-type: none"> • <code>unsigned int</code> - None
IpHeaderGetIpFragOffset()	<p>UInt16 IpHeaderGetIpFragOffset() (UInt16 ipFragment)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>ipFragment</code> - The variable containing the value of <p>Returns:</p> <ul style="list-style-type: none"> • <code>UInt16</code> - None
IpHeaderGetIpDontFrag()	<p>BOOL IpHeaderGetIpDontFrag() (UInt16 ipFragment)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>ipFragment</code> - The variable containing the value of <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - None
IpHeaderGetIpMoreFrag()	<p>BOOL IpHeaderGetIpMoreFrag() (UInt16 ipFragment)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>ipFragment</code> - The variable containing the value of <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - None
IpHeaderGetIpReserved()	<p>BOOL IpHeaderGetIpReserved() (UInt16 ipFragment)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>ipFragment</code> - The variable containing the value of <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - None
Ip_timestampSetFlag()	<p>void Ip_timestampSetFlag() (unsigned char flgOflw, unsigned char flag)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>flgOflw</code> - The variable containing the value of flag and • <code>flag</code> - Input value for set operation <p>Returns:</p>

	<ul style="list-style-type: none"> • void - None
Ip_timestampSetOvflw()	<p>void Ip_timestampSetOvflw() (unsigned char flgOflw, unsigned char ovflw)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • flgOflw - The variable containing the value of flag and • ovflw - Input value for set operation <p>Returns:</p> <ul style="list-style-type: none"> • void - None
Ip_timestampGetFlag()	<p>unsigned char Ip_timestampGetFlag() (unsigned char flgOflw)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • flgOflw - The variable containing the value of flag and <p>Returns:</p> <ul style="list-style-type: none"> • unsigned char - None
Ip_timestampGetOvflw()	<p>unsigned char Ip_timestampGetOvflw() (unsigned char flgOflw)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • flgOflw - The variable containing the value of flag and <p>Returns:</p> <ul style="list-style-type: none"> • unsigned char - None
ConvertNumHostBitsToSubnetMask	<p>NodeAddress ConvertNumHostBitsToSubnetMask (int numHostBits)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • numHostBits - number of host bit. <p>Returns:</p> <ul style="list-style-type: none"> • NodeAddress - subnetmask
ConvertSubnetMaskToNumHostBits	<p>int ConvertSubnetMaskToNumHostBits (NodeAddress subnetMask)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • subnetMask - subnetmask. <p>Returns:</p> <ul style="list-style-type: none"> • int - number of host bit.

MaskIpAddress	<p>To mask a ip address.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>address</code> - address of a node • <code>mask</code> - mask of subnet. <p>Returns:</p> <ul style="list-style-type: none"> • <code>NodeAddress</code> - masked node address.
MaskIpAddressWithNumHostBits	<p>To mask a ip address.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>address</code> - address of a node. • <code>numHostBits</code> - number of host bit. <p>Returns:</p> <ul style="list-style-type: none"> • <code>NodeAddress</code> - masked node address.
CalcBroadcastIpAddress	<p>To generate broadcast address.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>address</code> - address of a node. • <code>numHostBits</code> - number of host bit. <p>Returns:</p> <ul style="list-style-type: none"> • <code>NodeAddress</code> - Broadcast address.
IsIpAddressInSubnet	<p>To check if a ip address belongs to a subnet.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>address</code> - address of a node. • <code>subnetAddress</code> - address of a subnet. • <code>numHostbits</code> - number of host bit. <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - TRUE if ip address belongs to a subnet else FALSE.
NetworkIpAddHeader	<p>Add an IP packet header to a message. Just calls AddIpHeader.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>Node* node</code>, <code>Message* msg</code>, <code>NodeAddress sourceAddress</code>, <code>NodeAddress destinationAddress</code>, <code>TosType priority</code>, <code>unsigned char protocol</code>, <code>unsigned ttl</code>

	<ul style="list-style-type: none"> • <code>node</code> - Pointer to node. • <code>msg</code> - Pointer to message. • <code>sourceAddress</code> - Source IP address. • <code>destinationAddress</code> - Destination IP address. • <code>priority</code> - Currently a TosType. • <code>protocol</code> - IP protocol number. • <code>ttl</code> - Time to live. If 0, uses default <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
FindAnIpOptionField	<p><code>IpOptionsHeaderType* FindAnIpOptionField (const IpHeaderType* ipHeader, const int optionKey)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>ipHeader</code> - Pointer to an IP header. • <code>optionKey</code> - Option code for desired option field. <p>Returns:</p> <ul style="list-style-type: none"> • <code>IpOptionsHeaderType*</code> - to the header of the desired option field. NULL if no option fields, or the desired option field cannot be found.
NetworkIpPreInit	<p><code>void NetworkIpPreInit (Node* node)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - pointer to node. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
NetworkIpInit	<p><code>void NetworkIpInit (Node* node, const NodeInput* nodeInput)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - pointer to node. • <code>nodeInput</code> - Pointer to node input. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
NetworkIpLayer	<code>void NetworkIpLayer (Node* node, Message* msg)</code>

	<p>Handle IP layer events, incoming messages and messages sent to itself (timers, etc.).</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • msg - Pointer to message. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
<p>NetworkIpFinalize</p> <p>Finalize function for the IP model. Finalize functions for all network-layer IP protocols are called here.</p>	<p>void NetworkIpFinalize (Node* node)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
<p>NetworkIpReceivePacketFromTransportLayer</p> <p>Called by transport layer protocols (UDP, TCP) to send UDP datagrams and TCP segments using IP. Simply calls NetworkIpSendRawMessage().</p>	<p>void NetworkIpReceivePacketFromTransportLayer (Node* node, Message* msg, NodeAddress sourceAddress, NodeAddress destinationAddress, int outgoingInterface, TosType priority, unsigned char protocol, BOOL isEcnCapable)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • msg - Pointer to message from transport • sourceAddress - Source IP address. • destinationAddress - Destination IP address. • outgoingInterface - outgoing interface to use to • priority - Priority of packet. • protocol - IP protocol number. • isEcnCapable - Is this node ECN capable? <p>Returns:</p> <ul style="list-style-type: none"> • void - None
<p>NetworkIpSendRawMessage</p> <p>Called by NetworkIpReceivePacketFromTransportLayer() to send to send UDP datagrams, TCP segments using IP. Also called by network-layer routing protocols (AODV, OSPF, etc.) to send IP packets. This function adds an IP header and calls RoutePacketAndSendToMac().</p>	<p>void NetworkIpSendRawMessage (Node* node, Message* msg, NodeAddress sourceAddress, NodeAddress destinationAddress, int outgoingInterface, TosType priority, unsigned char protocol, unsigned ttl)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • msg - Pointer to message with payload data

	<ul style="list-style-type: none"> • <code>sourceAddress</code> - Source IP address. • <code>destinationAddress</code> - Destination IP address. • <code>outgoingInterface</code> - outgoing interface to use to • <code>priority</code> - Priority of packet. • <code>protocol</code> - IP protocol number. • <code>ttl</code> - Time to live. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
NetworkIpSendRawMessageWithDelay	<p>Same as <code>NetworkIpSendRawMessage()</code>, but schedules event after a simulation delay.</p> <p><code>void NetworkIpSendRawMessageWithDelay (Node* node, Message* msg, NodeAddress sourceAddress, NodeAddress destinationAddress, int outgoingInterface, TosType priority, unsigned char protocol, unsigned ttl, clocktype delay)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node. • <code>msg</code> - Pointer to message with payload data • <code>sourceAddress</code> - Source IP address. • <code>destinationAddress</code> - Destination IP address. • <code>outgoingInterface</code> - outgoing interface to use to • <code>priority</code> - TOS of packet. • <code>protocol</code> - IP protocol number. • <code>ttl</code> - Time to live. • <code>delay</code> - Delay <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
NetworkIpSendRawMessageToMacLayer	<p>Called by network-layer routing protocols (AODV, OSPF, etc.) to add an IP header to payload data, and with the resulting IP packet, calls <code>NetworkIpSendPacketOnInterface()</code>.</p> <p><code>void NetworkIpSendRawMessageToMacLayer (Node* node, Message* msg, NodeAddress sourceAddress, NodeAddress destinationAddress, TosType priority, unsigned char protocol, unsigned ttl, int interfaceIndex, NodeAddress nextHop)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node. • <code>msg</code> - Pointer to message with payload data

	<p><code>sourceAddress</code> - Source IP address.</p> <ul style="list-style-type: none"> • <code>destinationAddress</code> - Destination IP address. • <code>priority</code> - TOS of packet. • <code>protocol</code> - IP protocol number. • <code>ttl</code> - Time to live. • <code>interfaceIndex</code> - Index of outgoing interface. • <code>nextHop</code> - Next hop IP address. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
NetworkIpSendRawMessageToMacLayerWithDelay	<p>Same as <code>NetworkIpSendRawMessageToMacLayer()</code>, but schedules the event after a simulation delay by calling <code>NetworkIpSendPacketOnInterfaceWithDelay()</code>.</p> <p><code>void NetworkIpSendRawMessageToMacLayerWithDelay (Node* node, Message* msg, NodeAddress sourceAddress, NodeAddress destinationAddress, TosType priority, unsigned char protocol, unsigned ttl, int interfaceIndex, NodeAddress nextHop, clocktype delay)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node. • <code>msg</code> - Pointer to message with payload data • <code>sourceAddress</code> - Source IP address. • <code>destinationAddress</code> - Destination IP address. • <code>priority</code> - TOS of packet. • <code>protocol</code> - IP protocol number. • <code>ttl</code> - Time to live. • <code>interfaceIndex</code> - Index of outgoing interface. • <code>nextHop</code> - Next hop IP address. • <code>delay</code> - delay. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
NetworkIpSendPacketToMacLayer	<p>This function is called once the outgoing interface index and next hop address to which to route an IP packet are known.</p> <p><code>void NetworkIpSendPacketToMacLayer (Node* node, Message* msg, int interfaceIndex, NodeAddress nextHop)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node.

	<ul style="list-style-type: none"> • <code>msg</code> - Pointer to message with ip packet. • <code>interfaceIndex</code> - Index of outgoing interface. • <code>nextHop</code> - Next hop IP address. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
NetworkIpSendPacketOnInterface	<p>This function is called once the outgoing interface index and next hop address to which to route an IP packet are known. This queues an IP packet for delivery to the MAC layer. This functions calls <code>QueueUpIpFragmentForMacLayer()</code>. This function is used to initiate fragmentation if required, but since fragmentation has been disabled, all it does is assert false if the IP packet is too big before calling the next function.</p> <p>void NetworkIpSendPacketOnInterface (Node* node, Message* msg, int incomingInterface, int outgoingInterface, NodeAddress nextHop)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node. • <code>msg</code> - Pointer to message with ip packet. • <code>incomingInterface</code> - Index of incoming interface. • <code>outgoingInterface</code> - Index of outgoing interface. • <code>nextHop</code> - Next hop IP address. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
NetworkIpSendPacketToMacLayerWithDelay	<p>Same as <code>NetworkIpSendPacketOnInterface()</code>, but schedules event after a simulation delay.</p> <p>void NetworkIpSendPacketToMacLayerWithDelay (Node* node, Message* msg, int interfaceIndex, NodeAddress nextHop, clocktype delay)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node. • <code>msg</code> - Pointer to message with ip packet. • <code>interfaceIndex</code> - Index of outgoing interface. • <code>nextHop</code> - Next hop IP address. • <code>delay</code> - delay <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
NetworkIpSendPacketOnInterfaceWithDelay	<p>Same as <code>NetworkIpSendPacketOnInterface()</code>, but schedules event after a simulation delay.</p> <p>void NetworkIpSendPacketOnInterfaceWithDelay (Node* node, Message* msg, int incomingInterface, int outgoingInterface, NodeAddress nextHop, clocktype delay)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node.

	<ul style="list-style-type: none"> • <code>msg</code> - Pointer to message with ip packet. • <code>incommingInterface</code> - Index of incomming interface. • <code>outgoingInterface</code> - Index of outgoing interface. • <code>nextHop</code> - Next hop IP address. • <code>delay</code> - delay <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
NetworkIpSendRawPacketOnInterfaceWithDelay	<p>Same as <code>NetworkIpSendPacketOnInterface()</code>, but schedules event after a simulation delay and denotes raw packet.</p> <p><code>void NetworkIpSendRawPacketOnInterfaceWithDelay (Node* node, Message* msg, int incommingInterface, int outgoingInterface, NodeAddress nextHop, clocktype delay)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node. • <code>msg</code> - Pointer to message with ip packet. • <code>incommingInterface</code> - Index of incomming interface. • <code>outgoingInterface</code> - Index of outgoing interface. • <code>nextHop</code> - Next hop IP address. • <code>delay</code> - delay <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
NetworkIpSendPacketToMacLayerWithNewStrictSourceRoute	<p>Tacks on a new source route to an existing IP packet and then sends the packet to the MAC layer.</p> <p><code>void NetworkIpSendPacketToMacLayerWithNewStrictSourceRoute (Node* node, Message* msg, NodeAddress[] newRouteAddresses, int numNewRouteAddresses, BOOL removeExistingRecordedRoute)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node. • <code>msg</code> - Pointer to message with ip packet. • <code>newRouteAddresses</code> - Source route (address array). • <code>numNewRouteAddresses</code> - Number of array elements. • <code>removeExistingRecordedRoute</code> - Flag to indicate previous record <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
NetworkIpReceivePacketFromMacLayer	<code>void NetworkIpReceivePacketFromMacLayer (Node* node, Message* msg,</code>

	<p>IP received IP packet from MAC layer. Updates the Stats database and then calls NetworkIpReceivePacket.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • msg - Pointer to message with ip packet. • previousHopNodeId - nodeId of the previous hop. • interfaceIndex - Index of interface on which packet arrived. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
NetworkIpReceivePacket	<p>IP received IP packet. Determine whether the packet is to be delivered to this node, or needs to be forwarded. ipHeader->ip_ttl is decremented here, instead of the way BSD TCP/IP does it, which is to decrement TTL right before forwarding the packet. QualNet's alternative method suits its network-layer ad hoc routing protocols, which may do their own forwarding.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • msg - Pointer to message with ip packet. • previousHopNodeId - nodeId of the previous hop. • interfaceIndex - Index of interface on which packet arrived. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
NetworkIpNotificationOfPacketDrop	<p>Invoke callback functions when a packet is dropped.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • msg - Pointer to message with ip packet. • nextHopNodeAddres - next hop address of dropped packet. • interfaceIndex - interface that experienced the packet drop. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
NetworkIpGetMacLayerStatusEventHandlerFunction	<p>Get the status event handler function pointer.</p> <p>MacLayerStatusEventHandlerFunctionType NetworkIpGetMacLayerStatusEventHandlerFunction (Node* node, int interfaceIndex)</p> <p>Parameters:</p>

	<ul style="list-style-type: none"> • node - Pointer to node. • interfaceIndex - interface associated with the status <p>Returns:</p> <ul style="list-style-type: none"> • MacLayerStatusEventHandlerFunctionType - Status event handler function.
NetworkIpSetMacLayerStatusEventHandlerFunction	<p>Allows the MAC layer to send status messages (e.g., packet drop, link failure) to a network-layer routing protocol for routing optimization.</p> <p>void NetworkIpSetMacLayerStatusEventHandlerFunction (Node* node, MacLayerStatusEventHandlerFunctionType StatusEventHandlerPtr, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • StatusEventHandlerPtr - interface • interfaceIndex - interface associated with the status <p>Returns:</p> <ul style="list-style-type: none"> • void - None
NetworkIpSneakPeekAtMacPacket	<p>Called Directly by the MAC layer, this allows a routing protocol to "sneak a peek" or "tap" messages it would not normally see from the MAC layer. This function will possibly unfragment such packets and call the function registered by the routing protocol to do the "Peek".</p> <p>void NetworkIpSneakPeekAtMacPacket (Node* node, const Message* msg, int interfaceIndex, NodeAddress prevHop)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • msg - The message being peeked at from the • interfaceIndex - The interface of which the "peeked" message belongs to. • prevHop - next hop address of dropped packet. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
NetworkIpGetPromiscuousMessagePeekFunction	<p>Returns the network-layer function which will promiscuously inspect packets. See NetworkIpSneakPeekAtMacPacket().</p> <p>PromiscuousMessagePeekFunctionType NetworkIpGetPromiscuousMessagePeekFunction (Node* node, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • interfaceIndex - Interface associated with the peek function. <p>Returns:</p> <ul style="list-style-type: none"> • PromiscuousMessagePeekFunctionType - Function pointer
NetworkIpSetPromiscuousMessagePeekFunction	<p>void NetworkIpSetPromiscuousMessagePeekFunction (Node* node, PromiscuousMessagePeekFunctionType PeekFunctionPtr, int interfaceIndex)</p>

<p>Sets the network-layer function which will promiscuously inspect packets. See NetworkIpSneakPeekAtMacPacket().</p>	<p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node. • <code>PeekFunctionPtr</code> - Peek function. • <code>interfaceIndex</code> - Interface associated with the peek function. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
<p>NetworkIpReceiveMacAck</p> <p>MAC received an ACK, so call ACK handler function.</p>	<p><code>void NetworkIpReceiveMacAck (Node* node, int interfaceIndex, const Message* msg, NodeAddress nextHop)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node. • <code>interfaceIndex</code> - Interface associated with the ACK handler function. • <code>msg</code> - Message that was ACKed. • <code>nextHop</code> - Next hop that sent the MAC layer ACK <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
<p>NetworkIpGetMacLayerAckHandler</p> <p>Get MAC layer ACK handler</p>	<p><code>MacLayerAckHandlerType NetworkIpGetMacLayerAckHandler (Node* node, int interfaceIndex)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node. • <code>interfaceIndex</code> - Interface associated with ACK handler function <p>Returns:</p> <ul style="list-style-type: none"> • <code>MacLayerAckHandlerType</code> - MAC acknowledgement function pointer
<p>NetworkIpSetMacLayerAckHandler</p> <p>Set MAC layer ACK handler</p>	<p><code>void NetworkIpSetMacLayerAckHandler (Node* node, MacLayerAckHandlerType macAckHandlerPtr, int interfaceIndex)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node. • <code>macAckHandlerPtr</code> - Callback function handling • <code>interfaceIndex</code> - Interface associated with the ACK handler <p>Returns:</p>

	<ul style="list-style-type: none"> • void - None
SendToUdp	<p>Sends a UDP packet to UDP in the transport layer. The source IP address, destination IP address, and priority of the packet are also sent.</p> <p>void SendToUdp (Node* node, Message* msg, TosType priority, NodeAddress sourceAddress, NodeAddress destinationAddress, int incomingInterfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • msg - Pointer to message with UDP packet. • priority - TOS of UDP packet. • sourceAddress - Source IP address. • destinationAddress - Destination IP address. • incomingInterfaceIndex - interface that received the packet <p>Returns:</p> <ul style="list-style-type: none"> • void - None
SendToTcp	<p>Sends a TCP packet to TCP in the transport layer. The source IP address, destination IP address, and priority of the packet are also sent..</p> <p>void SendToTcp (Node* node, Message* msg, TosType priority, NodeAddress sourceAddress, NodeAddress destinationAddress, BOOL aCongestionExperienced)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • msg - Pointer to message with TCP packet. • priority - TOS of TCP packet. • sourceAddress - Source IP address. • destinationAddress - Destination IP address. • aCongestionExperienced - Determine if congestion is <p>Returns:</p> <ul style="list-style-type: none"> • void - None
SendToRsvp	<p>Sends a RSVP packet to RSVP in the transport layer. The source IP address, destination IP address, and priority of the packet are also sent.</p> <p>void SendToRsvp (Node* node, Message* msg, TosType priority, NodeAddress sourceAddress, NodeAddress destinationAddress, int interfaceIndex, unsigned ttl)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • msg - Pointer to message with RSVP packet. • priority - TOS of UDP packet.

	<ul style="list-style-type: none"> • <code>sourceAddress</code> - Source IP address. • <code>destinationAddress</code> - Destination IP address. • <code>interfaceIndex</code> - incoming interface index. • <code>ttl</code> - Receiving TTL <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
NetworkIpRemoveIpHeader	<p>Removes the IP header from a message while also returning all the fields of the header.</p> <p><code>void NetworkIpRemoveIpHeader (Node* node, Message* msg, NodeAddress* sourceAddress, NodeAddress* destinationAddress, TosType* priority, unsigned char* protocol, unsigned* ttl)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node. • <code>msg</code> - Pointer to message • <code>sourceAddress</code> - Storage for source IP address. • <code>destinationAddress</code> - Storage for destination IP • <code>priority</code> - Storage for TosType.(values are • <code>protocol</code> - Storage for IP protocol number • <code>ttl</code> - Storage for time to live. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
AddIpOptionField	<p>Inserts an option field in the header of an IP packet.</p> <p><code>void AddIpOptionField (Node* node, Message* msg, int optionCode, int optionSize)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node. • <code>msg</code> - Pointer to message • <code>optionCode</code> - The option code • <code>optionSize</code> - Size of the option <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
ExtractIpSourceAndRecordedRoute	<p><code>void ExtractIpSourceAndRecordedRoute (Message* msg, NodeAddress[] RouteAddresses, int* NumAddresses, int* RouteAddressIndex)</code></p> <p>Parameters:</p>

<p>Retrieves a copy of the source and recorded route from the options field in the header.</p>	<ul style="list-style-type: none"> • <code>msg</code> - Pointer to message with IP packet. • <code>RouteAddresses</code> - Storage for source/recoded route. • <code>NumAddresses</code> - Storage for size of <code>RouteAddresses[]</code> array. • <code>RouteAddressIndex</code> - The index of the first address of the <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
<p>NetworkIpGetRouterFunction</p> <p>Get the router function pointer.</p>	<p>RouterFunctionType NetworkIpGetRouterFunction (Node* node, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node. • <code>interfaceIndex</code> - interface associated with router function <p>Returns:</p> <ul style="list-style-type: none"> • <code>RouterFunctionType</code> - router function pointer.
<p>NetworkIpSetRouterFunction</p> <p>Allows a routing protocol to set the "routing function" (one of its functions) which is called when a packet needs to be routed.</p> <p><code>NetworkIpSetRouterFunction()</code> allows a routing protocol to define the routing function. The routing function is called by the network layer to ask the routing protocol to route the packet. The routing function is given the packet and its destination. The routing protocol can route the packet and set "PacketWasRouted" to TRUE; or not route the packet and set to FALSE. If the packet, was not routed, then the network layer will try to use the forwarding table or the source route the source route in the IP header. This function will also be given packets for the local node the routing protocols can look at packets for protocol reasons. In this case, the message should not be modified and <code>PacketWasRouted</code> must be set to FALSE.</p>	<p>void NetworkIpSetRouterFunction (Node* node, RouterFunctionType RouterFunctionPtr, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node. • <code>RouterFunctionPtr</code> - Router function to set. • <code>interfaceIndex</code> - interface associated with router function. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
<p>NetworkIpAddUnicastRoutingProtocolType</p> <p>Add unicast routing protocol type to interface.</p>	<p>void NetworkIpAddUnicastRoutingProtocolType (Node* node, NetworkRoutingProtocolType routingProtocolType, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node. • <code>routingProtocolType</code> - Router function to add. • <code>interfaceIndex</code> - Interface associated with the router function.

	<p>Returns:</p> <ul style="list-style-type: none"> • void - None
NetworkIpAddUnicastIntraRegionRoutingProtocolType	<p>Add unicast intra region routing protocol type to interface.</p> <p>void NetworkIpAddUnicastIntraRegionRoutingProtocolType (Node* node, NetworkRoutingProtocolType routingProtocolType, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • routingProtocolType - Router function to add. • interfaceIndex - Interface associated with the router function. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
NetworkIpGetRoutingProtocol	<p>Get routing protocol structure associated with routing protocol running on this interface.</p> <p>void* NetworkIpGetRoutingProtocol (Node* node, NetworkRoutingProtocolType routingProtocolType)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • routingProtocolType - Router function to <p>Returns:</p> <ul style="list-style-type: none"> • void* - Routing protocol structure requested.
NetworkIpGetUnicastRoutingProtocolType	<p>Get unicast routing protocol type on this interface.</p> <p>NetworkRoutingProtocolType NetworkIpGetUnicastRoutingProtocolType (Node* node, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • interfaceIndex - network interface for request. <p>Returns:</p> <ul style="list-style-type: none"> • NetworkRoutingProtocolType - The unicast routing protocol type.
NetworkIpSetHsrpOnInterface	<p>To enable hsrp on a interface</p> <p>void NetworkIpSetHsrpOnInterface (Node* node, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • interfaceIndex - network interface. <p>Returns:</p> <ul style="list-style-type: none"> • void - None

NetworkIpIsHsrpEnabled To test if any interface is hsrp enabled.	<p>BOOL NetworkIpIsHsrpEnabled (Node* node, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • interfaceIndex - network interface. <p>Returns:</p> <ul style="list-style-type: none"> • BOOL - return TRUE if any one interface is hsrp enabled else return FALSE.
NetworkIpAddNewInterface Add new interface to node.	<p>void NetworkIpAddNewInterface (Node* node, NodeAddress interfaceIpAddress, int numHostBits, int* newInterfaceIndex, const NodeInput* nodeInput)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • interfaceIpAddress - Interface to add. • numHostBits - Number of host bits for the interface. • newInterfaceIndex - The interface number of the new interface. • nodeInput - Provides access to <p>Returns:</p> <ul style="list-style-type: none"> • void - None
NetworkIpInitCpuQueueConfiguration Initializes cpu queue parameters during startup.	<p>void NetworkIpInitCpuQueueConfiguration (Node* node, const NodeInput* nodeInput)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • nodeInput - Pointer to node input. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
NetworkIpInitInputQueueConfiguration Initializes input queue parameters during startup.	<p>void NetworkIpInitInputQueueConfiguration (Node* node, const NodeInput* nodeInput, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • nodeInput - Pointer to node input. • interfaceIndex - interface associated with queue.

	<p>Returns:</p> <ul style="list-style-type: none"> • void - None
NetworkIpInitOutputQueueConfiguration	<p>void NetworkIpInitOutputQueueConfiguration (Node* node, const NodeInput* nodeInput, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • nodeInput - Pointer to node input. • interfaceIndex - interface associated with queue. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
NetworkIpCreateQueues	<p>void NetworkIpCreateQueues (Node* node, const NodeInput* nodeInput, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • nodeInput - Pointer to node input. • interfaceIndex - interface associated with queue. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
NetworkIpSchedulerParameterInit	<p>void NetworkIpSchedulerParameterInit (Scheduler* schedulerPtr, const int numPriorities, Queue* queue)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • schedulerPtr - pointer to scheduler • numPriorities - Number of priorities available • queue - pointer to ip queue. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
NetworkIpSchedulerInit	<p>void NetworkIpSchedulerInit (Node* node, const NodeInput* nodeInput, int interfaceIndex, Scheduler* schedulerPtr, const char* schedulerTypeString)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - pointer to node • nodeInput - pointer to nodeinput

	<ul style="list-style-type: none"> • <code>interfaceIndex</code> - interface index • <code>schedulerPtr</code> - type of Scheduler • <code>schedulerTypeString</code> - Scheduler name <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
NetworkIpCpuQueueInsert	<p>Calls the cpu packet scheduler for an interface to retrieve an IP packet from a queue associated with the interface. The dequeued packet, since it's already been routed, has an associated next-hop IP address. The packet's priority value is also returned.</p> <p><code>void NetworkIpCpuQueueInsert (Node* node, Message* msg, NodeAddress nextHopAddress, NodeAddress destinationAddress, int outgoingInterface, int networkType, BOOL* queueIsFull, int incomingInterface)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node. • <code>msg</code> - Pointer to message with IP packet. • <code>nextHopAddress</code> - Packet's next hop address. • <code>destinationAddress</code> - Packet's destination address. • <code>outgoingInterface</code> - Used to determine where packet • <code>networkType</code> - Type of network packet is using (IP, ...) • <code>queueIsFull</code> - Storage for boolean indicator. • <code>incomingInterface</code> - Incoming interface of packet. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
NetworkIpInputQueueInsert	<p>Calls input packet scheduler for an interface to retrieve an IP packet from a queue associated with the interface. The dequeued packet, since it's already been routed, has an associated next-hop IP address. The packet's priority value is also returned.</p> <p><code>void NetworkIpInputQueueInsert (Node* node, int incomingInterface, Message* msg, NodeAddress nextHopAddress, NodeAddress destinationAddress, int outgoingInterface, int networkType, BOOL* queueIsFull)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node. • <code>incomingInterface</code> - interface of input queue. • <code>msg</code> - Pointer to message with IP packet. • <code>nextHopAddress</code> - Packet's next hop address. • <code>destinationAddress</code> - Packet's destination address. • <code>outgoingInterface</code> - Used to determine where packet

	<ul style="list-style-type: none"> • <code>networkType</code> - Type of network packet is using (IP, • <code>queueIsFull</code> - Storage for boolean indicator. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
NetworkIpOutputQueueInsert	<p>Calls output packet scheduler for an interface to retrieve an IP packet from a queue associated with the interface. The dequeued packet, since it's already been routed, has an associated next-hop IP address. The packet's priority value is also returned. Called by <code>QueueUpIpFragmentForMacLayer()</code>.</p> <p>void NetworkIpOutputQueueInsert (Node* node, int interfaceIndex, Message* msg, NodeAddress nextHopAddress, NodeAddress destinationAddress, int networkType, BOOL* queueIsFull)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node. • <code>interfaceIndex</code> - interface of input queue. • <code>msg</code> - Pointer to message with IP packet. • <code>nextHopAddress</code> - Packet's next hop address. • <code>destinationAddress</code> - Packet's destination address. • <code>networkType</code> - Type of network packet is using (IP, • <code>queueIsFull</code> - Storage for boolean indicator. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
NetworkIpInputQueueDequeuePacket	<p>Calls the packet scheduler for an interface to retrieve an IP packet from the input queue associated with the interface.</p> <p>BOOL NetworkIpInputQueueDequeuePacket (Node* node, int incomingInterface, Message** msg, NodeAddress* nextHopAddress, int* outgoingInterface, int* networkType, QueuePriorityType* priority)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node. • <code>incomingInterface</code> - interface to dequeue from. • <code>msg</code> - Storage for pointer to message • <code>nextHopAddress</code> - Storage for Packet's • <code>outgoingInterface</code> - Used to determine where packet • <code>networkType</code> - Type of network packet is using (IP, • <code>priority</code> - Storage for <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - TRUE if dequeued successfully, FALSE otherwise.
NetworkIpOutputQueueDequeuePacket	<p>BOOL NetworkIpOutputQueueDequeuePacket (Node* node, int interfaceIndex, Message** msg,</p>

	<p>Calls the packet scheduler for an interface to retrieve an IP packet from a queue associated with the interface. The dequeued packet, since it's already been routed, has an associated next-hop IP address. The packet's priority value is also returned. This function is called by MAC_OutputQueueDequeuePacket() (mac/mac.pc), which itself is called from mac/mac_802_11.pc and other MAC protocol source files. This function will assert false if the scheduler cannot return an IP packet for whatever reason.</p>
NetworkIpOutputQueueDequeuePacketForAPriority	<p>Same as NetworkIpOutputQueueDequeuePacket(), except the packet dequeued is requested by a specific priority, instead of leaving the priority decision up to the packet scheduler. This function is called by MAC_OutputQueueDequeuePacketForAPriority() (mac/mac.pc), which itself is called from mac/mac_802_11.pc and other MAC protocol source files. This function will assert false if the scheduler cannot return an IP packet for whatever reason.</p>
NetworkIpOutputQueueDequeuePacketWithIndex	<p>Same as NetworkIpOutputQueueDequeuePacket(), except the packet dequeued is requested by a specific index. This function is called by MAC_OutputQueueDequeuePacketForAPriority() (mac/mac.pc), which itself is called from mac/mac_802_11.pc and other MAC protocol source files. This function will assert false if the scheduler cannot return an IP packet for whatever reason.</p>

	<ul style="list-style-type: none"> • <code>msg</code> - Storage for pointer to message • <code>nextHopAddress</code> - Storage for Packet's next hop address. • <code>networkType</code> - Type of network packet is using (IP, <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - TRUE if dequeued successfully, FALSE otherwise.
NetworkIpInputQueueTopPacket	<p>Same as <code>NetworkIpInputQueueDequeuePacket()</code>, except the packet is not actually dequeued. Note that the message containing the packet is not copied; the contents may (inadvertently or not) be directly modified.</p> <p>BOOL NetworkIpInputQueueTopPacket (<code>Node* node</code>, <code>int incomingInterface</code>, <code>Message** msg</code>, <code>NodeAddress* nextHopAddress</code>, <code>int* outgoingInterface</code>, <code>int* networkType</code>, <code>QueuePriorityType* priority</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node. • <code>incomingInterface</code> - interface to get top packet from. • <code>msg</code> - Storage for pointer to message • <code>nextHopAddress</code> - Storage for Packet's next hop addr. • <code>outgoingInterface</code> - Used to determine where packet should go • <code>networkType</code> - Type of network packet is using (IP, • <code>priority</code> - Storage for priority <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - TRUE if there is a packet, FALSE otherwise.
NetworkIpOutputQueueTopPacket	<p>Same as <code>NetworkIpOutputQueueDequeuePacket()</code>, except the packet is not actually dequeued. Note that the message containing the packet is not copied; the contents may (inadvertently or not) be directly modified. This function is called by <code>MAC_OutputQueueTopPacket()</code> (<code>mac/mac.pc</code>), which itself is called from <code>mac/mac_802_11.pc</code> and other MAC protocol source files. This function will assert false if the scheduler cannot return an IP packet for whatever reason.</p> <p>BOOL NetworkIpOutputQueueTopPacket (<code>Node* node</code>, <code>int interfaceIndex</code>, <code>Message** msg</code>, <code>NodeAddress* nextHopAddress</code>, <code>int* networkType</code>, <code>QueuePriorityType* priority</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node. • <code>interfaceIndex</code> - index to interface . • <code>msg</code> - Storage for pointer to message • <code>nextHopAddress</code> - Storage for Packet's next hop addr. • <code>networkType</code> - Type of network of the packet • <code>priority</code> - Storage for priority <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - TRUE if there is a packet, FALSE otherwise.
NetworkIpOutputQueuePeekWithIndex	BOOL NetworkIpOutputQueuePeekWithIndex (<code>Node* node</code> , <code>int interfaceIndex</code> , <code>int msgIndex</code> ,

	<p>Same as NetworkIpOutputQueueDequeuePacket(), except the packet is not actually dequeued. Note that the message containing the packet is not copied; the contents may (inadvertently or not) be directly modified. This function is called by MAC_OutputQueueTopPacket() (mac/mac.pc), which itself is called from mac/mac_802_11.pc and other MAC protocol source files. This function will assert false if the scheduler cannot return an IP packet for whatever reason.</p>
NetworkIpOutputQueueTopPacketForAPriority	<p>BOOL NetworkIpOutputQueueTopPacketForAPriority (Node* node, int interfaceIndex, QueuePriorityType priority, Message** msg, NodeAddress* nextHopAddress, int posInQueue)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • interfaceIndex - index to interface . • msgIndex - index to message . • msg - Storage for pointer to message • nextHopAddress - Storage for Packet's next hop addr. • priority - Storage for priority <p>Returns:</p> <ul style="list-style-type: none"> • BOOL - TRUE if there is a packet, FALSE otherwise.
NetworkIpInputQueueIsEmpty	<p>BOOL NetworkIpInputQueueIsEmpty (Node* node, int incomingInterface)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • incomingInterface - Index of interface. <p>Returns:</p> <ul style="list-style-type: none"> • BOOL - TRUE if the scheduler says the interface's input queue is empty. FALSE if the scheduler says the interface's input queue is not empty.
NetworkIpOutputQueueIsEmpty	<p>BOOL NetworkIpOutputQueueIsEmpty (Node* node, int interfaceIndex)</p>

	<p>Calls the packet scheduler for an interface to determine whether the interface's output queue is empty.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node. • <code>interfaceIndex</code> - Index of interface. <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - TRUE if the scheduler says the interface's output queue is empty. FALSE if the scheduler says the interface's output queue is not empty.
NetworkIpOutputQueueNumberInQueue	<p>Calls the packet scheduler for an interface to determine how many packets are in a queue. There may be multiple queues on an interface, so the priority of the desired queue is also provided.</p> <p>int NetworkIpOutputQueueNumberInQueue (Node* node, int interfaceIndex, BOOL specificPriorityOnly, QueuePriorityType priority)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node. • <code>interfaceIndex</code> - Index of interface. • <code>specificPriorityOnly</code> - Should we only get the number of packets • <code>priority</code> - Priority of queue. <p>Returns:</p> <ul style="list-style-type: none"> • <code>int</code> - Number of packets in queue.
NetworkIpOutputQueueDropPacket	<p>Drop a packet from the queue.</p> <p>NodeAddress NetworkIpOutputQueueDropPacket (Node* node, int interfaceIndex, Message** msg)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node. • <code>interfaceIndex</code> - index to interface . • <code>msg</code> - Storage for pointer to message <p>Returns:</p> <ul style="list-style-type: none"> • <code>NodeAddress</code> - Next hop of dropped packet.
NetworkIpDeleteOutboundPacketsToANode	<p>Deletes all packets in the queue going (probably broken), to the specified next hop address. There is option to return all such packets back to the routing protocols. via the usual mechanism (callback).</p> <p>void NetworkIpDeleteOutboundPacketsToANode (Node* node, const NodeAddress nextHopAddress, const NodeAddress destinationAddress, const BOOL returnPacketsToRoutingProtocol)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node. • <code>nextHopAddress</code> - Next hop associated with • <code>destinationAddress</code> - destination associated with • <code>returnPacketsToRoutingProtocol</code> - Determine whether or not

	<p>Returns:</p> <ul style="list-style-type: none"> • void - None
GetQueueNumberFromPriority	<p>unsigned GetQueueNumberFromPriority (TosType userTos, int numQueues)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • userTos - user TOS. • numQueues - Number of queues. <p>Returns:</p> <ul style="list-style-type: none"> • unsigned - Index of the queue.
ReturnPriorityForPHB	<p>QueuePriorityType ReturnPriorityForPHB (Node* node, TosType tos)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • tos - TOS field <p>Returns:</p> <ul style="list-style-type: none"> • QueuePriorityType - priority queue
NetworkGetInterfaceAndNextHopFromForwardingTable	<p>void NetworkGetInterfaceAndNextHopFromForwardingTable (Node* node, NodeAddress destinationAddress, int* interfaceIndex, NodeAddress* nextHopAddress)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • destinationAddress - Destination IP address. • interfaceIndex - Storage for index of outgoing • nextHopAddress - Storage for next hop IP address. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
NetworkGetInterfaceAndNextHopFromForwardingTable	<p>void NetworkGetInterfaceAndNextHopFromForwardingTable (Node* node, int currentInterface, NodeAddress destinationAddress, int* interfaceIndex, NodeAddress* nextHopAddress)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • currentInterface - Current interface in use.

	<p><code>destinationAddress</code> - Destination IP address.</p> <ul style="list-style-type: none"> • <code>interfaceIndex</code> - Storage for index of outgoing • <code>nextHopAddress</code> - Storage for next hop IP address. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
NetworkGetInterfaceAndNextHopFromForwardingTable	<p>Do a lookup on the routing table with a destination IP address to obtain a route (index of an outgoing interface and a next hop Ip address).</p> <p><code>void NetworkGetInterfaceAndNextHopFromForwardingTable (Node* node, NodeAddress destinationAddress, int* interfaceIndex, NodeAddress* nextHopAddress, BOOL testType, NetworkRoutingProtocolType type)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node. • <code>destinationAddress</code> - Destination IP address. • <code>interfaceIndex</code> - Storage for index of outgoing • <code>nextHopAddress</code> - Storage for next hop IP address. • <code>testType</code> - Same protocol's routes if true • <code>type</code> - routing protocol type. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
NetworkGetInterfaceAndNextHopFromForwardingTable	<p>Do a lookup on the routing table with a destination IP address to obtain a route (index of an outgoing interface and a next hop Ip address).</p> <p><code>void NetworkGetInterfaceAndNextHopFromForwardingTable (Node* node, int operatingInterface, NodeAddress destinationAddress, int* interfaceIndex, NodeAddress* nextHopAddress, BOOL testType, NetworkRoutingProtocolType type)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node. • <code>operatingInterface</code> - interface currently being • <code>destinationAddress</code> - Destination IP address. • <code>interfaceIndex</code> - Storage for index of outgoing • <code>nextHopAddress</code> - Storage for next hop IP address. • <code>testType</code> - Same protocol's routes if true • <code>type</code> - routing protocol type. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None

NetworkIpGetInterfaceIndexForNextHop	<p>This function looks at the network address of each of a node's network interfaces. When nextHopAddress is matched to a network, the interface index corresponding to the network is returned. (used by NetworkUpdateForwardingTable() and ospfv2.pc)</p>	int NetworkIpGetInterfaceIndexForNextHop (Node* node, NodeAddress nextHopAddress) <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • nextHopAddress - Destination IP address. <p>Returns:</p> <ul style="list-style-type: none"> • int - Index of outgoing interface, if nextHopAddress is on a directly connected network. -1, otherwise
NetworkGetInterfaceIndexForDestAddress	<p>Get interface for the destination address.</p>	int NetworkGetInterfaceIndexForDestAddress (Node* node, NodeAddress destAddress) <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • destAddress - Destination IP address. <p>Returns:</p> <ul style="list-style-type: none"> • int - interface index associated with destination.
NetworkRoutingGetAdminDistance	<p>Get the administrative distance of a routing protocol. These values don't quite match those recommended by Cisco.</p>	NetworkRoutingAdminDistanceType NetworkRoutingGetAdminDistance (Node* node, NetworkRoutingProtocolType type) <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • type - Type value of routing protocol. <p>Returns:</p> <ul style="list-style-type: none"> • NetworkRoutingAdminDistanceType - The administrative distance of the routing protocol.
NetworkInitForwardingTable	<p>Initialize the IP fowarding table, allocate enough memory for number of rows.</p>	void NetworkInitForwardingTable (Node* node) <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
NetworkUpdateForwardingTable	<p>Update or add entry to IP routing table. Search the routing table for an entry with an exact match for destAddress, destAddressMask, and routing protocol. Update this entry with the specified</p>	void NetworkUpdateForwardingTable (Node* node, NodeAddress destAddress, NodeAddress destAddressMask, NodeAddress nextHopAddress, int outgoingInterfaceIndex, int cost, NetworkRoutingProtocolType type) <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node.

<p>nextHopAddress (the outgoing interface is automatically determined from the nextHopAddress -- see code). If no matching entry found, then add a new route.</p>	<ul style="list-style-type: none"> • destAddress - IP address of destination • destAddressMask - Netmask. • nextHopAddress - Next hop IP address. • outgoingInterfaceIndex - outgoing interface. • cost - Cost metric associated with • type - type value of <p>Returns:</p> <ul style="list-style-type: none"> • void - None
<p>NetworkRemoveForwardingTableEntry</p> <p>Remove single entries in the routing table</p>	<p>void NetworkRemoveForwardingTableEntry (Node* node, NodeAddress destAddress, NodeAddress destAddressMask, NodeAddress nextHopAddress, int outgoingInterfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • destAddress - IP address of destination • destAddressMask - Netmask. • nextHopAddress - Next hop IP address. • outgoingInterfaceIndex - outgoing interface. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
<p>NetworkEmptyForwardingTable</p> <p>Remove entries in the routing table corresponding to a given routing protocol.</p>	<p>void NetworkEmptyForwardingTable (Node* node, NetworkRoutingProtocolType type)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • type - Type of routing protocol whose <p>Returns:</p> <ul style="list-style-type: none"> • void - None
<p>NetworkPrintForwardingTable</p> <p>Display all entries in node's routing table.</p>	<p>void NetworkPrintForwardingTable (Node* node)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. <p>Returns:</p>

	<ul style="list-style-type: none"> • void - None
NetworkGetMetricForDestAddress	<p>int NetworkGetMetricForDestAddress (Node* node, NodeAddress destAddress)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • destAddress - destination to get cost metric from. <p>Returns:</p> <ul style="list-style-type: none"> • int - Cost metric associated with destination.
NetworkIpSetRouteUpdateEventFunction	<p>void NetworkIpSetRouteUpdateEventFunction (Node* node, NetworkRouteUpdateEventType routeUpdateFunctionPtr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • routeUpdateFunctionPtr - Route update <p>Returns:</p> <ul style="list-style-type: none"> • void - None
NetworkIpGetRouteUpdateEventFunction	<p>NetworkRouteUpdateEventType NetworkIpGetRouteUpdateEventFunction (Node* node)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. <p>Returns:</p> <ul style="list-style-type: none"> • NetworkRouteUpdateEventType - Route update callback function to set.
NetworkIpGetInterfaceAddress	<p>NodeAddress NetworkIpGetInterfaceAddress (Node* node, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to the node • interfaceIndex - Number of interface <p>Returns:</p> <ul style="list-style-type: none"> • NodeAddress - IP address associated with the interface
NetworkIpGetInterfaceName	<p>char* NetworkIpGetInterfaceName (Node* node, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to the node

	<ul style="list-style-type: none"> • <code>interfaceIndex</code> - Number of interface <p>Returns:</p> <ul style="list-style-type: none"> • <code>char*</code> - interface name
NetworkIpGetInterfaceNetworkAddress	<p>NodeAddress NetworkIpGetInterfaceNetworkAddress (<code>Node* node, int interfaceIndex</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to the node • <code>interfaceIndex</code> - Number of interface <p>Returns:</p> <ul style="list-style-type: none"> • <code>NodeAddress</code> - network address associated with interface
NetworkIpGetInterfaceSubnetMask	<p>NodeAddress NetworkIpGetInterfaceSubnetMask (<code>Node* node, int interfaceIndex</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to the node • <code>interfaceIndex</code> - Number of interface <p>Returns:</p> <ul style="list-style-type: none"> • <code>NodeAddress</code> - subnet mask of the specified interface
NetworkIpGetInterfaceNumHostBits	<p>int NetworkIpGetInterfaceNumHostBits (<code>Node* node, int interfaceIndex</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to the node • <code>interfaceIndex</code> - Number of interface <p>Returns:</p> <ul style="list-style-type: none"> • <code>int</code> - Number of host bits on the specified interface
NetworkIpGetInterfaceBroadcastAddress	<p>NodeAddress NetworkIpGetInterfaceBroadcastAddress (<code>Node* node, int interfaceIndex</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to the node • <code>interfaceIndex</code> - Number of interface <p>Returns:</p> <ul style="list-style-type: none"> • <code>NodeAddress</code> - Broadcast address of specified interface

IsOutgoingBroadcast	<p>Checks whether IP packet's destination address is broadcast</p> <p>BOOL IsOutgoingBroadcast (Node* node, NodeAddress destAddress, int* outgoingInterface, NodeAddress* outgoingBroadcastAddress)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to the node • destAddress - IP packet's destination IP address. • outgoingInterface - Outgoing interface index. • outgoingBroadcastAddress - Broadcast address <p>Returns:</p> <ul style="list-style-type: none"> • BOOL - Returns true if destination is broadcast address
NetworkIpIsMyIP	<p>In turn calls IsMyPacket()</p> <p>BOOL NetworkIpIsMyIP (Node* node, NodeAddress ipAddress)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to the node • ipAddress - An IP packet's destination IP address. <p>Returns:</p> <ul style="list-style-type: none"> • BOOL - Returns if it belongs to it.
NetworkIpConfigurationError	<p>Prints out the IP configuration error</p> <p>void NetworkIpConfigurationError (Node* node, char [] parameterName, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to the node • parameterName - Error message to print • interfaceIndex - interface number <p>Returns:</p> <ul style="list-style-type: none"> • void - None
NetworkPrintIpHeader	<p>To print the IP header</p> <p>void NetworkPrintIpHeader (Message* msg)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • msg - Pointer to Message <p>Returns:</p> <ul style="list-style-type: none"> • void - None
NetworkIpAddToMulticastGroupList	void NetworkIpAddToMulticastGroupList (Node* node, NodeAddress groupAddress)

<p>Add a specified node to a multicast group</p>	<p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to the node • <code>groupAddress</code> - address of multicast group <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
<p>NetworkIpRemoveFromMulticastGroupList</p> <p>To remove specified node from a multicast group</p>	<p>void NetworkIpRemoveFromMulticastGroupList (Node* node, NodeAddress groupAddress)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to the node • <code>groupAddress</code> - address of multicast group <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
<p>NetworkIpPrintMulticastGroupList</p> <p>To print the multicast group list</p>	<p>void NetworkIpPrintMulticastGroupList (Node* node)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to the node <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
<p>NetworkIpIsPartOfMulticastGroup</p> <p>check if a node is part of specified multicast group</p>	<p>BOOL NetworkIpIsPartOfMulticastGroup (Node* node, NodeAddress groupAddress)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to the node • <code>groupAddress</code> - group to check if node is part of <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - None
<p>NetworkIpJoinMulticastGroup</p> <p>To join a multicast group</p>	<p>void NetworkIpJoinMulticastGroup (Node* node, NodeAddress mcastAddr, clocktype delay)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to the node • <code>mcastAddr</code> - multicast group address • <code>delay</code> - delay after which to join <p>Returns:</p>

	<ul style="list-style-type: none"> • void - None
NetworkIpJoinMulticastGroup	<p>void NetworkIpJoinMulticastGroup (Node* node, NodeAddress mcastAddr, clocktype delay)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to the node • mcastAddr - multicast group address • delay - delay after which to join <p>Returns:</p> <ul style="list-style-type: none"> • void - None
To join a multicast group	
NetworkIpLeaveMulticastGroup	<p>void NetworkIpLeaveMulticastGroup (Node* node, NodeAddress mcastAddr, clocktype delay)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to the node • mcastAddr - multicast group address • delay - delay after which to leave <p>Returns:</p> <ul style="list-style-type: none"> • void - None
To leave a multicast group	
NetworkIpLeaveMulticastGroup	<p>void NetworkIpLeaveMulticastGroup (Node* node, NodeAddress mcastAddr, clocktype delay)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to the node • mcastAddr - multicast group address • delay - delay after which to leave <p>Returns:</p> <ul style="list-style-type: none"> • void - None
To leave a multicast group	
NetworkIpSetMulticastTimer	<p>void NetworkIpSetMulticastTimer (Node* node, long eventType, NodeAddress mcastAddr, clocktype delay)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to the node • eventType - the event type • mcastAddr - multicast group address
To set a multicast timer to join or leave multicast groups	

	<ul style="list-style-type: none"> • <i>delay</i> - delay after which to leave <p>Returns:</p> <ul style="list-style-type: none"> • <i>void</i> - None
NetworkIpSetMulticastRoutingProtocol	<p>Assign a multicast routing protocol to an interface</p> <p>void NetworkIpSetMulticastRoutingProtocol (<i>Node*</i> node, <i>void*</i> multicastRoutingProtocol, <i>int</i> interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <i>node</i> - Pointer to the node • <i>multicastRoutingProtocol</i> - multicast routing protocol • <i>interfaceIndex</i> - interface number <p>Returns:</p> <ul style="list-style-type: none"> • <i>void</i> - None
NetworkIpGetMulticastRoutingProtocol	<p>To get the Multicast Routing Protocol structure</p> <p>void * NetworkIpGetMulticastRoutingProtocol (<i>Node*</i> node, <i>NetworkRoutingProtocolType</i> routingProtocolType)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <i>node</i> - Pointer to the node • <i>routingProtocolType</i> - routing protocol name <p>Returns:</p> <ul style="list-style-type: none"> • <i>void *</i> - None
NetworkIpAddMulticastProtocolType	<p>Assign a multicast protocol type to an interface</p> <p>void NetworkIpAddMulticastProtocolType (<i>Node*</i> node, <i>NetworkRoutingProtocolType</i> multicastProtocolType, <i>int</i> interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <i>node</i> - Pointer to this node • <i>multicastProtocolType</i> - routing protocol • <i>interfaceIndex</i> - interface number of the node <p>Returns:</p> <ul style="list-style-type: none"> • <i>void</i> - None
NetworkIpSetMulticastRouterFunction	<p>Set a multicast router function to an interface</p> <p>void NetworkIpSetMulticastRouterFunction (<i>Node*</i> node, <i>MulticastRouterFunctionType</i> routerFunctionPtr, <i>int</i> interfaceIndex)</p> <p>Parameters:</p>

	<ul style="list-style-type: none"> • node - Pointer to this node • routerFunctionPtr - router Func pointer • interfaceIndex - interface number of the node <p>Returns:</p> <ul style="list-style-type: none"> • void - None
NetworkIpGetMulticastRouterFunction	<p>MulticastRouterFunctionType NetworkIpGetMulticastRouterFunction (Node* node, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to this node • interfaceIndex - interface number of the node <p>Returns:</p> <ul style="list-style-type: none"> • MulticastRouterFunctionType - Multicast router function on this interface.
NetworkIpUpdateMulticastRoutingProtocolAndRouterFunction	<p>void NetworkIpUpdateMulticastRoutingProtocolAndRouterFunction (Node* node, NetworkRoutingProtocolType routingProtocolType, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - this node • routingProtocolType - multicast routing • interfaceIndex - interface index. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
NetworkIpUpdateUnicastRoutingProtocolAndRouterFunction	<p>void NetworkIpUpdateUnicastRoutingProtocolAndRouterFunction (Node* node, NetworkRoutingProtocolType routingProtocolType, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - this node • routingProtocolType - unicast routing • interfaceIndex - interface associated with unicast protocol. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
NetworkIpGetInterfaceIndexFromAddress	<p>int NetworkIpGetInterfaceIndexFromAddress (Node* node, NodeAddress address)</p> <p>Parameters:</p>

<p>Get the interface index from an IP address.</p>	<ul style="list-style-type: none"> • node - this node • address - address to determine interface index for <p>Returns:</p> <ul style="list-style-type: none"> • int - interface index associated with specified address.
<p>NetworkIpGetInterfaceIndexFromSubnetAddress</p> <p>Get the interface index from an IP subnet address.</p>	<p>int NetworkIpGetInterfaceIndexFromSubnetAddress (Node* node, NodeAddress address)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - this node • address - subnet address to determine interface <p>Returns:</p> <ul style="list-style-type: none"> • int - interface index associated with specified subnet address.
<p>NetworkIpIsMulticastAddress</p> <p>Check if an address is a multicast address.</p>	<p>BOOL NetworkIpIsMulticastAddress (Node* node, NodeAddress address)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - this node • address - address to determine if multicast address. <p>Returns:</p> <ul style="list-style-type: none"> • BOOL - TRUE if address is multicast address, FALSE, otherwise.
<p>NetworkInitMulticastForwardingTable</p> <p>initialize the multicast fowarding table, allocate enough memory for number of rows, used by ip</p>	<p>void NetworkInitMulticastForwardingTable (Node* node)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - this node <p>Returns:</p> <ul style="list-style-type: none"> • void - None
<p>NetworkEmptyMulticastForwardingTable</p> <p>empty out all the entries in the multicast forwarding table. basically set the size of table back to 0.</p>	<p>void NetworkEmptyMulticastForwardingTable (Node* node)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - this node <p>Returns:</p> <ul style="list-style-type: none"> • void - None
<p>NetworkGetOutgoingInterfaceFromMulticastForwardingTable</p>	<p>LinkedList* NetworkGetOutgoingInterfaceFromMulticastForwardingTable (Node* node, NodeAddress sourceAddress, NodeAddress groupAddress)</p>

	<p>get the interface Id node that lead to the (source, multicast group) pair.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - its own node • <code>sourceAddress</code> - multicast source address • <code>groupAddress</code> - multicast group <p>Returns:</p> <ul style="list-style-type: none"> • <code>LinkedList*</code> - interface Id from node to (source, multicast group), or <code>NETWORK_UNREACHABLE</code> (no such entry is found)
NetworkUpdateMulticastForwardingTable <p>update entry with(<code>sourceAddress,multicastGroupAddress</code>) pair. search for the row with(<code>sourceAddress,multicastGroupAddress</code>) and update its interface.</p>	<p>void NetworkUpdateMulticastForwardingTable (<code>Node* node, NodeAddress sourceAddress, NodeAddress multicastGroupAddress, int interfaceIndex</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - its own node • <code>sourceAddress</code> - multicast source • <code>multicastGroupAddress</code> - multicast group • <code>interfaceIndex</code> - interface to use for <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
NetworkPrintMulticastForwardingTable <p>display all entries in multicast forwarding table of the node.</p>	<p>void NetworkPrintMulticastForwardingTable (<code>Node* node</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - this node <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
NetworkPrintMulticastOutgoingInterface <p>Print multicast outgoing interfaces.</p>	<p>void NetworkPrintMulticastOutgoingInterface (<code>Node* node, list* list</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - this node • <code>list</code> - list of outgoing interfaces. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
NetworkInMulticastOutgoingInterface	<p>BOOL NetworkInMulticastOutgoingInterface (<code>Node* node, List* list, int interfaceIndex</code>)</p>

Determine if interface is in multicast outgoing interface list.	<p>Parameters:</p> <ul style="list-style-type: none"> • node - this node • list - list of outgoing interfaces. • interfaceIndex - interface to determine if in outgoing <p>Returns:</p> <ul style="list-style-type: none"> • BOOL - TRUE if interface is in multicast outgoing interface list, FALSE otherwise.
NetworkIpPrintTraceXML	<p>void NetworkIpPrintTraceXML (Node* node, Message* msg)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - this node • msg - Packet to print headers from. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
RouteThePacketUsingLookupTable	<p>void RouteThePacketUsingLookupTable (Node* node, Message* msg, int incomingInterface)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - this node • msg - Pointer to message with IP packet. • incomingInterface - incoming interface of packet <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
GetNetworkIPFragUnit	<p>int GetNetworkIPFragUnit (Node* node, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - this node • interfaceIndex - interface of node <p>Returns:</p> <ul style="list-style-type: none"> • int - None
NetworkIpUserProtocolInit	<p>void NetworkIpUserProtocolInit (Node* node, const NodeInput* nodeInput, const char* routingProtocolString, NetworkRoutingProtocolType* routingProtocolType, void** routingProtocolData)</p> <p>Parameters:</p>

	<ul style="list-style-type: none"> • <code>node</code> - this node • <code>nodeInput</code> - Provides access to • <code>routingProtocolString</code> - routing protocol • <code>routingProtocolType</code> - routing protocol • <code>routingProtocolData</code> - Access to routing protocol data <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
NetworkIpUserHandleProtocolEvent	<p>void NetworkIpUserHandleProtocolEvent (Node* node, Message* msg)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - The node that is handling the event. • <code>msg</code> - the event that is being handled <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
NetworkIpUserHandleProtocolPacket	<p>void NetworkIpUserHandleProtocolPacket (Node* node, Message* msg, unsigned char ipProtocol, NodeAddress sourceAddress, NodeAddress destinationAddress, int ttl)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - this node • <code>msg</code> - message that is being received. • <code>ipProtocol</code> - ip protocol • <code>sourceAddress</code> - source address • <code>destinationAddress</code> - destination address • <code>ttl</code> - time to live <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
NetworkIpUserProtocolFinalize	<p>void NetworkIpUserProtocolFinalize (Node* node, int userProtocolNumber)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - this node • <code>userProtocolNumber</code> - protocol number

	<p>Returns:</p> <ul style="list-style-type: none"> • void - None
Atm_RouteThePacketUsingLookupTable	<p>void Atm_RouteThePacketUsingLookupTable (Node* node, NodeAddress* destAddr, int* outIntf, NodeAddress* nextHop)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - this node • destAddr - destination Address • outIntf - this node • nextHop - nextHop address <p>Returns:</p> <ul style="list-style-type: none"> • void - None
RouteThePacketUsingMulticastForwardingTable	<p>Tries to route the multicast packet using the multicast forwarding table.</p> <p>void RouteThePacketUsingMulticastForwardingTable (Node* node, Message* msg, int incomingInterface)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - this node • msg - Pointer to Message • incomingInterface - Incomming Interface <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL.
NETWORKIpRoutingInit	<p>Initialization function for network layer. Initializes IP.</p> <p>int NETWORKIpRoutingInit (Node * node, const NodeInput *nodeInput nodeInput)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • nodeInput - Pointer to node input. <p>Returns:</p> <ul style="list-style-type: none"> • int - None
NetworkIpGetBandwidth	<p>getting the bandwidth information</p> <p>Int64 NetworkIpGetBandwidth (Node* node, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - the node who's bandwidth is needed.

	<ul style="list-style-type: none"> • <code>interfaceIndex</code> - interface Index. <p>Returns:</p> <ul style="list-style-type: none"> • <code>Int64</code> - inverted bandwidth ASSUMPTION : Bandwidth read from interface is in from of bps unit. To invert the bandwidth we use the equation $10000000 / \text{bandwidth}$. Where bandwidth is in Kbps unit.
NetworkIpGetPropDelay	<p>clocktype NetworkIpGetPropDelay (Node* node, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - the node who's bandwidth is needed. • <code>interfaceIndex</code> - interface Index. <p>Returns:</p> <ul style="list-style-type: none"> • <code>clocktype</code> - propagation delay ASSUMPTION : Array is exactly 3-byte long.
NetworkIpInterfaceIsEnabled	<p>BOOL NetworkIpInterfaceIsEnabled (Node* node, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node structure pointer. • <code>interfaceIndex</code> - interface Index. <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - None
NetworkIpIsWiredNetwork	<p>BOOL NetworkIpIsWiredNetwork (Node* node, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node structure pointer. • <code>interfaceIndex</code> - interface Index. <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - None
NetworkIpIsPointToPointNetwork	<p>BOOL NetworkIpIsPointToPointNetwork (Node* node, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node structure pointer. • <code>interfaceIndex</code> - interface Index. <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - None

IsIPV4MulticastEnabledOnInterface	<p>BOOL IsIPV4MulticastEnabledOnInterface (Node* node, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node structure pointer. • interfaceIndex - interface Index. <p>Returns:</p> <ul style="list-style-type: none"> • BOOL - None
IsIPV4RoutingEnabledOnInterface	<p>BOOL IsIPV4RoutingEnabledOnInterface (Node* node, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node structure pointer. • interfaceIndex - interface Index. <p>Returns:</p> <ul style="list-style-type: none"> • BOOL - None
NetworkIpGetNetworkProtocolType	<p>NetworkProtocolType NetworkIpGetNetworkProtocolType (Node* node, NodeAddress nodeId)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node structure pointer. • nodeId - node id. <p>Returns:</p> <ul style="list-style-type: none"> • NetworkProtocolType - None
ResolveNetworkTypeFromSrcAndDestNodeId	<p>NetworkType ResolveNetworkTypeFromSrcAndDestNodeId (Node* node, NodeId sourceNodeId, NodeId destNodeId)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to the node. • sourceNodeId - Source node id. • destNodeId - Destination node id. <p>Returns:</p> <ul style="list-style-type: none"> • NetworkType - None
NetworkIpIsWiredBroadcastNetwork	<p>BOOL NetworkIpIsWiredBroadcastNetwork (Node* node, int interfaceIndex)</p> <p>Parameters:</p>

<p>Determines if an interface is a wired interface.</p>	<ul style="list-style-type: none"> • <code>node</code> - node structure pointer. • <code>interfaceIndex</code> - interface Index. <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - None
<p>FindTraceRouteOption</p> <p>Sets the IP header for the Traceroute option field , and returns a pointer to traceroute header.</p>	<p><code>ip_traceroute* FindTraceRouteOption (const IpHeaderType* ipHeader)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>ipHeader</code> - Pointer to an IP header. <p>Returns:</p> <ul style="list-style-type: none"> • <code>ip_traceroute*</code> - pointer to the header of the traceroute option field. NULL if no option fields, or the desired option field cannot be found.



Contact [QualNet Support](#) for questions pertaining to the QualNet API Reference. This document is confidential and proprietary.
It may not be reproduced or distributed without the expressed written consent of [Scalable Network Technologies](#).

[QualNet](#)® is a Registered Trademark of [Scalable Network Technologies](#).

Copyright © 2001-2012 [Scalable Network Technologies, Inc.](#) All rights reserved.

QualNet 6.1 API Reference

IPv6

Data structures and parameters used in network layer are defined here.

Constant / Data Structure Summary

Type	Name
CONSTANT	MAX_KEY_LEN
	Maximum Key length of ipv6 address.
CONSTANT	MAX_PREFIX_LEN
	Maximum Prefix length of ipv6 address.
CONSTANT	CURR_HOP_LIMIT
	Current Hop limit a packet will traverse.
CONSTANT	IPV6_ADDR_LEN
	Ipv6 Address Length.
CONSTANT	IP6_NHDR_HOP
	Hop-by_hop IPv6 Next header field value.
CONSTANT	IP6_NHDR_RT
	Routing IPv6 Next header field value.
CONSTANT	IP6_NHDR_FRAG
	Fragment IPv6 Next header field value.
CONSTANT	IP6_NHDR_AUTH
	Authentication IPv6 Next header field value.
CONSTANT	IP6_NHDR_ESP

	Encryption IPv6 Next header field value. IP6_NHDR_IPCP
CONSTANT	Compression IPv6 Next header field value. IP6_NHDR_OSPF
CONSTANT	Compression IPv6 Next header field value. IP6_NHDR_DOPT
CONSTANT	Destination IPv6 Next header field value. IP6_NHDR_NONH
CONSTANT	No next header IPv6 Next header field value. IPV6_FLOWINFO_VERSION
CONSTANT	Flow information version. IPV6_VERSION
CONSTANT	IPv6 version no. IP6_MMU
CONSTANT	Minimal MTU and reassembly. IPPROTO_ICMPV6
CONSTANT	ICMPv6 protocol no. IP6ANY_ANYCAST
CONSTANT	IPv6 anycast. ND_DEFAULT_HOPLIM
CONSTANT	Node Discovery hop count. IP6_INSOPT_NOALLOC

	IPv6 insert option with no allocation.
CONSTANT	IP6_INSOPT_RAW
	IPv6 insert raw option.
CONSTANT	IP_FORWARDING
	IPv6 forwarding flag.
CONSTANT	IP6F_RESERVED_MASK
	Reserved fragment flag.
CONSTANT	IP_DF
	Don't fragment flag.
CONSTANT	IP6F_MORE_FRAG
	More fragments flag.
CONSTANT	IP6F_OFF_MASK
	Mask for fragmenting bits.
CONSTANT	IP6_FRAGTTL
	Time to live for frags.
CONSTANT	IP6_T_FLAG
	T Flag if set indicates transient multicast address.
CONSTANT	Multicast Address Scope Related constants.
CONSTANT	IP_FRAGMENT_HOLD_TIME
	IP Fragment hold time.
CONSTANT	IP_ROUTE_TO_IF
	IPv6 route to interface.

CONSTANT	IP_DEFAULT_MULTICAST_TTL	IPv6 route to interface.
CONSTANT	IPTTLDEC	TTL decrement.
CONSTANT	ENETUNREACH	Network unreachable.
CONSTANT	EHOSTUNREACH	Host unreachable.
CONSTANT	MAX_INITIAL_RTR_ADVERT_INTERVAL	Router Advertisement timer.
CONSTANT	MAX_INITIAL_RTR_ADVERTISEMENTS	Maximum Router Advertisement.
CONSTANT	MAX_RTR_ADVERT_INTERVAL	Maximum Router Advertisement timer.
CONSTANT	MIN_RTR_ADVERT_INTERVAL	Minimum Router Advertisement timer.
CONSTANT	RTR_SOLICITATION_INTERVAL	Router Solicitation timer.
CONSTANT	REACHABLE_TIME	reachable time
CONSTANT	UNREACHABLE_TIME	unreachable time
CONSTANT	RETRANS_TIMER	

	retransmission timer
CONSTANT	MAX_NEIGHBOR_ADVERTISEMENT
	maximum neighbor advertisement
CONSTANT	MAX_RTR_SOLICITATIONS
	maximum Router Solicitations NOTE : Sending only one Solicitation; modify it once autoconfiguration supported.
CONSTANT	MAX_MULTICAST_SOLICIT
	maximum multicast solicitation
CONSTANT	MAX_UNICAST_SOLICIT
	maximum unicast solicitation
CONSTANT	PKT_EXPIRE_DURATION
	Packet expiration interval
CONSTANT	INVALID_LINK_ADDR
	Invalid Link Layer Address
CONSTANT	MAX_HASHTABLE_SIZE
	Maximum size of Hash-Table
CONSTANT	MAX_REVLOOKUP_SIZE
	Maximum Rev Look up hash table size
CONSTANT	IP6_LSRRT
	type 0
CONSTANT	IP6_NIMRT
	type 1
CONSTANT	IP6_RT_MAX

	Maximum number of addresses.
CONSTANT	IP6ANY_HOST_PROXY proxy (host)
CONSTANT	IP6ANY_ROUTER_PROXY proxy (router)
STRUCT	ip6_hdr_struct QualNet typedefs struct ip6_hdr_struct to ip6_hdr. struct ip6_hdr_struct is 40 bytes, just like in the BSD code.
STRUCT	in6_multi_struct QualNet typedefs struct in6_multi_struct to in6_multi. struct in6_multi_struct is just like in the BSD code.
STRUCT	ipv6_h2hhdr_struct QualNet typedefs struct ipv6_h2hhdr_struct to ipv6_h2hhdr. struct ipv6_h2hhdr_struct is hop-by-Hop Options Header of 14 bytes, just like in the BSD code.
STRUCT	ipv6_rthdr_struct QualNet typedefs struct ipv6_rthdr_struct to ipv6_rthdr. struct ipv6_h2hhdr_struct is routing options header of 8 bytes, just like in the BSD code.
STRUCT	ipv6_rthdr_struct QualNet typedefs struct ipv6_rthdr_struct to ipv6_rthdr. struct ipv6_h2hhdr_struct is destination options header of 8 bytes, just like in the BSD code.
STRUCT	ip_moptions_struct QualNet typedefs struct ip_moptions_struct to ip_moptions. struct ip_moptions_struct is multicast option structure, just like in the BSD code.
STRUCT	ip6_frag_struct QualNet typedefs struct ip6_frag_struct to ipv6_fraghdr. struct ip6_frag_struct is fragmentation header structure.
STRUCT	ip6Stat_struct

	QualNet typedefs struct ip6stat_struct to ip6Stat. struct ip6stat_struct is statistic information structure.
STRUCT	Ipv6MulticastForwardingTableRow Structure of an entity of multicast forwarding table.
STRUCT	Ipv6MulticastForwardingTable Structure of multicast forwarding table
STRUCT	Ipv6MulticastGroupEntry Structure for Multicast Group Entry
STRUCT	IPv6InterfaceInfo QualNet typedefs struct ipv6_interface_struct to IPv6InterfaceInfo. struct ipv6_interface_struct is interface information structure.
STRUCT	messageBuffer QualNet typedefs struct messageBufferStruct to messageBuffer. struct messageBufferStruct is the buffer to hold messages when neighbour discovery is not done.
STRUCT	ip6q QualNet typedefs struct ip6q_struct to ip6q. struct ip6q is a simple queue to hold fragmented packets.
STRUCT	Ipv6FragQueue Ipv6 fragment queue structure.
STRUCT	FragmetedMsg QualNet typedefs struct fragmeted_msg_struct to ip6q. struct fragmeted_msg_struct is a simple fragmented packets msg hold structure.
STRUCT	defaultRouterList default router list structure.
STRUCT	destination_route_struct QualNet typedefs struct destination_route_struct to destinationRoute. struct destination_route_struct is destination information structure of a node.
STRUCT	DestinationCache

	Destination cache entry structure
STRUCT	Ipv6HashData Ipv6 hash data structure.
STRUCT	Ipv6HashBlockData Ipv6 hash block-data structure.
STRUCT	Ipv6HashBlock Ipv6 hash block structure.
STRUCT	Ipv6HashTable Ipv6 hash table structure
STRUCT	IPv6Data QualNet typedefs struct ipv6_data_struct to IPv6Data. struct ipv6_data_struct is ipv6 information structure of a node.
STRUCT	ndpNadvEvent QualNet typedefs struct ndp_event_struct to IPv6Data. struct ndp_event_struct is neighbor advertisement information structure.

Function / Macro Summary

Return Type	Summary
MACRO	ND_DEFAULT_CLASS(0xe0) Node Discovery sets class.
MACRO	NDP_DELAY NDP neighbor advertisement delay.
MACRO	IPV6JITTER_RANGE IPv6 jitter timer.
MACRO	IPV6_SET_CLASS(hdr, priority)

	Sets the flow class.
MACRO	IPV6_GET_CLASS(hdr)
	Gets the flow class.
void	ip6_hdrSetVersion() (UInt32 ipv6HdrVcf, UInt32 version)
	Set the value of version for ip6_hdr
void	ip6_hdrSetClass() (UInt32 ipv6HdrVcf, unsigned char ipv6Class)
	Set the value of class for ip6_hdr
void	ip6_hdrSetFlow() (UInt32 ipv6HdrVcf, UInt32 flow)
	Set the value of flow for ip6_hdr
UInt32	ip6_hdrGetVersion() (unsigned int ipv6HdrVcf)
	Returns the value of version for ip6_hdr
UInt32	ip6_hdrGetClass() (unsigned int ipv6HdrVcf)
	Returns the value of ip6_class for ip6_hdr
UInt32	ip6_hdrGetFlow() (unsigned int ipv6HdrVcf)
	Returns the value of ip6_flow for ip6_hdr
int	in6_isanycast (Node* node, in6_addr addr)
	Checks whether the address is anycast address of the node.
None	Ipv6AddIpv6Header (Node* node, Message* msg, in6_addr srcaddr, in6_addr dst_addr, TosType priority, unsigned char protocol, unsigned hlim)
	Add an IPv6 packet header to a message. Just calls AddIpHeader.
None	Ipv6AddFragmentHeader (Node *node node, Message *msg msg, unsigned char nextHeader, unsigned short offset, unsigned int id)
	Adds fragment header
None	(Node *node node, Message *msg msg, Address* sourceAddress, Address*

	Ipv6RemoveIpv6Header destinationAddress destinationAddress, TosType *priority priority, unsigned char *protocol protocol, unsigned *hLim hLim)	Removes Ipv6 header
None	Ipv6PreInit (Node* node)	IPv6 Pre Initialization.
None	IPv6Init (Node* node, const NodeInput* nodeInput)	IPv6 Initialization.
BOOL	Ipv6IsMyPacket (Node* node, in6_addr* dst_addr)	Checks whether the packet is the nodes packet. if the packet is of the node then returns TRUE, otherwise FALSE.
BOOL	Ipv6IsAddressInNetwork (const in6_addr* globalAddr, unsigned int tla, unsigned int vla, unsigned int sla)	Checks whether the address is in the same network. : if in the same network then returns TRUE, otherwise FALSE.
NodeAddress	Ipv6GetLinkLayerAddress (Node* node, int interfaceId, char* ll_addr_str)	Returns 32 bit link layer address of the interface.
None	Ipv6AddNewInterface (Node* node, in6_addr* globalAddr, unsigned int tla, unsigned int nla, unsigned int sla, int* newinterfaceIndex, const NodeInput* nodeInput)	Adds an ipv6 interface to the node.
BOOL	Ipv6IsForwardingEnabled (IPv6Data* ipv6)	Checks whether the node is forwarding enabled.
None	Ipv6Layer (Node* node, Message* msg)	Handle IPv6 layer events, incoming messages and messages sent to itself (timers, etc.).
None	Ipv6Finalize (Node* node)	Finalize function for the IPv6 model. Finalize functions for all network-layer IPv6 protocols are called here.
int	Ipv6GetMTU (Node* node, int interfaceId)	

	Returns the maximum transmission unit of the interface.
int	Ipv6GetInterfaceIndexFromAddress (Node* node, in6_addr* dst)
	Returns interface index of the specified address.
None	Ipv6CpuQueueInsert (Node* node, Message* msg, NodeAddress nextHopAddress, in6_addr destinationAddress, int outgoingInterface, int networkType, BOOL* queueIsFull)
	Calls the cpu packet scheduler for an interface to retrieve an IPv6 packet from a queue associated with the interface. The dequeued packet, since it's already been routed, has an associated next-hop IPv6 address. The packet's priority value is also returned.
None	Ipv6InputQueueInsert (Node* node, int incomingInterface, Message* msg, NodeAddress nextHopAddress, in6_addr destinationAddress, int outgoingInterface, int networkType, BOOL* queueIsFull)
	Calls input packet scheduler for an interface to retrieve an IP packet from a queue associated with the interface. The dequeued packet, since it's already been routed, has an associated next-hop IPv6 address. The packet's priority value is also returned.
None	Ipv6OutputQueueInsert (Node* node, int interfaceIndex, Message* msg, NodeAddress nextHopAddress, NodeAddress destinationAddress, int networkType, BOOL* queueIsFull)
	Calls output packet scheduler for an interface to retrieve an IP packet from a queue associated with the interface. The dequeued packet, since it's already been routed, has an associated next-hop IPv6 address. The packet's priority value is also returned. Called by QueueUpIpFragmentForMacLayer().
None	QueueUpIpv6FragmentForMacLayer (Node* node, int interfaceIndex, Message* msg, NodeAddress nextHopAddress, NodeAddress destinationAddress, int networkType, BOOL* queueIsFull)
	Calls output packet scheduler for an interface to retrieve an IP packet from a queue associated with the interface. The dequeued packet, since it's already been routed, has an associated next-hop IPv6 address. The packet's priority value is also returned. Called by QueueUpIpFragmentForMacLayer().
None	Ipv6SendPacketOnInterface (Node* node, Message* msg, int incommingInterface, int outgoingInterface, NodeAddress nextHop)
	This function is called once the outgoing interface index and next hop address to which to route an IPv6 packet are known. This queues an IPv6 packet for delivery to the MAC layer. This functions calls QueueUpIpFragmentForMacLayer(). This function is used to initiate fragmentation if required,before calling the next function.
None	Ipv6SendOnBackplane (Node* node, Message* msg, int incommingInterface, int outgoingInterface, NodeAddress hopAddr)
	This function is called when the packet delivered through backplane delay. required,before calling the next function.
None	Ipv6SendRawMessage (Node* node, Message* msg, in6_addr sourceAddress, in6_addr destinationAddress, int outgoingInterface, TosType priority, unsigned char protocol, unsigned ttl)
	Called by NetworkIpReceivePacketFromTransportLayer() to send to send UDP datagrams using IPv6. This function adds an IPv6 header and calls RoutePacketAndSendToMac().

None	<code>Ipv6SendToUdp</code> (Node* node, Message* msg, TosType priority, Address sourceAddress, Address destinationAddress, int incomingInterfaceIndex)	Sends a UDP packet to UDP in the transport layer. The source IPv6 address, destination IPv6 address, and priority of the packet are also sent.
None	<code>Ipv6SendToTCP</code> (Node* node, Message* msg, TosType priority, Address sourceAddress, Address destinationAddress, int incomingInterfaceIndex)	Sends a TCP packet to UDP in the transport layer. The source IPv6 address, destination IPv6 address, and priority of the packet are also sent.
None	<code>Ipv6ReceivePacketFromMacLayer</code> (Node* node, Message* msg, NodeAddress previousHopNodeId, int interfaceIndex)	IPv6 received IPv6 packet from MAC layer. Determine whether the packet is to be delivered to this node, or needs to be forwarded.
BOOL	<code>Ipv6AddMessageInBuffer</code> (Node* node, Message* msg, in6_addr* nextHopAddr, int inCommingInterface)	Adds an ipv6 packet in message in the hold buffer
BOOL	<code>Ipv6DeleteMessageInBuffer</code> (Node* node, messageBuffer* mBuf)	Deletes an ipv6 packet in the hold buffer
void	<code>Ipv6DropMessageFromBuffer</code> (Node* node, messageBuffer* mBuf)	Drops an ipv6 packet from the hold buffer
NetworkType	<code>Ipv6GetAddressTypeFromString</code> (char* interfaceAddr)	Returns network type from string ip address.
IPv6 multicast address	<code>Ipv6GetInterfaceMulticastAddress</code> (Node* node, int interfaceIndex)	Get multicast address of this interface
None	<code>Ipv6SolicitationMulticastAddress</code> (in6_addr* dst_addr, in6_addr* target)	Copies multicast solicitation address.
None	<code>Ipv6AllRoutersMulticastAddress</code> (in6_addr* dst dst)	Function to assign all routers multicast address.
None	<code>IPv6GetLinkLocalAddress</code> (node, int interface, in6_addr* addr)	

	Gets ipv6 link local address of the interface in output parameter addr. <code>IPv6GetSiteLocalAddress</code> (node, int interface, in6_addr* addr)
None	Gets ipv6 site local address of the interface in output parameter addr. <code>IPv6GetSiteLocalAddress</code> (node, int interface, in6_addr* addr)
None	Gets ipv6 global agreeable address of the interface in output parameter addr. <code>Ipv6GetPrefix</code> (in6_addr* addr, in6_addr* prefix)
Prefix for this interface	Gets ipv6 prefix from address. <code>Ipv6GetPrefixFromInterfaceIndex</code> (Node* node, int interfaceIndex)
BOOL	Gets ipv6 prefix from address. <code>Ipv6OutputQueueIsEmpty</code> (Node *node node, int interfaceIndex)
None	Check weather output queue is empty <code>Ipv6RoutingStaticInit</code> (Node *node node, const NodeInput nodeInput, NetworkRoutingProtocolType type)
None	Ipv6 Static routing initialization function. <code>Ipv6RoutingStaticEntry</code> (Node *node node, char currentLine[] currentLine)
None	Static routing route entry function <code>Ipv6AddDestination</code> (Node* node node, route* ro ro)
None	Adds destination in the destination cache. <code>Ipv6DeleteDestination</code> (Node* node node)
int	Deletes destination from the destination cache. <code>Ipv6CheckForValidPacket</code> (Node* node node, SchedulerType* scheduler scheduler, unsigned int* pIndex pIndex)
None	Checks the packet's validity <code>Ipv6NdpProcessing</code> (Node* node node)

	Ipv6 Destination cache and neighbor cache : processing function <u>Ipv6UpdateForwardingTable</u> (Node* node node, in6_addr destPrefix destPrefix, in6_addr nextHopPrefix nextHopPrefix, int interfaceIndex, int metric metric)
None	Updates Ipv6 Forwarding Table <u>Ipv6EmptyForwardingTable</u> (Node* node node, NetworkRoutingProtocolType type type)
None	Empties Ipv6 Forwarding Table for a particular routing protocol entry <u>Ipv6PrintForwardingTable</u> (Node* node node)
Interface index associated with specified subnet address.	Prints the forwarding table. <u>Ipv6InterfaceIndexFromSubnetAddress</u> (Node* node node, in6_addr* address)
void	Get the interface index from an IPv6 subnet address. <u>Ipv6GetInterfaceAndNextHopFromForwardingTable</u> (Node* node node, in6_addr destAddr, int* interfaceIndex, in6_addr* nextHopAddr)
interface index associated with destination.	Do a lookup on the routing table with a destination IPv6 address to obtain an outgoing interface and a next hop Ipv6 address. <u>Ipv6GetInterfaceIndexForDestAddress</u> (Node* node node, in6_addr destAddr)
interface index associated with destination.	Get interface for the destination address. <u>Ipv6GetMetricForDestAddress</u> (Node* node node, in6_addr destAddr)
Interface index associated with destination if found,	Get the cost metric for a destination from the forwarding table. <u>Ipv6IpGetInterfaceIndexForNextHop</u> (Node* node node, in6_addr destAddr)
Ipv6RouterFunctionType	This function looks at the network address of each of a node's network interfaces. When nextHopAddress is matched to a network, the interface index corresponding to the network is returned. <u>Ipv6GetRouterFunction</u> (Node* node, int interfaceIndex)
void	Get the router function pointer. <u>Ipv6SendPacketToMacLayer</u> (Node* node node, Message* msg, in6_addr destAddr, in6_addr* nextHopAddr, int* interfaceIndex)
void	Used if IPv6 next hop address and outgoing interface is known. <u>Ipv6JoinMulticastGroup</u> (Node* node, in6_addr mcastAddr, clocktype delay)

	Join a multicast group.
void	<code>Ipv6AddToMulticastGroupList</code> (Node* node, in6_addr groupAddress)
	Add group to multicast group list.
void	<code>Ipv6LeaveMulticastGroup</code> (Node* node, in6_addr mcastAddr)
	Leave a multicast group.
void	<code>Ipv6RemoveFromMulticastGroupList</code> (Node* node, in6_addr groupAddress)
	Remove group from multicast group list.
void	<code>Ipv6NotificationOfPacketDrop</code> (Node* node, Message* msg, const NodeAddress nextHopAddress, int interfaceIndex)
	Invoke callback functions when a packet is dropped.
TRUE if node is part of multicast group,	<code>Ipv6IsPartOfMulticastGroup</code> (Node* node, Message* msg, in6_addr groupAddress)
	Check if destination is part of the multicast group.
TRUE if reserved multicast address, FALSE otherwise.	<code>Ipv6IsReservedMulticastAddress</code> (Node* node, in6_addr mcastAddr)
	Check if address is reserved multicast address.
TRUE if interface is in multicast outgoing interface	<code>Ipv6InMulticastOutgoingInterface</code> (Node* node, LinkedList* list, int interfaceIndex)
	Determine if interface is in multicast outgoing interface list.
void	<code>Ipv6UpdateMulticastForwardingTable</code> (Node* node, in6_addr sourceAddress, in6_addr multicastGroupAddress)
	update entry with (sourceAddress, multicastGroupAddress) pair. search for the row with (sourceAddress, multicastGroupAddress) and update its interface.
Interface List if match found, NULL otherwise.	<code>Ipv6GetOutgoingInterfaceFromMulticastTable</code> (Node* node, in6_addr sourceAddress, in6_addr groupAddress)
	get the interface List that lead to the (source, multicast group) pair.
void	<code>Ipv6CreateBroadcastAddress</code> ()
	Create IPv6 Broadcast Address (ff02 followed by all one).
Prefix Length.	<code>Ipv6GetPrefixLength</code> ()

	Get prefix length of an interface.
void	<p>Ipv6SetMacLayerStatusEventHandlerFunction(Node* node, Ipv6MacLayerStatusEventHandlerFunctionType StatusEventHandlerPtr, int interfaceIndex)</p> <p>Allows the MAC layer to send status messages (e.g., packet drop, link failure) to a network-layer routing protocol for routing optimization.</p>
void	<p>Ipv6DeleteOutboundPacketsToANode(Node* node, const in6_addr nextHopAddress, const in6_addr destinationAddress, const BOOL returnPacketsToRoutingProtocol)</p> <p>Deletes all packets in the queue going to the specified next hop address. There is option to return all such packets back to the routing protocols.</p>
void	<p>Ipv6IsLoopbackAddress(Node* node, in6_addr address)</p> <p>Check if address is self loopback address.</p>
TRUE if my Ip, FALSE otherwise.	<p>Ipv6IsMyIp(Node* node, in6_addr* dst_addr)</p> <p>Check if address is self loopback address.</p>
Scope value if valid multicast address, 0 otherwise.	<p>Ipv6IsValidGetMulticastScope(Node* node, in6_addr multiAddr)</p> <p>Check if multicast address has valid scope.</p>
BOOL	<p>IsIPV6RoutingEnabledOnInterface(Node* node, int interfaceIndex)</p> <p>To check if IPV6 Routing is enabled on interface?</p>

Constant / Data Structure Detail

Constant	MAX_KEY_LEN 128 Maximum Key length of ipv6 address.
Constant	MAX_PREFIX_LEN 64 Maximum Prefix length of ipv6 address.
Constant	CURR_HOP_LIMIT 255

	Current Hop limit a packet will traverse.
Constant	IPV6_ADDR_LEN 16 Ipv6 Address Length.
Constant	IP6_NHDR_HOP 0 Hop-by_hop IPv6 Next header field value.
Constant	IP6_NHDR_RT 43 Routing IPv6 Next header field value.
Constant	IP6_NHDR_FRAG 44 Fragment IPv6 Next header field value.
Constant	IP6_NHDR_AUTH 51 Authentication IPv6 Next header field value.
Constant	IP6_NHDR_ESP 50 Encryption IPv6 Next header field value.
Constant	IP6_NHDR_IPCP 108 Compression IPv6 Next header field value.
Constant	IP6_NHDR OSPF 89 Compression IPv6 Next header field value.
Constant	IP6_NHDR_DOPT 60 Destination IPv6 Next header field value.
Constant	IP6_NHDR_NONH 59

	No next header IPv6 Next header field value.
Constant	IPV6_FLOWINFO_VERSION 0x000000f0 Flow information version.
Constant	IPV6_VERSION 6 IPv6 version no.
Constant	IP6_MMTU 1280 Minimal MTU and reassembly.
Constant	IPPROTO_ICMPV6 58 ICMPv6 protocol no.
Constant	IP6ANY_ANYCAST 3 IPv6 anycast.
Constant	ND_DEFAULT_HOPLIM 255 Node Discovery hop count.
Constant	IP6_INSOPT_NOALLOC 1 IPv6 insert option with no allocation.
Constant	IP6_INSOPT_RAW 2 IPv6 insert raw option.
Constant	IP_FORWARDING 1 IPv6 forwarding flag.
Constant	IP6F_RESERVED_MASK 0x0600 Reserved fragment flag.

Constant	IP_DF 0x4000 Don't fragment flag.
Constant	IP6F_MORE_FRAG 0x01 More fragments flag.
Constant	IP6F_OFF_MASK 0xf8ff Mask for fragmenting bits.
Constant	IP6_FRAGTTL 120 Time to live for frags.
Constant	IP6_T_FLAG 0x10 T Flag if set indicates transient multicast address.
Constant	Multicast Address Scope Related constants.
Constant	IP_FRAGMENT_HOLD_TIME 60 * SECOND IP Fragment hold time.
Constant	IP_ROUTETOIF 4 IPv6 route to interface.
Constant	IP_DEFAULT_MULTICAST_TTL 255 IPv6 route to interface.
Constant	IPTTLDEC 1 TTL decrement.
Constant	ENETUNREACH 1

	Network unreachable.
Constant	EHOSTUNREACH 2 Host unreachable.
Constant	MAX_INITIAL_RTR_ADVERT_INTERVAL 16 * SECOND Router Advertisement timer.
Constant	MAX_INITIAL_RTR_ADVERTISEMENTS 3 Maximum Router Advertisement.
Constant	MAX_RTR_ADVERT_INTERVAL 600 * SECOND Maximum Router Advertisement timer.
Constant	MIN_RTR_ADVERT_INTERVAL MAX_RTR_ADVERT_INTERVAL * 0.33 Minimum Router Advertisement timer.
Constant	RTR_SOLICITATION_INTERVAL 4 * SECOND Router Solicitation timer.
Constant	REACHABLE_TIME (30 * SECOND) reachable time
Constant	UNREACHABLE_TIME (30 * SECOND) unreachable time
Constant	RETRANS_TIMER (2 * SECOND) retransmission timer
Constant	MAX_NEIGHBOR_ADVERTISEMENT 3 maximum neighbor advertisement
Constant	MAX_RTR_SOLICITATIONS 1

	maximum Router Solicitations NOTE : Sending only one Solicitation; modify it once autoconfiguration supported.
Constant	MAX_MULTICAST_SOLICIT 3 maximum multicast solicitation
Constant	MAX_UNICAST_SOLICIT 3 maximum unicast solicitation
Constant	PKT_EXPIRE_DURATION (3 * SECOND) Packet expiration interval
Constant	INVALID_LINK_ADDR -3 Invalid Link Layer Address
Constant	MAX_HASHTABLE_SIZE 4 Maximum size of Hash-Table
Constant	MAX_REVLOOKUP_SIZE 100 Maximum Rev Look up hash table size
Constant	IP6_LSRRT 0 type 0
Constant	IP6_NIMRT 1 type 1
Constant	IP6_RT_MAX 3 Maximum number of addresses.
Constant	IP6ANY_HOST_PROXY 1

	proxy (host)
Constant	IP6ANY_ROUTER_PROXY 2
	proxy (router)
Structure	ip6_hdr_struct
	QualNet typedefs struct ip6_hdr_struct to ip6_hdr. struct ip6_hdr_struct is 40 bytes, just like in the BSD code.
Structure	in6_multi_struct
	QualNet typedefs struct in6_multi_struct to in6_multi. struct in6_multi_struct is just like in the BSD code.
Structure	ipv6_h2hhdr_struct
	QualNet typedefs struct ipv6_h2hhdr_struct to ipv6_h2hhdr. struct ipv6_h2hhdr_struct is hop-by-Hop Options Header of 14 bytes, just like in the BSD code.
Structure	ipv6_rthdr_struct
	QualNet typedefs struct ipv6_rthdr_struct to ipv6_rthdr. struct ipv6_h2hhdr_struct is routing options header of 8 bytes, just like in the BSD code.
Structure	ipv6_rthdr_struct
	QualNet typedefs struct ipv6_rthdr_struct to ipv6_rthdr. struct ipv6_h2hhdr_struct is destination options header of 8 bytes, just like in the BSD code.
Structure	ip_moptions_struct
	QualNet typedefs struct ip_moptions_struct to ip_moptions. struct ip_moptions_struct is multicast option structure, just like in the BSD code.
Structure	ip6_frag_struct
	QualNet typedefs struct ip6_frag_struct to ipv6_fraghdr. struct ip6_frag_struct is fragmentation header structure.
Structure	ip6Stat_struct
	QualNet typedefs struct ip6stat_struct to ip6Stat. struct ip6stat_struct is statistic information structure.
Structure	Ipv6MulticastForwardingTableRow

	Structure of an entity of multicast forwarding table.
Structure	Ipv6MulticastForwardingTable Structure of multicast forwarding table
Structure	Ipv6MulticastGroupEntry Structure for Multicast Group Entry
Structure	IPv6InterfaceInfo QualNet typedefs struct ipv6_interface_struct to IPv6InterfaceInfo. struct ipv6_interface_struct is interface information structure.
Structure	messageBuffer QualNet typedefs struct messageBufferStruct to messageBuffer. struct messageBufferStruct is the buffer to hold messages when neighbour discovery is not done.
Structure	ip6q QualNet typedefs struct ip6q_struct to ip6q. struct ip6q is a simple queue to hold fragmented packets.
Structure	Ipv6FragQueue Ipv6 fragment queue structure.
Structure	FragmetedMsg QualNet typedefs struct fragmeted_msg_struct to ip6q. struct fragmeted_msg_struct is a simple fragmented packets msg hold structure.
Structure	defaultRouterList default router list structure.
Structure	destination_route_struct QualNet typedefs struct destination_route_struct to destinationRoute. struct destination_route_struct is destination information structure of a node.
Structure	DestinationCache

	Destination cache entry structure
Structure	Ipv6HashData Ipv6 hash data structure.
Structure	Ipv6HashBlockData Ipv6 hash block-data structure.
Structure	Ipv6HashBlock Ipv6 hash block structure.
Structure	Ipv6HashTable Ipv6 hash table structure
Structure	IPv6Data QualNet typedefs struct ipv6_data_struct to IPv6Data. struct ipv6_data_struct is ipv6 information structure of a node.
Structure	ndpNadvEvent QualNet typedefs struct ndp_event_struct to IPv6Data. struct ndp_event_struct is neighbor advertisement information structure.

Function / Macro Detail

Function / Macro	Format
ND_DEFAULT_CLASS(0xe0)	Node Discovery sets class.
NDP_DELAY	NDP neighbor advertisement delay.
IPV6JITTER_RANGE	IPv6 jitter timer.
IPV6_SET_CLASS(hdr, priority)	Sets the flow class.

IPV6_GET_CLASS(hdr)	Gets the flow class.
ip6_hdrSetVersion() Set the value of version for ip6_hdr	<p>void ip6_hdrSetVersion() (UInt32 ipv6HdrVcf, UInt32 version)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>ipv6HdrVcf</code> - The variable containing the value of ip6_v,ip6_class • <code>version</code> - Input value for set operation <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL.
ip6_hdrSetClass() Set the value of class for ip6_hdr	<p>void ip6_hdrSetClass() (UInt32 ipv6HdrVcf, unsigned char ipv6Class)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>ipv6HdrVcf</code> - The variable containing the value of ip6_v,ip6_class • <code>ipv6Class</code> - Input value for set operation <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL.
ip6_hdrSetFlow() Set the value of flow for ip6_hdr	<p>void ip6_hdrSetFlow() (UInt32 ipv6HdrVcf, UInt32 flow)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>ipv6HdrVcf</code> - The variable containing the value of ip6_v,ip6_class • <code>flow</code> - Input value for set operation <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL.
ip6_hdrGetVersion() Returns the value of version for ip6_hdr	<p>UInt32 ip6_hdrGetVersion() (unsigned int ipv6HdrVcf)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>ipv6HdrVcf</code> - The variable containing the value of ip6_v,ip6_class <p>Returns:</p> <ul style="list-style-type: none"> • <code>UInt32</code> - None
ip6_hdrGetClass() Returns the value of ip6_class for ip6_hdr	<p>UInt32 ip6_hdrGetClass() (unsigned int ipv6HdrVcf)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>ipv6HdrVcf</code> - The variable containing the value of ip6_v,ip6_class

	<p>Returns:</p> <ul style="list-style-type: none"> • UInt32 - None
ip6_hdrGetFlow()	<p>UInt32 ip6_hdrGetFlow() (unsigned int ipv6HdrVcf)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>ipv6HdrVcf</code> - The variable containing the value of ip6_v,ip6_class <p>Returns:</p> <ul style="list-style-type: none"> • UInt32 - None
in6_isanycast	<p>int in6_isanycast (Node* node, in6_addr addr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node structure. • <code>addr</code> - ipv6 address. <p>Returns:</p> <ul style="list-style-type: none"> • int - None
Ipv6AddIpv6Header	<p>None Ipv6AddIpv6Header (Node* node, Message* msg, in6_addr srcaddr, in6_addr dst_addr, TosType priority, unsigned char protocol, unsigned hlim)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node. • <code>msg</code> - Pointer to message. • <code>srcaddr</code> - Source IPv6 address. • <code>dst_addr</code> - Destination IPv6 address. • <code>priority</code> - Current type of service • <code>protocol</code> - IPv6 protocol number. • <code>hlim</code> - Hop limit. <p>Returns:</p> <ul style="list-style-type: none"> • None - None
Ipv6AddFragmentHeader	<p>None Ipv6AddFragmentHeader (Node *node node, Message *msg msg, unsigned char nextHeader, unsigned short offset, unsigned int id)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node

	<ul style="list-style-type: none"> • <code>msg</code> - Pointer to Message • <code>nextHeader</code> - <code>nextHeader</code> • <code>offset</code> - <code>offset</code> • <code>id</code> - <code>id</code> <p>Returns:</p> <ul style="list-style-type: none"> • <code>None</code> - <code>None</code>
Ipv6RemoveIpv6Header	<p>None Ipv6RemoveIpv6Header (<code>Node *node node</code>, <code>Message *msg msg</code>, <code>Address* sourceAddress</code>, <code>Address* destinationAddress destinationAddress</code>, <code>TosType *priority priority</code>, <code>unsigned char *protocol protocol</code>, <code>unsigned *hLim hLim</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node • <code>msg</code> - Pointer to message • <code>sourceAddress</code> - Poineter Source address • <code>destinationAddress</code> - Destination address • <code>priority</code> - Priority • <code>protocol</code> - protocol • <code>hLim</code> - <code>hLim</code> <p>Returns:</p> <ul style="list-style-type: none"> • <code>None</code> - <code>None</code>
Ipv6PreInit	<p>None Ipv6PreInit (<code>Node* node</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node structure. <p>Returns:</p> <ul style="list-style-type: none"> • <code>None</code> - <code>None</code>
IPv6Init	<p>None IPv6Init (<code>Node* node</code>, <code>const NodeInput* nodeInput</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node structure. • <code>nodeInput</code> - Node input.

	<p>Returns:</p> <ul style="list-style-type: none"> • None - None
Ipv6IsMyPacket	<p>BOOL Ipv6IsMyPacket (Node* node, in6_addr* dst_addr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node structure. • dst_addr - ipv6 packet destination address. <p>Returns:</p> <ul style="list-style-type: none"> • BOOL - None
Ipv6IsAddressInNetwork	<p>BOOL Ipv6IsAddressInNetwork (const in6_addr* globalAddr, unsigned int tla, unsigned int vla, unsigned int sla)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • globalAddr - Pointer to ipv6 address. • tla - Top level ipv6 address. • vla - Next level ipv6 address. • sla - Site local ipv6 address. <p>Returns:</p> <ul style="list-style-type: none"> • BOOL - None
Ipv6GetLinkLayerAddress	<p>NodeAddress Ipv6GetLinkLayerAddress (Node* node, int interfaceId, char* ll_addr_str)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node structure. • interfaceId - Interface Id. • ll_addr_str - Pointer to character link layer <p>Returns:</p> <ul style="list-style-type: none"> • NodeAddress - None
Ipv6AddNewInterface	<p>None Ipv6AddNewInterface (Node* node, in6_addr* globalAddr, unsigned int tla, unsigned int nla, unsigned int sla, int* newinterfaceIndex, const NodeInput* nodeInput)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node structure. • globalAddr - Global ipv6 address pointer.

	<ul style="list-style-type: none"> • <code>tla</code> - Top level id. • <code>nla</code> - Next level id. • <code>sla</code> - Site level id. • <code>newInterfaceIndex</code> - Pointer to new interface index. • <code>nodeInput</code> - Node Input. <p>Returns:</p> <ul style="list-style-type: none"> • <code>None</code> - None
Ipv6IsForwardingEnabled	<p>BOOL Ipv6IsForwardingEnabled (IPv6Data* <code>ipv6</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>ipv6</code> - Pointer to ipv6 data structure. <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - None
Ipv6Layer	<p>None Ipv6Layer (Node* <code>node</code>, Message* <code>msg</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node. • <code>msg</code> - Pointer to message. <p>Returns:</p> <ul style="list-style-type: none"> • <code>None</code> - None
Ipv6Finalize	<p>None Ipv6Finalize (Node* <code>node</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node. <p>Returns:</p> <ul style="list-style-type: none"> • <code>None</code> - None
Ipv6GetMTU	<p>int Ipv6GetMTU (Node* <code>node</code>, int <code>interfaceId</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node. • <code>interfaceId</code> - Interface Id. <p>Returns:</p>

	<ul style="list-style-type: none"> • int - None
Ipv6GetInterfaceIndexFromAddress	<p>int Ipv6GetInterfaceIndexFromAddress (Node* node, in6_addr* dst)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • dst - IPv6 address. <p>Returns:</p> <ul style="list-style-type: none"> • int - None
Ipv6CpuQueueInsert	<p>None Ipv6CpuQueueInsert (Node* node, Message* msg, NodeAddress nextHopAddress, in6_addr destinationAddress, int outgoingInterface, int networkType, BOOL* queueIsFull)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • msg - Pointer to message with IPv6 packet. • nextHopAddress - Packet's next hop link layer address. • destinationAddress - Packet's destination address. • outgoingInterface - Used to determine where packet • networkType - Type of network packet is using (IPv6, • queueIsFull - Storage for boolean indicator. <p>Returns:</p> <ul style="list-style-type: none"> • None - None
Ipv6InputQueueInsert	<p>None Ipv6InputQueueInsert (Node* node, int incomingInterface, Message* msg, NodeAddress nextHopAddress, in6_addr destinationAddress, int outgoingInterface, int networkType, BOOL* queueIsFull)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • incomingInterface - interface of input queue. • msg - Pointer to message with IPv6 packet. • nextHopAddress - Packet's next hop link layer address. • destinationAddress - Packet's destination address. • outgoingInterface - Used to determine where packet

	<ul style="list-style-type: none"> • <code>networkType</code> - Type of network packet is using (IPv6, • <code>queueIsFull</code> - Storage for boolean indicator. <p>Returns:</p> <ul style="list-style-type: none"> • <code>None</code> - None
Ipv6OutputQueueInsert	<p>None Ipv6OutputQueueInsert (<code>Node* node, int interfaceIndex, Message* msg, NodeAddress nextHopAddress, NodeAddress destinationAddress, int networkType, BOOL* queueIsFull</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node. • <code>interfaceIndex</code> - interface of input queue. • <code>msg</code> - Pointer to message with IPv6 packet. • <code>nextHopAddress</code> - Packet's next link layer hop address. • <code>destinationAddress</code> - Packet's destination address. • <code>networkType</code> - Type of network packet is using (IPv6, • <code>queueIsFull</code> - Storage for boolean indicator. <p>Returns:</p> <ul style="list-style-type: none"> • <code>None</code> - None
QueueUpIpv6FragmentForMacLayer	<p>None QueueUpIpv6FragmentForMacLayer (<code>Node* node, int interfaceIndex, Message* msg, NodeAddress nextHopAddress, NodeAddress destinationAddress, int networkType, BOOL* queueIsFull</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node. • <code>interfaceIndex</code> - interface of input queue. • <code>msg</code> - Pointer to message with IPv6 packet. • <code>nextHopAddress</code> - Packet's next hop address. • <code>destinationAddress</code> - Packet's destination address. • <code>networkType</code> - Type of network packet is using (IPv6, • <code>queueIsFull</code> - Storage for boolean indicator. <p>Returns:</p> <ul style="list-style-type: none"> • <code>None</code> - None
Ipv6SendPacketOnInterface	<p>None Ipv6SendPacketOnInterface (<code>Node* node, Message* msg, int incomingInterface, int outgoingInterface,</code></p>

	<p>This function is called once the outgoing interface index and next hop address to which to route an IPv6 packet are known. This queues an IPv6 packet for delivery to the MAC layer. This functions calls QueueUpIpFragmentForMacLayer(). This function is used to initiate fragmentation if required, before calling the next function.</p>
Ipv6SendOnBackplane	<p>This function is called when the packet delivered through backplane delay. required, before calling the next function.</p> <p>None <code>Ipv6SendOnBackplane</code> (Node* node, Message* msg, int incomingInterface, int outgoingInterface, NodeAddress hopAddr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node. • <code>msg</code> - Pointer to message with ip packet. • <code>incomingInterface</code> - Index of incoming interface. • <code>outgoingInterface</code> - Index of outgoing interface. • <code>hopAddr</code> - Next hop link layer address. <p>Returns:</p> <ul style="list-style-type: none"> • <code>None</code> - None
Ipv6SendRawMessage	<p>Called by NetworkIpReceivePacketFromTransportLayer() to send to send UDP datagrams using IPv6. This function adds an IPv6 header and calls RoutePacketAndSendToMac().</p> <p>None <code>Ipv6SendRawMessage</code> (Node* node, Message* msg, in6_addr sourceAddress, in6_addr destinationAddress, int outgoingInterface, TosType priority, unsigned char protocol, unsigned ttl)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node. • <code>msg</code> - Pointer to message with payload data • <code>sourceAddress</code> - Source IPv6 address. • <code>destinationAddress</code> - Destination IPv6 address. • <code>outgoingInterface</code> - outgoing interface to use to • <code>priority</code> - Priority of packet. • <code>protocol</code> - IPv6 protocol number.

	<ul style="list-style-type: none"> • <code>ttl</code> - Time to live. <p>Returns:</p> <ul style="list-style-type: none"> • <code>None</code> - None
Ipv6SendToUdp	<p>Sends a UDP packet to UDP in the transport layer. The source IPv6 address, destination IPv6 address, and priority of the packet are also sent.</p> <p>None Ipv6SendToUdp (<code>Node* node, Message* msg, TosType priority, Address sourceAddress, Address destinationAddress, int incomingInterfaceIndex</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node. • <code>msg</code> - Pointer to message with UDP packet. • <code>priority</code> - Priority of UDP • <code>sourceAddress</code> - Source IP address info. • <code>destinationAddress</code> - Destination IP address info. • <code>incomingInterfaceIndex</code> - interface that received the packet <p>Returns:</p> <ul style="list-style-type: none"> • <code>None</code> - None
Ipv6SendToTCP	<p>Sends a TCP packet to UDP in the transport layer. The source IPv6 address, destination IPv6 address, and priority of the packet are also sent.</p> <p>None Ipv6SendToTCP (<code>Node* node, Message* msg, TosType priority, Address sourceAddress, Address destinationAddress, int incomingInterfaceIndex</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node. • <code>msg</code> - Pointer to message with UDP packet. • <code>priority</code> - Priority of TCP • <code>sourceAddress</code> - Source IP address info. • <code>destinationAddress</code> - Destination IP address info. • <code>incomingInterfaceIndex</code> - interface that received the packet <p>Returns:</p> <ul style="list-style-type: none"> • <code>None</code> - None
Ipv6ReceivePacketFromMacLayer	<p>IPv6 received IPv6 packet from MAC layer. Determine whether the packet is to be delivered to this node, or</p> <p>None Ipv6ReceivePacketFromMacLayer (<code>Node* node, Message* msg, NodeAddress previousHopNodeId, int interfaceIndex</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node.

<p>needs to be forwarded.</p>	<ul style="list-style-type: none"> • <code>msg</code> - Pointer to message with ip packet. • <code>previousHopNodeId</code> - nodeId of the previous hop. • <code>interfaceIndex</code> - Index of interface on which packet arrived. <p>Returns:</p> <ul style="list-style-type: none"> • <code>None</code> - None
<p>Ipv6AddMessageInBuffer</p> <p>Adds an ipv6 packet in message in the hold buffer</p>	<p>BOOL Ipv6AddMessageInBuffer (<code>Node* node, Message* msg, in6_addr* nextHopAddr, int inCommingInterface</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node structure. • <code>msg</code> - Pointer to message with ip packet. • <code>nextHopAddr</code> - Source IPv6 address. • <code>inCommingInterface</code> - Incoming interface <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - None
<p>Ipv6DeleteMessageInBuffer</p> <p>Deletes an ipv6 packet in the hold buffer</p>	<p>BOOL Ipv6DeleteMessageInBuffer (<code>Node* node, messageBuffer* mBuf</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node structure. • <code>mBuf</code> - Pointer to messageBuffer tail. <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - None
<p>Ipv6DropMessageFromBuffer</p> <p>Drops an ipv6 packet from the hold buffer</p>	<p>void Ipv6DropMessageFromBuffer (<code>Node* node, messageBuffer* mBuf</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node structure. • <code>mBuf</code> - Pointer to messageBuffer tail. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
<p>Ipv6GetAddressTypeFromString</p>	<p><code>NetworkType Ipv6GetAddressTypeFromString (char* interfaceAddr)</code></p> <p>Parameters:</p>

Returns network type from string ip address.	<ul style="list-style-type: none"> • <code>interfaceAddr</code> - Character Pointer to ip address. <p>Returns:</p> <ul style="list-style-type: none"> • <code>NetworkType</code> - None
Ipv6GetInterfaceMulticastAddress Get multicast address of this interface	<p>IPv6 multicast address Ipv6GetInterfaceMulticastAddress (<code>Node* node, int interfaceIndex</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Node pointer • <code>interfaceIndex</code> - interface for which multicast is required <p>Returns:</p> <ul style="list-style-type: none"> • IPv6 multicast address - None
Ipv6SolicitationMulticastAddress Copies multicast solicitation address.	<p>None Ipv6SolicitationMulticastAddress (<code>in6_addr* dst_addr, in6_addr* target</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>dst_addr</code> - ipv6 address pointer. • <code>target</code> - ipv6 multicast address pointer. <p>Returns:</p> <ul style="list-style-type: none"> • None - None
Ipv6AllRoutersMulticastAddress Function to assign all routers multicast address.	<p>None Ipv6AllRoutersMulticastAddress (<code>in6_addr* dst dst</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>dst</code> - IPv6 address pointer, <p>Returns:</p> <ul style="list-style-type: none"> • None - None
IPv6GetLinkLocalAddress Gets ipv6 link local address of the interface in output parameter <code>addr</code> .	<p>None IPv6GetLinkLocalAddress (<code>node, int interface, in6_addr* addr</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to the node structure. • <code>interface</code> - interface Index. • <code>addr</code> - ipv6 address pointer. <p>Returns:</p> <ul style="list-style-type: none"> • None - None
IPv6GetSiteLocalAddress	None IPv6GetSiteLocalAddress (<code>node, int interface, in6_addr* addr</code>)

	<p>Gets ipv6 site local address of the interface in output parameter addr.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to the node structure. • interface - interface Index. • addr - ipv6 address pointer. <p>Returns:</p> <ul style="list-style-type: none"> • None - None
IPv6GetSiteLocalAddress	<p>None IPv6GetSiteLocalAddress (node, int interface, in6_addr* addr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to the node structure. • interface - interface Index. • addr - ipv6 address pointer. <p>Returns:</p> <ul style="list-style-type: none"> • None - None
Ipv6GetPrefix	<p>None Ipv6GetPrefix (in6_addr* addr, in6_addr* prefix)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • addr - ipv6 address pointer. • prefix - ipv6 prefix pointer. <p>Returns:</p> <ul style="list-style-type: none"> • None - None
Ipv6GetPrefixFromInterfaceIndex	<p>Prefix for this interface Ipv6GetPrefixFromInterfaceIndex (Node* node, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Node pointer • interfaceIndex - interface for which multicast is required <p>Returns:</p> <ul style="list-style-type: none"> • Prefix for this interface - None
Ipv6OutputQueueIsEmpty	<p>BOOL Ipv6OutputQueueIsEmpty (Node *node node, int interfaceIndex)</p> <p>Parameters:</p>

<p>Check weather output queue is empty</p>	<ul style="list-style-type: none"> • node - Pointer to Node • interfaceIndex - interfaceIndex <p>Returns:</p> <ul style="list-style-type: none"> • BOOL - None
<p>Ipv6RoutingStaticInit</p> <p>Ipv6 Static routing initialization function.</p>	<p>None Ipv6RoutingStaticInit (Node *node node, const NodeInput nodeInput, NetworkRoutingProtocolType type)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node • nodeInput - *nodeInput • type - type <p>Returns:</p> <ul style="list-style-type: none"> • None - None
<p>Ipv6RoutingStaticEntry</p> <p>Static routing route entry function</p>	<p>None Ipv6RoutingStaticEntry (Node *node node, char currentLine[] currentLine)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node • currentLine - Static entry's current line. <p>Returns:</p> <ul style="list-style-type: none"> • None - None
<p>Ipv6AddDestination</p> <p>Adds destination in the destination cache.</p>	<p>None Ipv6AddDestination (Node* node node, route* ro ro)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node • ro - Pointer to destination route. <p>Returns:</p> <ul style="list-style-type: none"> • None - None
<p>Ipv6DeleteDestination</p> <p>Deletes destination from the destination cache.</p>	<p>None Ipv6DeleteDestination (Node* node node)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node <p>Returns:</p> <ul style="list-style-type: none"> • None - None

Ipv6CheckForValidPacket	<p>int Ipv6CheckForValidPacket (Node* node node, SchedulerType* scheduler scheduler, unsigned int* pIndex pIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node • scheduler - pointer to scheduler • pIndex - packet index <p>Returns:</p> <ul style="list-style-type: none"> • int - None
Ipv6NdpProcessing	<p>None Ipv6NdpProcessing (Node* node node)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node <p>Returns:</p> <ul style="list-style-type: none"> • None - None
Ipv6UpdateForwardingTable	<p>None Ipv6UpdateForwardingTable (Node* node node, in6_addr destPrefix destPrefix, in6_addr nextHopPrefix nextHopPrefix, int interfaceIndex, int metric metric)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node • destPrefix - IPv6 destination address • nextHopPrefix - IPv6 next hop address for this destination • interfaceIndex - interfaceIndex • metric - hop count between source and destination <p>Returns:</p> <ul style="list-style-type: none"> • None - None
Ipv6EmptyForwardingTable	<p>None Ipv6EmptyForwardingTable (Node* node node, NetworkRoutingProtocolType type type)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node • type - Routing protocol type <p>Returns:</p> <ul style="list-style-type: none"> • None - None

Ipv6PrintForwardingTable	<p>Prints the forwarding table.</p> <p>None Ipv6PrintForwardingTable (Node* node node)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node <p>Returns:</p> <ul style="list-style-type: none"> • None - None
Ipv6InterfaceIndexFromSubnetAddress	<p>Get the interface index from an IPv6 subnet address.</p> <p>Interface index associated with specified subnet address. Ipv6InterfaceIndexFromSubnetAddress (Node* node node, in6_addr* address)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node • address - Subnet Address <p>Returns:</p> <ul style="list-style-type: none"> • Interface index associated with specified subnet address. - None
Ipv6GetInterfaceAndNextHopFromForwardingTable	<p>Do a lookup on the routing table with a destination IPv6 address to obtain an outgoing interface and a next hop IPv6 address.</p> <p>void Ipv6GetInterfaceAndNextHopFromForwardingTable (Node* node node, in6_addr destAddr, int* interfaceIndex, in6_addr* nextHopAddr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node • destAddr - Destination Address • interfaceIndex - Pointer to interface index • nextHopAddr - Next Hop Addr for destination. <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL.
Ipv6GetInterfaceIndexForDestAddress	<p>Get interface for the destination address.</p> <p>interface index associated with destination. Ipv6GetInterfaceIndexForDestAddress (Node* node node, in6_addr destAddr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node • destAddr - Destination Address <p>Returns:</p> <ul style="list-style-type: none"> • interface index associated with destination. - None
Ipv6GetMetricForDestAddress	interface index associated with destination. Ipv6GetMetricForDestAddress (Node* node node, in6_addr destAddr)

	<p>Get the cost metric for a destination from the forwarding table.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node • destAddr - Destination Address <p>Returns:</p> <ul style="list-style-type: none"> • interface index associated with destination. - None
Ipv6IpGetInterfaceIndexForNextHop <p>This function looks at the network address of each of a node's network interfaces. When nextHopAddress is matched to a network, the interface index corresponding to the network is returned.</p>	<p>Interface index associated with destination if found, Ipv6IpGetInterfaceIndexForNextHop (Node* node node, in6_addr destAddr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node • destAddr - Destination Address <p>Returns:</p> <ul style="list-style-type: none"> • Interface index associated with destination if found, - None
Ipv6GetRouterFunction <p>Get the router function pointer.</p>	<p>Ipv6RouterFunctionType Ipv6GetRouterFunction (Node* node, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • interfaceIndex - interface associated with router function <p>Returns:</p> <ul style="list-style-type: none"> • Ipv6RouterFunctionType - router function pointer.
Ipv6SendPacketToMacLayer <p>Used if IPv6 next hop address and outgoing interface is known.</p>	<p>void Ipv6SendPacketToMacLayer (Node* node node, Message* msg, in6_addr destAddr, in6_addr* nextHopAddr, int* interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node • msg - Pointer to message • destAddr - Destination Address • nextHopAddr - Next Hop Addr for destination. • interfaceIndex - Pointer to interface index <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL.

Ipv6JoinMulticastGroup	<p>Join a multicast group.</p> <p>void Ipv6JoinMulticastGroup (Node* node, in6_addr mcastAddr, clocktype delay)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • mcastAddr - multicast group to join. • delay - delay. <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL.
Ipv6AddToMulticastGroupList	<p>Add group to multicast group list.</p> <p>void Ipv6AddToMulticastGroupList (Node* node, in6_addr groupAddress)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • groupAddress - Group to add to multicast group list. <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL.
Ipv6LeaveMulticastGroup	<p>Leave a multicast group.</p> <p>void Ipv6LeaveMulticastGroup (Node* node, in6_addr mcastAddr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • mcastAddr - multicast group to leave. <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL.
Ipv6RemoveFromMulticastGroupList	<p>Remove group from multicast group list.</p> <p>void Ipv6RemoveFromMulticastGroupList (Node* node, in6_addr groupAddress)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • groupAddress - Group to be removed from multicast <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL.
Ipv6NotificationOfPacketDrop	<p>Invoke callback functions when a packet is dropped.</p> <p>void Ipv6NotificationOfPacketDrop (Node* node, Message* msg, const NodeAddress nextHopAddress, int interfaceIndex)</p> <p>Parameters:</p>

	<ul style="list-style-type: none"> • node - Pointer to node. • msg - Pointer to message. • nextHopAddress - Next Hop Address • interfaceIndex - Interface Index <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL.
Ipv6IsPartOfMulticastGroup	<p>TRUE if node is part of multicast group, Ipv6IsPartOfMulticastGroup (Node* node, Message* msg, in6_addr groupAddress)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • msg - Pointer to message. • groupAddress - Multicast Address <p>Returns:</p> <ul style="list-style-type: none"> • TRUE if node is part of multicast group, - None
Ipv6IsReservedMulticastAddress	<p>TRUE if reserved multicast address, FALSE otherwise. Ipv6IsReservedMulticastAddress (Node* node, in6_addr mcastAddr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • mcastAddr - multicast group to join. <p>Returns:</p> <ul style="list-style-type: none"> • TRUE if reserved multicast address, FALSE otherwise. - None
Ipv6InMulticastOutgoingInterface	<p>TRUE if interface is in multicast outgoing interface Ipv6InMulticastOutgoingInterface (Node* node, LinkedList* list, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • list - Pointer to Linked List. • interfaceIndex - Interface Index. <p>Returns:</p> <ul style="list-style-type: none"> • TRUE if interface is in multicast outgoing interface - None

Ipv6UpdateMulticastForwardingTable	<p>update entry with (sourceAddress, multicastGroupAddress) pair. search for the row with (sourceAddress, multicastGroupAddress) and update its interface.</p>	<p>void Ipv6UpdateMulticastForwardingTable (Node* node, in6_addr sourceAddress, in6_addr multicastGroupAddress)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • sourceAddress - Source Address. • multicastGroupAddress - multicast group. <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL.
Ipv6GetOutgoingInterfaceFromMulticastTable	<p>get the interface List that lead to the (source, multicast group) pair.</p>	<p>Interface List if match found, NULL otherwise. Ipv6GetOutgoingInterfaceFromMulticastTable (Node* node, in6_addr sourceAddress, in6_addr groupAddress)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • sourceAddress - Source Address • groupAddress - multicast group address <p>Returns:</p> <ul style="list-style-type: none"> • Interface List if match found, NULL otherwise. - None
Ipv6CreateBroadcastAddress	<p>Create IPv6 Broadcast Address (ff02 followed by all one).</p>	<p>void Ipv6CreateBroadcastAddress ()</p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL.
Ipv6GetPrefixLength	<p>Get prefix length of an interface.</p>	<p>Prefix Length. Ipv6GetPrefixLength ()</p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> • Prefix Length. - None
Ipv6SetMacLayerStatusEventHandlerFunction	<p>Allows the MAC layer to send status messages (e.g., packet drop, link failure) to a network-layer routing protocol for routing optimization.</p>	<p>void Ipv6SetMacLayerStatusEventHandlerFunction (Node* node, Ipv6MacLayerStatusEventHandlerFunctionType StatusEventHandlerPtr, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • StatusEventHandlerPtr - Function Pointer • interfaceIndex - Interface Index

	<p>Returns:</p> <ul style="list-style-type: none"> • void - NULL.
Ipv6DeleteOutboundPacketsToANode	<p>Deletes all packets in the queue going to the specified next hop address. There is option to return all such packets back to the routing protocols.</p> <p>void Ipv6DeleteOutboundPacketsToANode (Node* node, const in6_addr nextHopAddress, const in6_addr destinationAddress, const BOOL returnPacketsToRoutingProtocol)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • nextHopAddress - Next Hop Address. • destinationAddress - Destination Address • returnPacketsToRoutingProtocol - bool <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL.
Ipv6IsLoopbackAddress	<p>Check if address is self loopback address.</p> <p>void Ipv6IsLoopbackAddress (Node* node, in6_addr address)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • address - ipv6 address <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL.
Ipv6IsMyIp	<p>Check if address is self loopback address.</p> <p>TRUE if my Ip, FALSE otherwise. Ipv6IsMyIp (Node* node, in6_addr* dst_addr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • dst_addr - Pointer to ipv6 address <p>Returns:</p> <ul style="list-style-type: none"> • TRUE if my Ip, FALSE otherwise. - None
Ipv6IsValidGetMulticastScope	<p>Check if multicast address has valid scope.</p> <p>Scope value if valid multicast address, 0 otherwise. Ipv6IsValidGetMulticastScope (Node* node, in6_addr multiAddr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • multiAddr - multicast address. <p>Returns:</p>

	<ul style="list-style-type: none">• Scope value if valid multicast address, 0 otherwise. - None
IsIPV6RoutingEnabledOnInterface To check if IPV6 Routing is enabled on interface?	BOOL IsIPV6RoutingEnabledOnInterface (Node* node, int interfaceIndex) Parameters: <ul style="list-style-type: none">• node - node structure pointer.• interfaceIndex - interface Index. Returns: <ul style="list-style-type: none">• BOOL - None



Contact [QualNet Support](#) for questions pertaining to the QualNet API Reference. This document is confidential and proprietary.
It may not be reproduced or distributed without the expressed written consent of [Scalable Network Technologies](#).

QualNet® is a Registered Trademark of [Scalable Network Technologies](#).

Copyright © 2001-2012 [Scalable Network Technologies, Inc.](#) All rights reserved.

QualNet 6.1 API Reference

LIST

This file describes the data structures and functions used in the implementation of lists.

Constant / Data Structure Summary

Type	Name
STRUCT	ListItem_template Structure for each item of a generic container list
STRUCT	List A list that stores different types of structures.
STRUCT	IntList A list that stores integers.

Function / Macro Summary

Return Type	Summary
void	ListInit (Node* node, LinkedList** list) Initialize the list
BOOL	ListIsEmpty (Node* node, LinkedList* list) Check if list is empty
int	ListGetSize (Node* node, LinkedList* list) Get the size of the list
void	ListInsert (Node* node, LinkedList* list, clocktype timeStamp, void* data)

	Insert an item at the end of the list
void*	FindItem (Node* node, List* list, int byteSkip, char* key, int size)
	Find an item from the list
void*	FindItem (Node* node, List* list, int byteSkip, char* key, int size)
	Find an item from the list
void	ListGet (Node* node, List* list, ListItem* listItem, BOOL freeItem, BOOL isMsg)
	Remove an item from the list
void	ListFree (Node* node, List* list, BOOL isMsg)
	Free the entire list
void	IntListInit (Node* node, IntList** list)
	Initialize the list
BOOL	IntListIsEmpty (Node* node, IntList* list)
	Check if list is empty
int	IntListGetSize (Node* node, IntList* list)
	Get the size of the list
void	ListInsert (Node* node, List* list, clocktype timeStamp, void* data)
	Insert an item at the end of the list
void	IntListGet (Node* node, IntList* list, IntListItem* listItem, BOOL freeItem, BOOL isMsg)
	Remove an item from the list
void	IntListFree (Node* node, IntList* list, BOOL isMsg)
	Free the entire list

Constant / Data Structure Detail

Structure	ListItem template Structure for each item of a generic container list
Structure	List A list that stores different types of structures.
Structure	IntList A list that stores integers.

Function / Macro Detail

Function / Macro	Format
ListInit Initialize the list	void ListInit (Node* node, LinkedList** list) Parameters: <ul style="list-style-type: none">• node - Node that contains the list• list - Pointer to list pointer Returns: <ul style="list-style-type: none">• void - NULL
ListIsEmpty Check if list is empty	BOOL ListIsEmpty (Node* node, LinkedList* list) Parameters: <ul style="list-style-type: none">• node - Node that contains the list• list - Pointer to the list Returns: <ul style="list-style-type: none">• BOOL - If empty, TRUE, non-empty, FALSE
ListGetSize Get the size of the list	int ListGetSize (Node* node, LinkedList* list) Parameters: <ul style="list-style-type: none">• node - Pointer to the node containing the list

	<ul style="list-style-type: none"> • <code>list</code> - Pointer to the list <p>Returns:</p> <ul style="list-style-type: none"> • <code>int</code> - Size of the list
ListInsert	<p><code>void ListInsert (Node* node, LinkedList* list, clocktype timeStamp, void* data)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to the node containing the list • <code>list</code> - Pointer to the list • <code>timeStamp</code> - Time the item was last inserted. • <code>data</code> - item to be inserted <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
FindItem	<p><code>void* FindItem (Node* node, List* list, int byteSkip, char* key, int size)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to the node containing the list • <code>list</code> - Pointer to the list • <code>byteSkip</code> - How many bytes skip to get the key item • <code>key</code> - The key that an item is idendified. • <code>size</code> - Size of the key element in byte <p>Returns:</p> <ul style="list-style-type: none"> • <code>void*</code> - Item found, NULL if not found
FindItem	<p><code>void* FindItem (Node* node, List* list, int byteSkip, char* key, int size)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to the node containing the list • <code>list</code> - Pointer to the list • <code>byteSkip</code> - How many bytes skip to get the key item • <code>key</code> - The key that an item is idendified. • <code>size</code> - Size of the key element in byte <p>Returns:</p>

	<ul style="list-style-type: none"> • <code>void*</code> - Item found, <code>NULL</code> if not found
ListGet	<p>void ListGet (<code>Node* node, List* list, ListItem* listItem, BOOL freeItem, BOOL isMsg</code>)</p> <p>Parameters:</p> <p>Remove an item from the list</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to the node containing the list • <code>list</code> - Pointer to the list to remove item from • <code>listItem</code> - item to be removed • <code>freeItem</code> - Whether to free the item • <code>isMsg</code> - Whether is this item a message? If it is <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - <code>NULL</code>
ListFree	<p>void ListFree (<code>Node* node, List* list, BOOL isMsg</code>)</p> <p>Parameters:</p> <p>Free the entire list</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to the node containing the list • <code>list</code> - Pointer to the list to be freed • <code>isMsg</code> - Does the list contain Messages? If so, we <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - <code>NULL</code>
IntListInit	<p>void IntListInit (<code>Node* node, IntList** list</code>)</p> <p>Parameters:</p> <p>Initialize the list</p> <ul style="list-style-type: none"> • <code>node</code> - Node that contains the list • <code>list</code> - Pointer to list pointer <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - <code>NULL</code>
IntListIsEmpty	<p>BOOL IntListIsEmpty (<code>Node* node, IntList* list</code>)</p> <p>Parameters:</p> <p>Check if list is empty</p> <ul style="list-style-type: none"> • <code>node</code> - Node that contains the list • <code>list</code> - Pointer to the list

	<p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - If empty, TRUE, non-empty, FALSE
IntListGetSize	<p><code>int IntListGetSize (Node* node, IntList* list)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to the node containing the list • <code>list</code> - Pointer to the list <p>Returns:</p> <ul style="list-style-type: none"> • <code>int</code> - Size of the list
ListInsert	<p><code>void ListInsert (Node* node, List* list, clocktype timeStamp, void* data)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to the node containing the list • <code>list</code> - Pointer to the list • <code>timeStamp</code> - Time the item was last inserted. • <code>data</code> - item to be inserted <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
IntListGet	<p><code>void IntListGet (Node* node, IntList* list, IntListItem* listItem, BOOL freeItem, BOOL isMsg)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to the node containing the list • <code>list</code> - Pointer to the list to remove item from • <code>listItem</code> - item to be removed • <code>freeItem</code> - Whether to free the item • <code>isMsg</code> - Whether is this item a message? If it is <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
IntListFree	<p><code>void IntListFree (Node* node, IntList* list, BOOL isMsg)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to the node containing the list

- `list` - Pointer to the list to be freed
- `isMsg` - Does the list contain Messages? If so, we

Returns:

- `void` - NULL



Contact [QualNet Support](#) for questions pertaining to the QualNet API Reference. This document is confidential and proprietary.
It may not be reproduced or distributed without the expressed written consent of [Scalable Network Technologies](#).

QualNet® is a Registered Trademark of [Scalable Network Technologies](#).

Copyright © 2001-2012 [Scalable Network Technologies, Inc.](#). All rights reserved.

QualNet 6.1 API Reference

MAC LAYER

This file describes data structures and functions used by the MAC Layer.

Constant / Data Structure Summary

Type	Name
CONSTANT	MAC_PROPAGATION_DELAY Peer to Peer Propogation delay in the MAC
CONSTANT	MAC_ADDRESS_LENGTH_IN_BYTE MAC address length
CONSTANT	Max_MacAdress_Length Maximum MAC address length
CONSTANT	MAC_ADDRESS_DEFAULT_LENGTH MAC address length in byte or octets
CONSTANT	MAC_CONFIGURATION_ATTRIBUTE Number of attribute of mac address file
CONSTANT	HW_TYPE_NETROM From KA9Q NET/ROM pseudo Hardware type.
CONSTANT	HW_TYPE_ETHER Ethernet 10/100Mbps Hardware type Ethernet.
CONSTANT	HW_TYPE_EETHER Hardware type Experimental Ethernet
CONSTANT	HW_TYPE_AX25

	Hardware type AX.25 Level 2
CONSTANT	HW_TYPE_PRONET
	Hardware type PROnet token ring
CONSTANT	HW_TYPE_CHAOS
	Hardware type Chaosnet
CONSTANT	HW_TYPE_IEEE802
	IEEE 802.2 Ethernet/TR/TB
CONSTANT	HW_TYPE_ARCNET
	Hardware type ARCnet
CONSTANT	HW_TYPE_APPLETALK
	Hardware type APPletalk
CONSTANT	HW_TYPE_DLCI
	Frame Relay DLCI
CONSTANT	HW_TYPE_ATM
	ATM 10/100Mbps
CONSTANT	HW_TYPE_METRICOM
	Hardware type HW_TYPE_METRICOM
CONSTANT	HW_TYPE_IEEE_1394
	Hardware type IEEE_1394
CONSTANT	HW_TYPE_EUI_64
	Hardware identifier
CONSTANT	HW_TYPE_UNKNOWN

	Unknown Hardware type MAC protocol HARDWARE identifiers.
CONSTANT	MAC_IPV4_LINKADDRESS_LENGTH Length of 4 byte MacAddress
CONSTANT	MAC_NODEID_LINKADDRESS_LENGTH Length of 2 byte MacAddress
CONSTANT	IPV4_LINKADDRESS Hardware identifier
CONSTANT	HW_NODE_ID Hardware identifier
CONSTANT	INVALID_MAC_ADDRESS INVALID MAC ADDRESS
CONSTANT	STATION_VLAN_TAGGING_DEFAULT Default VLAN TAGGING Value for a STATION node
ENUMERATION	MacInterfaceState Describes one out of two possible states of MAC interface - enable or disable
ENUMERATION	MacLinkType Describes different link type
ENUMERATION	MAC_PROTOCOL Specifies different MAC_PROTOCOLs used
ENUMERATION	MAC_SECURITY Specifies different MAC_SECURITY_PROTOCOLs used
ENUMERATION	ManagementRequestType

	Type of management request message
ENUMERATION	ManagementResponseType
	Type of management response message
ENUMERATION	MacLinkType
	Describes different fault type
STRUCT	MacHWAddress
	MAC hardware address of variable length
STRUCT	Mac802Address
	MAC address of size MAC_ADDRESS_LENGTH_IN_BYTE. It is default Mac address of type 802
STRUCT	MacVlan
	Structure of VLAN in MAC sublayer
STRUCT	MacHeaderVlanTag
	Structure of MAC sublayer VLAN header
STRUCT	MacData
	A composite structure representing MAC sublayer which is typedefed to MacData in main.h
STRUCT	ManagementRequest
	data structure of management request
STRUCT	ManagementResponse
	data structure of management response
STRUCT	MacToPhyPacketDelayInfoType
	Specifies the MAC to Physical layer delay information structure
STRUCT	MacFaultInfo

	Fields for keeping track of interface faults
STRUCT	<p>RandFault</p> <p>Structure containing random fault information.</p>

Function / Macro Summary

Return Type	Summary
MACRO	<p>MAC_EnableInterface(node, interfaceIndex)</p> <p>Enable the MAC_interface</p>
MACRO	<p>MAC_DisableInterface(node, interfaceIndex)</p> <p>Disable the MAC_interface</p>
MACRO	<p>MAC_ToggleInterfaceStatus(node, interfaceIndex)</p> <p>Toggle the MAC_interface status</p>
MACRO	<p>MAC_InterfaceIsEnabled(node, interfaceIndex)</p> <p>To query MAC_interface status is enabled or not</p>
void	<p>MacReportInterfaceStatus(Node* node, int interfaceIndex, MacInterfaceState state)</p> <p>Callback funtion to report interface status</p>
void	<p>MAC_SetInterfaceStatusHandlerFunction(Node* node, int interfaceIndex, MacReportInterfaceStatus statusHandler)</p> <p>Set the MAC interface handler function to be called when interface faults occurs</p>
MacReportInterfaceStatus	<p>MAC_GetInterfaceStatusHandlerFunction(Node* node, int interfaceIndex)</p> <p>To get the MACInterface status handling function for the system</p>
void	<p>MacHasFrameToSendFn(Node* node, int interfaceIndex)</p> <p>Callback funtion for sending packet. It calls when network layer has packet to send.</p>
void	<p>MacReceiveFrameFn(Node* node, int interfaceIndex, Message* msg)</p>

	Callback funtion to receive packet.
void	MAC_NetworkLayerHasPacketToSend (Node* node, int interfaceIndex)
	Handles packets from the network layer when the network queue is empty
void	MAC_SwitchHasPacketToSend (Node* node, int interfaceIndex)
	To inform MAC that the Switch has packets to to send
void	MAC_ReceivePacketFromPhy (Node* node, int interfaceIndex, Message* packet)
	Handles packets received from physical layer
void	MAC_ManagementRequest (Node* node, int interfaceIndex, ManagementRequest* req, ManagementResponse* resp)
	Deliver a network management request to the MAC
void	MAC_ReceivePhyStatusChangeNotification (Node* node, int interfaceIndex, PhyStatusType oldPhyStatus, PhyStatusType newPhyStatus, clocktype receiveDuration, Message* potentialIncomingPacket)
	Handles status changes received from the physical layer
void	MAC_InitUserMacProtocol (Node* node, NodeInput nodeInput, const char* macProtocolName, int interfaceIndex)
	Initialisation function for the User MAC_protocol
void	MacFinalizeUserMacProtocol (Node* node, int interfaceIndex)
	Finalization function for the User MAC_protocol
void	MAC_HandleUserMacProtocolEvent (Node* node, int interfaceIndex, Message* packet)
	Handles the MAC protocol event
BOOL	MAC_OutputQueueIsEmpty (Node* node, int interfaceIndex)
	To check if Output queue for an interface of a node if empty or not
void	MAC_NotificationOfPacketDrop (Node* node, NodeAddress nextHopAddress, int interfaceIndex, Message* msg)
	To notify MAC of packet drop
void	MAC_NotificationOfPacketDrop (Node* node, MacHWAddress nextHopAddress, int interfaceIndex, Message* msg)

	To notify MAC of packet drop <code>MAC_NotificationOfPacketDrop(Node* node, Mac802Address nextHopAddress, int interfaceIndex, Message* msg)</code>
BOOL	To notify MAC of packet drop <code>MAC_OutputQueueTopPacketForAPriority(Node* node, int interfaceIndex, TosType priority, Message** msg, NodeAddress nextHopAddress)</code>
BOOL	To notify MAC of priority packet arrival <code>MAC_OutputQueueTopPacketForAPriority(Node* node, int interfaceIndex, TosType priority, Message** msg, Mac802Address* nextHopAddress)</code>
BOOL	To notify MAC of priority packet arrival <code>MAC_OutputQueueTopPacketForAPriority(Node* node, int interfaceIndex, TosType priority, Message** msg, MacHWAddress* nextHopAddress)</code>
BOOL	To notify MAC of priority packet arrival <code>MAC_OutputQueueDequeuePacketForAPriority(Node* node, int interfaceIndex, TosType priority, Message** msg, NodeAddress* nextHopAddress, int* networkType)</code>
BOOL	To remove the packet at the front of the specified priority output queue <code>MAC_OutputQueueDequeuePacketForAPriority(Node* node, int interfaceIndex, TosType priority, Message** msg, MacHWAddress* nextHopAddress, int* networkType)</code>
BOOL	To remove the packet at the front of the specified priority output queue <code>MAC_OutputQueueDequeuePacketForAPriority(Node* node, int interfaceIndex, TosType priority, Message** msg, Mac802Address* nextHopAddress, int* networkType)</code>
BOOL	To remove the packet at the front of the specified priority output queue <code>MAC_OutputQueueDequeuePacketForAPriority(Node* node, int interfaceIndex, TosType* priority, Message** msg, MacHWAddress* destMacAddr, int* networkType, int* packType)</code>
BOOL	To allow a peek by network layer at packet before processing It is overloading function used for ARP packet <code>MAC_OutputQueueDequeuePacketForAPriority(Node* node, int interfaceIndex, TosType* priority, Message** msg, Mac802Address* destMacAddr, int* networkType, int* packType)</code>
void	To allow a peek by network layer at packet before processing It is overloading function used for ARP packet <code>MAC_SneakPeekAtMacPacket(Node* node, int interfaceIndex, const Message* msg, NodeAddress prevHop, NodeAddress destAddr, int messageType)</code>

	To allow a peek by network layer at packet before processing
void	<code>MAC_SneakPeekAtMacPacket</code> (Node* node, int interfaceIndex, const Message* msg, MacHWAddress prevHop, MacHWAddress destAddr, int arpMessageType)
	To allow a peek by network layer at packet before processing
void	<code>MAC_SneakPeekAtMacPacket</code> (Node* node, int interfaceIndex, const Message* msg, Mac802Address prevHop, Mac802Address destAddr, int messageType)
	To allow a peek by network layer at packet before processing
void	<code>MAC_MacLayerAcknowledgement</code> (Node* node, int interfaceIndex, Message* msg, NodeAddress nextHop)
	To send acknowledgement from MAC
void	<code>MAC_MacLayerAcknowledgement</code> (Node* node, int interfaceIndex, Message* msg, MacHWAddress& nextHop)
	To send acknowledgement from MAC
void	<code>MAC_MacLayerAcknowledgement</code> (Node* node, int interfaceIndex, Message* msg, Mac802Address& nextHop)
	To send acknowledgement from MAC
void	<code>MAC_HandOffSuccessfullyReceivedPacket</code> (Node* node, int interfaceIndex, Message* msg, NodeAddress lastHopAddress)
	Pass packet successfully up to the network layer
void	<code>MAC_HandOffSuccessfullyReceivedPacket</code> (Node* node, int interfaceIndex, Message* msg, MacHWAddress* lastHopAddr)
	Pass packet successfully up to the network layer
void	<code>MAC_HandOffSuccessfullyReceivedPacket</code> (Node* node, int interfaceIndex, Message* msg, Mac802Address* lastHopAddr)
	Pass packet successfully up to the network layer
void	<code>MAC_HandOffSuccessfullyReceivedPacket</code> (Node* node, int interfaceIndex, Message* msg, MacHWAddress* lastHopAddress, int arpMessageType)
	Pass packet successfully up to the network layer It is overloading function used for ARP packet
void	<code>MAC_HandOffSuccessfullyReceivedPacket</code> (Node* node, int interfaceIndex, Message* msg, Mac802Address* lastHopAddress, int arpMessageType)
	Pass packet successfully up to the network layer It is overloading function used for ARP packet
BOOL	<code>MAC_OutputQueueTopPacket</code> (Node* node, int interfaceIndex, Message** msg, NodeAddress* nextHopAddress, int networkType, TosType* priority)

	To check packet at the top of output queue
BOOL	<code>MAC_OutputQueueTopPacket</code> (Node* node, int interfaceIndex, Message** msg, MacHWAddress* nextHopAddress, int networkType, TosType* priority)
	To check packet at the top of output queue
BOOL	<code>MAC_OutputQueueTopPacket</code> (Node* node, int interfaceIndex, Message** msg, Mac802Address* nextHopAddress, int networkType, TosType* priority)
	To check packet at the top of output queue
BOOL	<code>MAC_OutputQueueDequeuePacket</code> (Node* node, int interfaceIndex, Message** msg, NodeAddress* nextHopAddress, int networkType, TosType * priority)
	To remove packet from front of output queue
BOOL	<code>MAC_OutputQueueDequeuePacket</code> (Node* node, int interfaceIndex, Message** msg, MacHWAddress* nextHopAddress, int networkType, TosType * priority)
	To remove packet from front of output queue
BOOL	<code>MAC_OutputQueueDequeuePacket</code> (Node* node, int interfaceIndex, Message** msg, Mac802Address* nextHopAddress, int networkType, TosType * priority)
	To remove packet from front of output queue, Its a overloaded function
BOOL	<code>MAC_OutputQueueDequeuePacket</code> (Node* node, int interfaceIndex, Message** msg, NodeAddress* nextHopAddress, int* networkType, TosType * priority, MacOutputQueueDequeueOption dequeueOption, MacOutputQueueDequeueCriteria dequeueCriteria, int * numFreeByte, int* numPacketPacked, TraceProtocolType tracePrt, BOOL eachWithMacHeader, int maxHeaderSize, BOOL returnPackedMsg)
	To remove packet(s) from front of output queue; process packets with options for example, pakcing multiple packets with same next hop address together
BOOL	<code>MAC_OutputQueueDequeuePacketForAPriority</code> (Node* node, int interfaceIndex, int priority, Message** msg, NodeAddress* nextHopAddress, int* networkType, MacOutputQueueDequeueOption dequeueOption, MacOutputQueueDequeueCriteria dequeueCriteria, int * numFreeByte, int* numPacketPacked, TraceProtocolType tracePrt, BOOL eachWithMacHeader, int maxHeaderSize, BOOL returnPackedMsg)
	To remove packet(s) from front of output queue; process packets with options for example, pakcing multiple packets with same next hop address together
BOOL	<code>MAC_IsMyUnicastFrame</code> (Node* node, NodeAddress destAddr)
	Check if a packet (or frame) belongs to this node Should be used only for four byte mac address
BOOL	<code>MAC_IsWiredNetwork</code> (Node* node, int interfaceIndex)

	To check if an interface is a wired interface MAC_IsPointToPointNetwork(Node* node, int interfaceIndex)
BOOL	Checks if an interface belongs to Point to PointNetwork MAC_IsPointToMultiPointNetwork(Node* node, int interfaceIndex)
BOOL	Checks if an interface belongs to Point to Multi-Point network. MAC_IsWiredBroadcastNetwork(Node* node, int interfaceIndex)
BOOL	Determines if an interface is a wired broadcast interface MAC_IsWirelessNetwork(Node* node, int interfaceIndex)
BOOL	Determine if a node's interface is a wireless interface MAC_IsWirelessAdHocNetwork(Node* node, int interfaceIndex)
BOOL	Determine if a node's interface is a possible wireless ad hoc interface MAC_IsOneHopBroadcastNetwork(Node* node, int interfaceIndex)
BOOL	Determines if an interface is a single Hop Broadcast interface MAC_IsASwitch(Node* node)
void	To check if a node is a switch MAC_SetVirtualMacAddress(Node* node, int interfaceIndex, NodeAddress virtualMacAddress)
void	To set MAC address MacSetDefaultHWAddress(NodeId nodeId, MacHWAddress* macAddr, int interfaceIndex)
NodeAddress	Set Default interface Hardware Address of node MAC_IsMyMacAddress(Node* node, int interfaceIndex, NodeAddress destAddr)
BOOL	To check if received mac address belongs to itself MAC_IsMyHWAddress(Node* node, int interfaceIndex, MacAddress* macAddr)

	Checks for own MAC address.
void	MacValidateAndSetHWAddress (char* macAddrStr, MacHWAddress* macAddr)
	Validate MAC Address String after fetching from user
NodeAddress	DefaultMacHWAddressToIpv4Address (Node* node, MacHWAddress* macAddr)
	Retrieve the IP Address from Default HW Address . Default HW address is equal to 6 bytes
void	MacGetHardwareLength (Node* node, int interface, unsigned short hwLength)
	Retrieve the Hardware Length.
void	MacGetHardwareType (Node* node, int interface, unsigned short* type)
	Retrieve the Hardware Type.
void	MacGetHardwareAddressString (Node* node, int interface)
	Retrieve the Hardware Address String.
void	MacAddNewInterface (Node* node, NodeAddress interfaceAddress, int numHostBits, int* interfaceIndex, const NodeInput nodeInput, char* macProtocolName)
	To add a new Interface at MAC
void	MacAddVlanInfoForThisInterface (Node* node, int* interfaceIndex, NodeAddress interfaceAddress, const NodeInput nodeInput)
	Init and read VLAN configuration from user input for node and interface passed as arguments
NodeAddress	MacReleaseVlanInfoForThisInterface (Node* node, int interfaceIndex)
	To flush VLAN info for an interface
BOOL	MAC_IsBroadcastHWAddress (MacHWAddress* macAddr)
	Checks Broadcast MAC address
BOOL	MAC_IsIdenticalHWAddress (MacHWAddress* macAddr1, MacHWAddress* macAddr2)
	Compares two MAC addresses
void	MAC_PrintHWAddr (MacHWAddress* macAddr)

	Prints interface Mac Address
void	MAC_PrintMacAddr (Mac802Address* macAddr)
	Prints interface Mac Address
void	MAC_RandFaultInit (Node* node, int interfaceIndex, const char* currentLine)
	Initialization the Random Fault structure from input file
void	MAC_RandFaultFinalize (Node* node, int interfaceIndex)
	IPrint the statistics of Random link fault.
TosType	MAC_GetPacketsPriority (Message* msg)
	Returns the priority of the packet
void	MAC_TranslateMulticastIPv4AddressToMulticastMacAddress (NodeAddress multicastAddress, MacHWAddress* macMulticast)
	Convert the Multicast ip address to multicast MAC address
BOOL	MAC_OutputQueuePeekByIndex (Node* node, int interfaceIndex, int msgIndex, Message** msg, NodeAddress nextHopAddress, TosType priority)
	Look at the packet at the index of the output queue.
BOOL	MAC_OutputQueuePeekByIndex (Node* node, int interfaceIndex, int msgIndex, Message** msg, Mac802Address* nextHopAddress, TosType priority)
	Look at the packet at the index of the output queue.
BOOL	MAC_OutputQueuePeekByIndex (int interfaceIndex, int msgIndex, Message** msg, MacHWAddress* nextHopAddress, TosType priority)
	Look at the packet at the index of the output queue.
BOOL	MAC_OutputQueueDequeuePacketWithIndex (Node* node, int interfaceIndex, int msgIndex, Message** msg, NodeAddress nextHopAddress, int networkType)
	To remove the packet at specified index output queue.
BOOL	MAC_OutputQueueDequeuePacketWithIndex (Node* node, int interfaceIndex, int msgIndex, Message** msg, Mac802Address* nextHopMacAddress, int networkType)

	To remove the packet at specified index output queue. MAC_OutputQueueDequeuePacketWithIndex (Node* node, int interfaceIndex, int msgIndex, Message** msg, MacHWAddress nextHopMacAddress, int networkType)
BOOL	To remove the packet at specified index output queue. MAC_IPv4addressIsMulticastAddress (NodeAddress ipV4)
BOOL	Check the given address is Multicast address or not. MAC_IsBroadcastMacAddress (MacAddress* macAddr)
void	Checks Broadcast MAC address. IPv4AddressToDefaultMac802Address (Node* node, int index, NodeAddress ipv4Address, Mac802Address* macAddr)
Bool	Retrieve the Mac802Address from IP address. ConvertVariableHWAddressTo802Address (Node* node, MacHWAddress* macHWAddr, Mac802Address* mac802Addr)
void	Convert Variable Hardware address to Mac 802 addtess MAC_CopyMacHWAddress (MacHWAddress* destAddr, MacHWAddress* srcAddr)
NodeAddress	Copies Hardware address address DefaultMac802AddressToIpv4Address (Node* node, Mac802Address* macAddr)
BOOL	Retrieve IP address from.Mac802Address IPv4AddressToHWAddress (Node* node, int interfaceIndex, Message* msg, NodeAddress ipv4Address)
NodeAddress	Converts IP address.To MacHWAddress MacHWAddressToIpv4Address (Node * node, int interfaceIndex, MacHWAddress* macAddr)
char*	This functions converts variable length Mac address to IPv4 address It checks the type of hardware address and based on that conversion is done. decToHex (int dec)
void	Convert one byte decimal number to hex number. MAC_FourByteMacAddressToVariableHWAddress (Node * node, int interfaceIndex, MacHWAddress * macAddr, NodeAddress nodeAddr)

NodeAddress	MAC_VariableHWAddressToFourByteMacAddress (Node* node, MacHWAddress* macAddr) Retrieve IP address from.MacHWAddress of type IPV4_LINKADDRESS
MacHWAddress	GetBroadCastAddress (Node* node, int interfaceIndex) Returns Broadcast Address of an interface
MacHWAddress	GetMacHWAddress (Node* node, int interfaceIndex) Returns MacHWAddress of an interface
int	MacGetInterfaceIndexFromMacAddress (Node* node, MacHWAddress macAddr) Returns interfaceIndex at which Macaddress is configured
int	MacGetInterfaceIndexFromMacAddress (Node* node, Mac802Address macAddr) Returns interfaceIndex at which Macaddress is configured
int	MacGetInterfaceIndexFromMacAddress (Node* node, NodeAddress macAddr) Returns interfaceIndex at which Macaddress is configured
void	MAC_Reset (Node* node, int InterfaceIndex) Reset the Mac protocols use by the node
void	MAC_AddResetFunctionList (Node* node, int InterfaceIndex, void* param) Add which protocols in the Mac layer to be reset to a fuction list pointer.

Constant / Data Structure Detail

Constant	Description
Constant	MAC_PROPAGATION_DELAY 1 * MICRO_SECOND Peer to Peer Propogation delay in the MAC
Constant	MAC_ADDRESS_LENGTH_IN_BYTE 6

	MAC address length
Constant	Max_MacAdress_Length 16 Maximum MAC address length
Constant	MAC_ADDRESS_DEFAULT_LENGTH 6 MAC address length in byte or octets
Constant	MAC_CONFIGURATION_ATTRIBUTE 5 Number of attribute of mac address file
Constant	HW_TYPE_NETROM 0 From KA9Q NET/ROM pseudo Hardware type.
Constant	HW_TYPE_ETHER 1 Ethernet 10/100Mbps Hardware type Ethernet.
Constant	HW_TYPE_EETHER 2 Hardware type Experimental Ethernet
Constant	HW_TYPE_AX25 3 Hardware type AX.25 Level 2
Constant	HW_TYPE_PRONET 4 Hardware type PROnet token ring
Constant	HW_TYPE_CHAOS 5 Hardware type Chaosnet
Constant	HW_TYPE_IEEE802 6

	IEEE 802.2 Ethernet/TR/TB
Constant	HW_TYPE_ARCNET 7 Hardware type ARCnet
Constant	HW_TYPE_APPLETALK 8 Hardware type APPLETalk
Constant	HW_TYPE_DLCI 15 Frame Relay DLCI
Constant	HW_TYPE_ATM 19 ATM 10/100Mbps
Constant	HW_TYPE_METRICOM 23 Hardware type HW_TYPE_METRICOM
Constant	HW_TYPE_IEEE_1394 24 Hardware type IEEE_1394
Constant	HW_TYPE_EUI_64 27 Hardware identifier
Constant	HW_TYPE_UNKNOWN 0xffff Unknown Hardware type MAC protocol HARDWARE identifiers.
Constant	MAC_IPV4_LINKADDRESS_LENGTH 4 Length of 4 byte MacAddress
Constant	MAC_NODEID_LINKADDRESS_LENGTH 2 Length of 2 byte MacAddress
Constant	IPV4_LINKADDRESS 28

	Hardware identifier
Constant	HW_NODE_ID 29
	Hardware identifier
Constant	INVALID_MAC_ADDRESS MacHWAddress()
	INVALID MAC ADDRESS
Constant	STATION_VLAN_TAGGING_DEFAULT FALSE
	Default VLAN TAGGING Value for a STATION node
Enumeration	MacInterfaceState
	Describes one out of two possible states of MAC interface - enable or disable
Enumeration	MacLinkType
	Describes different link type
Enumeration	MAC_PROTOCOL
	Specifies different MAC_PROTOCOLs used
Enumeration	MAC_SECURITY
	Specifies different MAC_SECURITY_PROTOCOLs used
Enumeration	ManagementRequestType
	Type of management request message
Enumeration	ManagementResponseType
	Type of management response message
Enumeration	MacLinkType

	Describes different fault type
Structure	<p>MacHWAddress</p> <p>MAC hardware address of variable length</p>
Structure	<p>Mac802Address</p> <p>MAC address of size MAC_ADDRESS_LENGTH_IN_BYTE. It is default Mac address of type 802</p>
Structure	<p>MacVlan</p> <p>Structure of VLAN in MAC sublayer</p>
Structure	<p>MacHeaderVlanTag</p> <p>Structure of MAC sublayer VLAN header</p>
Structure	<p>MacData</p> <p>A composite structure representing MAC sublayer which is typedefed to MacData in main.h</p>
Structure	<p>ManagementRequest</p> <p>data structure of management request</p>
Structure	<p>ManagementResponse</p> <p>data structure of management response</p>
Structure	<p>MacToPhyPacketDelayInfoType</p> <p>Specifies the MAC to Physical layer delay information structure</p>
Structure	<p>MacFaultInfo</p> <p>Fields for keeping track of interface faults</p>
Structure	<p>RandFault</p> <p>Structure containing random fault information.</p>

Function / Macro Detail

Function / Macro	Format
MAC_EnableInterface(node, interfaceIndex)	Enable the MAC_interface
MAC_DisableInterface(node, interfaceIndex)	Disable the MAC_interface
MAC_ToggleInterfaceStatus(node, interfaceIndex)	Toggle the MAC_interface status
MAC_InterfaceIsEnabled(node, interfaceIndex)	To query MAC_interface status is enabled or not
MacReportInterfaceStatus Callback funtion to report interface status	void MacReportInterfaceStatus (Node* node, int interfaceIndex, MacInterfaceState state) Parameters: <ul style="list-style-type: none">• node - Pointer to a network node• interfaceIndex - index of interface• state - Wheather it enable or disable Returns: <ul style="list-style-type: none">• void - None
MAC_SetInterfaceStatusHandlerFunction Set the MAC interface handler function to be called when interface faults occurs	void MAC_SetInterfaceStatusHandlerFunction (Node* node, int interfaceIndex, MacReportInterfaceStatus statusHandler) Parameters: <ul style="list-style-type: none">• node - Pointer to a network node• interfaceIndex - index of interface• statusHandler - Pointer to status Handler Returns: <ul style="list-style-type: none">• void - None
MAC_GetInterfaceStatusHandlerFunction To get the MACInterface status handling function for the system	MacReportInterfaceStatus MAC_GetInterfaceStatusHandlerFunction (Node* node, int interfaceIndex) Parameters: <ul style="list-style-type: none">• node - Pointer to a network node• interfaceIndex - index of interface

	<p>Returns:</p> <ul style="list-style-type: none"> • <code>MacReportInterfaceStatus</code> - Pointer to status handler
MacHasFrameToSendFn	<p>Callback function for sending packet. It calls when network layer has packet to send.</p> <p>void MacHasFrameToSendFn (Node* node, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node • <code>interfaceIndex</code> - index of interface <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
MacReceiveFrameFn	<p>Callback function to receive packet.</p> <p>void MacReceiveFrameFn (Node* node, int interfaceIndex, Message* msg)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node • <code>interfaceIndex</code> - index of interface • <code>msg</code> - Pointer to the message <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
MAC_NetworkLayerHasPacketToSend	<p>Handles packets from the network layer when the network queue is empty</p> <p>void MAC_NetworkLayerHasPacketToSend (Node* node, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to a network node • <code>interfaceIndex</code> - index of interface <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
MAC_SwitchHasPacketToSend	<p>To inform MAC that the Switch has packets to send</p> <p>void MAC_SwitchHasPacketToSend (Node* node, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to a network node • <code>interfaceIndex</code> - index of interface <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None

MAC_ReceivePacketFromPhy	<p>Handles packets received from physical layer</p> <p>void MAC_ReceivePacketFromPhy (Node* node, int interfaceIndex, Message* packet)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to a network node • interfaceIndex - index of interface • packet - Pointer to Message <p>Returns:</p> <ul style="list-style-type: none"> • void - None
MAC_ManagementRequest	<p>Deliver a network management request to the MAC</p> <p>void MAC_ManagementRequest (Node* node, int interfaceIndex, ManagementRequest* req, ManagementResponse* resp)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to a network node • interfaceIndex - index of interface • req - Pointer to a management request • resp - Pointer to a management response <p>Returns:</p> <ul style="list-style-type: none"> • void - None
MAC_ReceivePhyStatusChangeNotification	<p>Handles status changes received from the physical layer</p> <p>void MAC_ReceivePhyStatusChangeNotification (Node* node, int interfaceIndex, PhyStatusType oldPhyStatus, PhyStatusType newPhyStatus, clocktype receiveDuration, Message* potentialIncomingPacket)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to a network node • interfaceIndex - index of interface • oldPhyStatus - Old status of physical layer • newPhyStatus - New status of physical layer • receiveDuration - Duration after which received • potentialIncomingPacket - Pointer to incoming message <p>Returns:</p> <ul style="list-style-type: none"> • void - None
MAC_InitUserMacProtocol	<p>void MAC_InitUserMacProtocol (Node* node, NodeInput nodeInput, const char* macProtocolName, int interfaceIndex)</p>

<p>Initialisation function for the User MAC_protocol</p>	<p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to a network node • nodeInput - Configured Inputs for the node • macProtocolName - MAC protocol name • interfaceIndex - interface index <p>Returns:</p> <ul style="list-style-type: none"> • void - None
<p>MacFinalizeUserMacProtocol</p> <p>Finalization function for the User MAC_protocol</p>	<p>void MacFinalizeUserMacProtocol (Node* node, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to a network node • interfaceIndex - index of interface <p>Returns:</p> <ul style="list-style-type: none"> • void - None
<p>MAC_HandleUserMacProtocolEvent</p> <p>Handles the MAC protocol event</p>	<p>void MAC_HandleUserMacProtocolEvent (Node* node, int interfaceIndex, Message* packet)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to a network node • interfaceIndex - index of interface • packet - Pointer to Message <p>Returns:</p> <ul style="list-style-type: none"> • void - None
<p>MAC_OutputQueueIsEmpty</p> <p>To check if Output queue for an interface of a node if empty or not</p>	<p>BOOL MAC_OutputQueueIsEmpty (Node* node, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to a network node • interfaceIndex - index of interface <p>Returns:</p> <ul style="list-style-type: none"> • BOOL - empty or not
<p>MAC_NotificationOfPacketDrop</p>	<p>void MAC_NotificationOfPacketDrop (Node* node, NodeAddress nextHopAddress, int interfaceIndex, Message* msg)</p>

<p>To notify MAC of packet drop</p>	<p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to a network node • <code>nextHopAddress</code> - Node address • <code>interfaceIndex</code> - interfaceIndex • <code>msg</code> - Pointer to Message <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
<p>MAC_NotificationOfPacketDrop</p> <p>To notify MAC of packet drop</p>	<p>void MAC_NotificationOfPacketDrop (<code>Node* node, MacHWAddress nextHopAddress, int interfaceIndex, Message* msg</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to a network node • <code>nextHopAddress</code> - Node address • <code>interfaceIndex</code> - interfaceIndex • <code>msg</code> - Pointer to Message <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
<p>MAC_NotificationOfPacketDrop</p> <p>To notify MAC of packet drop</p>	<p>void MAC_NotificationOfPacketDrop (<code>Node* node, Mac802Address nextHopAddress, int interfaceIndex, Message* msg</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to a network node • <code>nextHopAddress</code> - mac address • <code>interfaceIndex</code> - interfaceIndex • <code>msg</code> - Pointer to Message <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
<p>MAC_OutputQueueTopPacketForAPriority</p> <p>To notify MAC of priority packet arrival</p>	<p>BOOL MAC_OutputQueueTopPacketForAPriority (<code>Node* node, int interfaceIndex, TosType priority, Message** msg, NodeAddress nextHopAddress</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to a network node

	<ul style="list-style-type: none"> • <code>interfaceIndex</code> - interfaceIndex • <code>priority</code> - Message Priority • <code>msg</code> - Pointer to Message • <code>nextHopAddress</code> - Next hop address <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - TRUE if there is a packet, FALSE otherwise.
MAC_OutputQueueTopPacketForAPriority	<p>To notify MAC of priority packet arrival</p> <p><code>BOOL MAC_OutputQueueTopPacketForAPriority (Node* node, int interfaceIndex, TosType priority, Message** msg, Mac802Address* nextHopAddress)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to a network node • <code>interfaceIndex</code> - interfaceIndex • <code>priority</code> - Message Priority • <code>msg</code> - Pointer to Message • <code>nextHopAddress</code> - Next hop mac address <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - TRUE if there is a packet, FALSE otherwise.
MAC_OutputQueueTopPacketForAPriority	<p>To notify MAC of priority packet arrival</p> <p><code>BOOL MAC_OutputQueueTopPacketForAPriority (Node* node, int interfaceIndex, TosType priority, Message** msg, MacHWAddress* nextHopAddress)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to a network node • <code>interfaceIndex</code> - interfaceIndex • <code>priority</code> - Message Priority • <code>msg</code> - Pointer to Message • <code>nextHopAddress</code> - Next hop mac address <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - TRUE if there is a packet, FALSE otherwise.
MAC_OutputQueueDequeuePacketForAPriority	<p><code>BOOL MAC_OutputQueueDequeuePacketForAPriority (Node* node, int interfaceIndex, TosType priority, Message** msg, NodeAddress* nextHopAddress, int* networkType)</code></p> <p>Parameters:</p>

<p>To remove the packet at the front of the specified priority output queue</p>	<ul style="list-style-type: none"> • <code>node</code> - Pointer to a network node • <code>interfaceIndex</code> - interfaceIndex • <code>priority</code> - Message Priority • <code>msg</code> - Pointer to Message • <code>nextHopAddress</code> - Next hop address • <code>networkType</code> - network type <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - TRUE if dequeued successfully, FALSE otherwise.
<p>MAC_OutputQueueDequeuePacketForAPriority</p> <p>To remove the packet at the front of the specified priority output queue</p>	<p><code>BOOL MAC_OutputQueueDequeuePacketForAPriority (Node* node, int interfaceIndex, TosType priority, Message** msg, MacHWAddress* nextHopAddress, int* networkType)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to a network node • <code>interfaceIndex</code> - interfaceIndex • <code>priority</code> - Message Priority • <code>msg</code> - Pointer to Message • <code>nextHopAddress</code> - Next hop mac address • <code>networkType</code> - network type <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - TRUE if dequeued successfully, FALSE otherwise.
<p>MAC_OutputQueueDequeuePacketForAPriority</p> <p>To remove the packet at the front of the specified priority output queue</p>	<p><code>BOOL MAC_OutputQueueDequeuePacketForAPriority (Node* node, int interfaceIndex, TosType priority, Message** msg, Mac802Address* nextHopAddress, int* networkType)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to a network node • <code>interfaceIndex</code> - interfaceIndex • <code>priority</code> - Message Priority • <code>msg</code> - Pointer to Message • <code>nextHopAddress</code> - Next hop mac address • <code>networkType</code> - network type

	<p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - TRUE if dequeued successfully, FALSE otherwise.
MAC_OutputQueueDequeuePacketForAPriority	<p>To allow a peek by network layer at packet before processing It is overloading function used for ARP packet</p> <p><code>BOOL MAC_OutputQueueDequeuePacketForAPriority (Node* node, int interfaceIndex, TosType* priority, Message** msg, MacHWAddress* destMacAddr, int* networkType, int* packType)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to a network node • <code>interfaceIndex</code> - interfaceIndex • <code>priority</code> - tos value • <code>msg</code> - Pointer to Message • <code>destMacAddr</code> - Dest addr Pointer • <code>networkType</code> - Network Type pointer • <code>packType</code> - packet Type pointer <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - If success TRUE NOTE : Overloaded <code>MAC_OutputQueueDequeuePacketForAPriority()</code>
MAC_OutputQueueDequeuePacketForAPriority	<p>To allow a peek by network layer at packet before processing It is overloading function used for ARP packet</p> <p><code>BOOL MAC_OutputQueueDequeuePacketForAPriority (Node* node, int interfaceIndex, TosType* priority, Message** msg, Mac802Address* destMacAddr, int* networkType, int* packType)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to a network node • <code>interfaceIndex</code> - interfaceIndex • <code>priority</code> - tos value • <code>msg</code> - Pointer to Message • <code>destMacAddr</code> - Dest addr Pointer • <code>networkType</code> - Network Type pointer • <code>packType</code> - packet Type pointer <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - If success TRUE NOTE : Overloaded <code>MAC_OutputQueueDequeuePacketForAPriority()</code>
MAC_SneakPeekAtMacPacket	<p>To allow a peek by network layer at packet before processing</p> <p><code>void MAC_SneakPeekAtMacPacket (Node* node, int interfaceIndex, const Message* msg, NodeAddress prevHop, NodeAddress destAddr, int messageType)</code></p> <p>Parameters:</p>

	<ul style="list-style-type: none"> • <code>node</code> - Pointer to a network node • <code>interfaceIndex</code> - interfaceIndex • <code>msg</code> - Pointer to Message • <code>prevHop</code> - Previous Node address • <code>destAddr</code> - Destination Node address • <code>messageType</code> - Distinguish between the ARP and general message <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
MAC_SneakPeekAtMacPacket	<p>To allow a peek by network layer at packet before processing</p> <p><code>void MAC_SneakPeekAtMacPacket (Node* node, int interfaceIndex, const Message* msg, MacHWAddress prevHop, MacHWAddress destAddr, int arpMessageType)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to a network node • <code>interfaceIndex</code> - interfaceIndex • <code>msg</code> - Pointer to Message • <code>prevHop</code> - Previous Node mac address • <code>destAddr</code> - Destination Node mac address • <code>arpMessageType</code> - Distinguish between the ARP and general message <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
MAC_SneakPeekAtMacPacket	<p>To allow a peek by network layer at packet before processing</p> <p><code>void MAC_SneakPeekAtMacPacket (Node* node, int interfaceIndex, const Message* msg, Mac802Address prevHop, Mac802Address destAddr, int messageType)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to a network node • <code>interfaceIndex</code> - interfaceIndex • <code>msg</code> - Pointer to Message • <code>prevHop</code> - Previous Node address • <code>destAddr</code> - Destination Node address • <code>messageType</code> - Distinguish between the ARP and general message <p>Returns:</p>

	<ul style="list-style-type: none"> • void - NULL
MAC_MacLayerAcknowledgement To send acknowledgement from MAC	<p>void MAC_MacLayerAcknowledgement (Node* node, int interfaceIndex, Message* msg, NodeAddress nextHop)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to a network node • interfaceIndex - interfaceIndex • msg - Pointer to Message • nextHop - Pointer to Node address <p>Returns:</p> <ul style="list-style-type: none"> • void - None
MAC_MacLayerAcknowledgement To send acknowledgement from MAC	<p>void MAC_MacLayerAcknowledgement (Node* node, int interfaceIndex, Message* msg, MacHWAddress& nextHop)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to a network node • interfaceIndex - interfaceIndex • msg - Pointer to Message • nextHop - Pointer to Node address <p>Returns:</p> <ul style="list-style-type: none"> • void - None
MAC_MacLayerAcknowledgement To send acknowledgement from MAC	<p>void MAC_MacLayerAcknowledgement (Node* node, int interfaceIndex, Message* msg, Mac802Address& nextHop)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to a network node • interfaceIndex - interfaceIndex • msg - Pointer to Message • nextHop - Pointer to nexthop mac address <p>Returns:</p> <ul style="list-style-type: none"> • void - None
MAC_HandOffSuccessfullyReceivedPacket	void MAC_HandOffSuccessfullyReceivedPacket (Node* node, int interfaceIndex, Message* msg,

Pass packet successfully up to the network layer	<p><code>NodeAddress lastHopAddress)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to a network node • <code>interfaceIndex</code> - interfaceIndex • <code>msg</code> - Pointer to Message • <code>lastHopAddress</code> - Node address <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
MAC_HandOffSuccessfullyReceivedPacket Pass packet successfully up to the network layer	<p><code>void MAC_HandOffSuccessfullyReceivedPacket (Node* node, int interfaceIndex, Message* msg, MacHWAddress* lastHopAddr)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to a network node • <code>interfaceIndex</code> - interfaceIndex • <code>msg</code> - Pointer to Message • <code>lastHopAddr</code> - mac address <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
MAC_HandOffSuccessfullyReceivedPacket Pass packet successfully up to the network layer	<p><code>void MAC_HandOffSuccessfullyReceivedPacket (Node* node, int interfaceIndex, Message* msg, Mac802ddress* lastHopAddr)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to a network node • <code>interfaceIndex</code> - interfaceIndex • <code>msg</code> - Pointer to Message • <code>lastHopAddr</code> - mac address <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
MAC_HandOffSuccessfullyReceivedPacket Pass packet successfully up to the network layer It is overloading	<p><code>void MAC_HandOffSuccessfullyReceivedPacket (Node* node, int interfaceIndex, Message* msg, MacHWAddress* lastHopAddress, int arpMessageType)</code></p> <p>Parameters:</p>

<p>function used for ARP packet</p>	<ul style="list-style-type: none"> • node - Pointer to a network node • interfaceIndex - interfaceIndex • msg - Pointer to Message • lastHopAddress - mac address • arpMessageType - Distinguish between ARP and general message <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
<p>MAC_HandOffSuccessfullyReceivedPacket</p> <p>Pass packet successfully up to the network layer It is overloading function used for ARP packet</p>	<p>void MAC_HandOffSuccessfullyReceivedPacket (Node* node, int interfaceIndex, Message* msg, Mac802Address* lastHopAddress, int arpMessageType)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to a network node • interfaceIndex - interfaceIndex • msg - Pointer to Message • lastHopAddress - mac address • arpMessageType - Distinguish between ARP and general message <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
<p>MAC_OutputQueueTopPacket</p> <p>To check packet at the top of output queue</p>	<p>BOOL MAC_OutputQueueTopPacket (Node* node, int interfaceIndex, Message*** msg, NodeAddress* nextHopAddress, int networkType, TosType* priority)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to a network node • interfaceIndex - interfaceIndex • msg - Pointer to Message • nextHopAddress - Next hop address • networkType - network type • priority - Message Priority <p>Returns:</p> <ul style="list-style-type: none"> • BOOL - TRUE if there is a packet, FALSE otherwise.
<p>MAC_OutputQueueTopPacket</p>	<p>BOOL MAC_OutputQueueTopPacket (Node* node, int interfaceIndex, Message*** msg,</p>

	<p>MacHWAddress* nextHopAddress, int networkType, TosType* priority)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to a network node • interfaceIndex - interfaceIndex • msg - Pointer to Message • nextHopAddress - Next hop address • networkType - network type • priority - Message Priority <p>Returns:</p> <ul style="list-style-type: none"> • BOOL - TRUE if there is a packet, FALSE otherwise.
MAC_OutputQueueTopPacket	<p>BOOL MAC_OutputQueueTopPacket (Node* node, int interfaceIndex, Message** msg, Mac802Address* nextHopAddress, int networkType, TosType* priority)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to a network node • interfaceIndex - interfaceIndex • msg - Pointer to Message • nextHopAddress - Next hop address • networkType - network type • priority - Message Priority <p>Returns:</p> <ul style="list-style-type: none"> • BOOL - TRUE if there is a packet, FALSE otherwise.
MAC_OutputQueueDequeuePacket	<p>BOOL MAC_OutputQueueDequeuePacket (Node* node, int interfaceIndex, Message** msg, NodeAddress* nextHopAddress, int networkType, TosType * priority)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to a network node • interfaceIndex - interfaceIndex • msg - Pointer to Message • nextHopAddress - Pointer to Node address • networkType - network type

	<ul style="list-style-type: none"> • <code>priority</code> - Pointer to queuing priority type <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - TRUE if dequeued successfully, FALSE otherwise.
MAC_OutputQueueDequeuePacket	<p>To remove packet from front of output queue</p> <p><code>BOOL MAC_OutputQueueDequeuePacket (Node* node, int interfaceIndex, Message** msg, MacHWAddress* nextHopAddress, int networkType, TosType * priority)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to a network node • <code>interfaceIndex</code> - interfaceIndex • <code>msg</code> - Pointer to Message • <code>nextHopAddress</code> - Pointer to Mac address • <code>networkType</code> - network type • <code>priority</code> - Pointer to queuing priority type <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - TRUE if dequeued successfully, FALSE otherwise.
MAC_OutputQueueDequeuePacket	<p>To remove packet from front of output queue, Its a overloaded function</p> <p><code>BOOL MAC_OutputQueueDequeuePacket (Node* node, int interfaceIndex, Message** msg, Mac802Address* nextHopAddress, int networkType, TosType * priority)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to a network node • <code>interfaceIndex</code> - interfaceIndex • <code>msg</code> - Pointer to Message • <code>nextHopAddress</code> - Pointer to MacAddress address • <code>networkType</code> - network type • <code>priority</code> - Pointer to queuing priority type <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - TRUE if dequeued successfully, FALSE otherwise.
MAC_OutputQueueDequeuePacket	<p>To remove packet(s) from front of output queue; process packets with options for example, packing multiple packets with same</p> <p><code>BOOL MAC_OutputQueueDequeuePacket (Node* node, int interfaceIndex, Message** msg, NodeAddress* nextHopAddress, int* networkType, TosType * priority, MacOutputQueueDequeueOption dequeueOption, MacOutputQueueDequeueCriteria dequeueCriteria, int * numFreeByte, int* numPacketPacked, TraceProtocolType tracePrt, BOOL eachWithMacHeader, int maxHeaderSize, BOOL returnPackedMsg)</code></p>

<p>next hop address together</p>	<p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to a network node • <code>interfaceIndex</code> - interfaceIndex • <code>msg</code> - Pointer to Message • <code>nextHopAddress</code> - Pointer to Node address • <code>networkType</code> - network type • <code>priority</code> - Pointer to queuing priority type • <code>dequeueOption</code> - option • <code>dequeueCriteria</code> - criteria • <code>numFreeByte</code> - number of bytes can be packed in 1 transmission • <code>numPacketPacked</code> - number of packets packed • <code>tracePrt</code> - Trace Protocol Type • <code>eachWithMacHeader</code> - Each msg has its own MAC header? • <code>maxHeaderSize</code> - max mac header size • <code>returnPackedMsg</code> - return Packed msg or a list of msgs <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - TRUE if dequeued successfully, FALSE otherwise.
<p>MAC_OutputQueueDequeuePacketForAPriority</p> <p>To remove packet(s) from front of output queue; process packets with options for example, packing multiple packets with same next hop address together</p>	<p><code>BOOL MAC_OutputQueueDequeuePacketForAPriority (Node* node, int interfaceIndex, int priority, Message** msg, NodeAddress* nextHopAddress, int* networkType, MacOutputQueueDequeueOption dequeueOption, MacOutputQueueDequeueCriteria dequeueCriteria, int * numFreeByte, int* numPacketPacked, TraceProtocolType tracePrt, BOOL eachWithMacHeader, int maxHeaderSize, BOOL returnPackedMsg)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to a network node • <code>interfaceIndex</code> - interfaceIndex • <code>priority</code> - Pointer to queuing priority type • <code>msg</code> - Pointer to Message • <code>nextHopAddress</code> - Pointer to Node address • <code>networkType</code> - network type • <code>dequeueOption</code> - option

	<ul style="list-style-type: none"> • <code>dequeueCriteria</code> - criteria • <code>numFreeByte</code> - number of bytes can be packed in 1 transmission • <code>numPacketPacked</code> - number of packets packed • <code>tracePrt</code> - Trace Protocol Type • <code>eachWithMacHeader</code> - Each msg has its own MAC header? • <code>maxHeaderSize</code> - max mac header size • <code>returnPackedMsg</code> - return Packed msg or a list of msgs <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - TRUE if dequeued successfully, FALSE otherwise.
MAC_IsMyUnicastFrame	<p>Check if a packet (or frame) belongs to this node Should be used only for four byte mac address</p> <p><code>BOOL MAC_IsMyUnicastFrame (Node* node, NodeAddress destAddr)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to a network node • <code>destAddr</code> - Destination Address <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - boolean
MAC_IsWiredNetwork	<p>To check if an interface is a wired interface</p> <p><code>BOOL MAC_IsWiredNetwork (Node* node, int interfaceIndex)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to a network node • <code>interfaceIndex</code> - interfaceIndex <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - boolean
MAC_IsPointToPointNetwork	<p>Checks if an interface belongs to Point to PointNetwork</p> <p><code>BOOL MAC_IsPointToPointNetwork (Node* node, int interfaceIndex)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to a network node • <code>interfaceIndex</code> - interfaceIndex <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - boolean

MAC_IsPointToMultiPointNetwork	<p>Checks if an interface belongs to Point to Multi-Point network.</p> <p>BOOL MAC_IsPointToMultiPointNetwork (Node* node, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to a network node • <code>interfaceIndex</code> - interfaceIndex <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - boolean
MAC_IsWiredBroadcastNetwork	<p>Determines if an interface is a wired broadcast interface</p> <p>BOOL MAC_IsWiredBroadcastNetwork (Node* node, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to a network node • <code>interfaceIndex</code> - interfaceIndex <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - boolean
MAC_IsWirelessNetwork	<p>Determine if a node's interface is a wireless interface</p> <p>BOOL MAC_IsWirelessNetwork (Node* node, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to a network node • <code>interfaceIndex</code> - interfaceIndex <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - boolean
MAC_IsWirelessAdHocNetwork	<p>Determine if a node's interface is a possible wireless ad hoc interface</p> <p>BOOL MAC_IsWirelessAdHocNetwork (Node* node, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to a network node • <code>interfaceIndex</code> - interfaceIndex <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - boolean
MAC_IsOneHopBroadcastNetwork	<p>Determines if an interface is a single Hop Broadcast interface</p> <p>BOOL MAC_IsOneHopBroadcastNetwork (Node* node, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to a network node • <code>interfaceIndex</code> - interfaceIndex

	<p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - boolean
MAC_IsASwitch	<p><code>BOOL MAC_IsASwitch (Node* node)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to a network node <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - boolean
MAC_SetVirtualMacAddress	<p><code>void MAC_SetVirtualMacAddress (Node* node, int interfaceIndex, NodeAddress virtualMacAddress)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to a network node • <code>interfaceIndex</code> - interface index • <code>virtualMacAddress</code> - MAC address <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
MacSetDefaultHWAddress	<p><code>void MacSetDefaultHWAddress (NodeId nodeId, MacHWAddress* macAddr, int interfaceIndex)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>nodeId</code> - Id of the input node • <code>macAddr</code> - Pointer to hardware structure • <code>interfaceIndex</code> - Interface on which the hardware address set <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
MAC_IsMyMacAddress	<p><code>NodeAddress MAC_IsMyMacAddress (Node* node, int interfaceIndex, NodeAddress destAddr)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to a network node • <code>interfaceIndex</code> - interface index • <code>destAddr</code> - dest address <p>Returns:</p>

	<ul style="list-style-type: none"> • NodeAddress - Node Address
MAC_IsMyHWAddress	<p>BOOL MAC_IsMyHWAddress (Node* node, int interfaceIndex, MacAddress* macAddr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Node pointer • interfaceIndex - Interface index • macAddr - Mac Address <p>Returns:</p> <ul style="list-style-type: none"> • BOOL - None
MacValidateAndSetHWAddress	<p>void MacValidateAndSetHWAddress (char* macAddrStr, MacHWAddress* macAddr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • macAddrStr - Pointer to address string • macAddr - Pointer to hardware address structure <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
DefaultMacHWAddressToIpv4Address	<p>NodeAddress DefaultMacHWAddressToIpv4Address (Node* node, MacHWAddress* macAddr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to Node structure • macAddr - Pointer to hardware address structure <p>Returns:</p> <ul style="list-style-type: none"> • NodeAddress - Ip address
MacGetHardwareLength	<p>void MacGetHardwareLength (Node* node, int interface, unsigned short hwLength)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to Node structure • interface - interface whose hardware length required • hwLength - Pointer to hardware string <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
MacGetHardwareType	void MacGetHardwareType (Node* node, int interface, unsigned short* type)

<p>Retrieve the Hardware Type.</p>	<p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to Node structure • <code>interface</code> - interface whose mac type requires • <code>type</code> - Pointer to hardware type <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
<p>MacGetHardwareAddressString</p> <p>Retrieve the Hardware Address String.</p>	<p>void MacGetHardwareAddressString (Node* node, int interface)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to Node structure • <code>interface</code> - interface whose hardware address retrieved <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
<p>MacAddNewInterface</p> <p>To add a new Interface at MAC</p>	<p>void MacAddNewInterface (Node* node, NodeAddress interfaceAddress, int numHostBits, int* interfaceIndex, const NodeInput nodeInput, char* macProtocolName)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to a network node • <code>interfaceAddress</code> - interface IP add • <code>numHostBits</code> - No of host bits • <code>interfaceIndex</code> - interface index • <code>nodeInput</code> - node input • <code>macProtocolName</code> - Mac protocol of interface <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
<p>MacAddVlanInfoForThisInterface</p> <p>Init and read VLAN configuration from user input for node and interface passed as arguments</p>	<p>void MacAddVlanInfoForThisInterface (Node* node, int* interfaceIndex, NodeAddress interfaceAddress, const NodeInput nodeInput)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to a network node • <code>interfaceIndex</code> - interface index

	<ul style="list-style-type: none"> • <code>interfaceAddress</code> - interface IP add • <code>nodeInput</code> - node input <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
MacReleaseVlanInfoForThisInterface	<p>NodeAddress MacReleaseVlanInfoForThisInterface (Node* node, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to a network node • <code>interfaceIndex</code> - interface index <p>Returns:</p> <ul style="list-style-type: none"> • <code>NodeAddress</code> - Node Address
MAC_IsBroadcastHWAddress	<p>BOOL MAC_IsBroadcastHWAddress (MacHWAddress* macAddr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>macAddr</code> - structure to hardware address <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - TRUE or FALSE
MAC_IsIdenticalHWAddress	<p>BOOL MAC_IsIdenticalHWAddress (MacHWAddress* macAddr1, MacHWAddress* macAddr2)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>macAddr1</code> - Pointer to harware address structure • <code>macAddr2</code> - Pointer to harware address structure <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - TRUE or FALSE
MAC_PrintHWAddr	<p>void MAC_PrintHWAddr (MacHWAddress* macAddr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>macAddr</code> - Mac address <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
MAC_PrintMacAddr	<p>void MAC_PrintMacAddr (Mac802Address* macAddr)</p> <p>Parameters:</p>

<p>Prints interface Mac Address</p>	<ul style="list-style-type: none"> • <code>macAddr</code> - Mac address <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
<p>MAC_RandFaultInit</p> <p>Initialization the Random Fault structure from input file</p>	<p><code>void MAC_RandFaultInit (Node* node, int interfaceIndex, const char* currentLine)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Node pointer • <code>interfaceIndex</code> - Interface index • <code>currentLine</code> - pointer to the input string <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
<p>MAC_RandFaultFinalize</p> <p>IPrint the statistics of Random link fault.</p>	<p><code>void MAC_RandFaultFinalize (Node* node, int interfaceIndex)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Node pointer • <code>interfaceIndex</code> - Interface index <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
<p>MAC_GetPacketsPriority</p> <p>Returns the priority of the packet</p>	<p><code>TosType MAC_GetPacketsPriority (Message* msg)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>msg</code> - Node Pointer <p>Returns:</p> <ul style="list-style-type: none"> • <code>TosType</code> - priority NOTE: DOT11e updates
<p>MAC_TranslateMulticastIPv4AddressToMulticastMacAddress</p> <p>Convert the Multicast ip address to multicast MAC address</p>	<p><code>void MAC_TranslateMulticastIPv4AddressToMulticastMacAddress (NodeAddress multicastAddress, MacHWAddress* macMulticast)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>multicastAddress</code> - Multicast ip address • <code>macMulticast</code> - Pointer to mac hardware address <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL

MAC_OutputQueuePeekByIndex Look at the packet at the index of the output queue.	<p>BOOL MAC_OutputQueuePeekByIndex (Node* node, int interfaceIndex, int msgIndex, Message** msg, NodeAddress nextHopAddress, TosType priority)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Node pointer • interfaceIndex - Interface index • msgIndex - Message index • msg - Double pointer to message • nextHopAddress - Next hop mac address • priority - priority <p>Returns:</p> <ul style="list-style-type: none"> • BOOL - TRUE if the messeage found, FALSE otherwise
MAC_OutputQueuePeekByIndex Look at the packet at the index of the output queue.	<p>BOOL MAC_OutputQueuePeekByIndex (Node* node, int interfaceIndex, int msgIndex, Message** msg, Mac802Address* nextHopAddress, TosType priority)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Node pointer • interfaceIndex - Interface index • msgIndex - Message index • msg - Double pointer to message • nextHopAddress - Next hop mac address • priority - priority <p>Returns:</p> <ul style="list-style-type: none"> • BOOL - TRUE if the messeage found, FALSE otherwise
MAC_OutputQueuePeekByIndex Look at the packet at the index of the output queue.	<p>BOOL MAC_OutputQueuePeekByIndex (int interfaceIndex, int msgIndex, Message** msg, MacHWAddress* nextHopAddress, TosType priority)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • interfaceIndex - Interface index • msgIndex - Message index • msg - Double pointer to message • nextHopAddress - Next hop mac address

	<ul style="list-style-type: none"> • priority - priority <p>Returns:</p> <ul style="list-style-type: none"> • BOOL - TRUE if the message found, FALSE otherwise
MAC_OutputQueueDequeuePacketWithIndex	<p>To remove the packet at specified index output queue.</p> <p>BOOL MAC_OutputQueueDequeuePacketWithIndex (Node* node, int interfaceIndex, int msgIndex, Message** msg, NodeAddress nextHopAddress, int networkType)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Node pointer • interfaceIndex - Interface index • msgIndex - Message index • msg - Double pointer to message • nextHopAddress - Next hop IP address • networkType - Type of network <p>Returns:</p> <ul style="list-style-type: none"> • BOOL - TRUE if the message dequeued properly, FALSE otherwise
MAC_OutputQueueDequeuePacketWithIndex	<p>To remove the packet at specified index output queue.</p> <p>BOOL MAC_OutputQueueDequeuePacketWithIndex (Node* node, int interfaceIndex, int msgIndex, Message** msg, Mac802Address* nextHopMacAddress, int networkType)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Node pointer • interfaceIndex - Interface index • msgIndex - Message index • msg - Double pointer to message • nextHopMacAddress - Next hop mac address • networkType - Type of network <p>Returns:</p> <ul style="list-style-type: none"> • BOOL - TRUE if the message dequeued properly, FALSE otherwise
MAC_OutputQueueDequeuePacketWithIndex	<p>To remove the packet at specified index output queue.</p> <p>BOOL MAC_OutputQueueDequeuePacketWithIndex (Node* node, int interfaceIndex, int msgIndex, Message** msg, MacHWAddress nextHopMacAddress, int networkType)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Node pointer

	<ul style="list-style-type: none"> • <code>interfaceIndex</code> - Interface index • <code>msgIndex</code> - Message index • <code>msg</code> - Double pointer to message • <code>nextHopMacAddress</code> - Next hop mac address • <code>networkType</code> - Type of network <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - TRUE if the message dequeued properly, FALSE otherwise
MAC_IPv4addressIsMulticastAddress	<p>Check the given address is Multicast address or not.</p> <p>BOOL MAC_IPv4addressIsMulticastAddress (NodeAddress ipV4)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>ipV4</code> - ipV4 address <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - TRUE or FALSE
MAC_IsBroadcastMacAddress	<p>Checks Broadcast MAC address.</p> <p>BOOL MAC_IsBroadcastMacAddress (MacAddress* macAddr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>macAddr</code> - Mac Address. <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - None
IPv4AddressToDefaultMac802Address	<p>Retrieve the Mac802Address from IP address.</p> <p>void IPv4AddressToDefaultMac802Address (Node* node, int index, NodeAddress ipv4Address, Mac802Address* macAddr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to Node structure • <code>index</code> - Interface Index • <code>ipv4Address</code> - Ipv4 address from which the • <code>macAddr</code> - Pointer to Mac802address structure <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
ConvertVariableHWAddressTo802Address	<p>Bool ConvertVariableHWAddressTo802Address (Node* node, MacHWAddress* macHWAddr, Mac802Address* mac802Addr)</p>

<p>Convert Variable Hardware address to Mac 802 addtess</p>	<p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to Node structure • macHWAddr - Pointer to hardware address structure • mac802Addr - Pointer to mac 802 address structure <p>Returns:</p> <ul style="list-style-type: none"> • Bool - None
<p>MAC_CopyMacHWAddress</p> <p>Copies Hardware address address</p>	<p>void MAC_CopyMacHWAddress (MacHWAddress* destAddr, MacHWAddress* srcAddr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • destAddr - structure to destination hardware address • srcAddr - structure to source hardware address <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
<p>DefaultMac802AddressToIpv4Address</p> <p>Retrieve IP address from Mac802Address</p>	<p>NodeAddress DefaultMac802AddressToIpv4Address (Node* node, Mac802Address* macAddr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to Node structure • macAddr - Pointer to hardware address structure <p>Returns:</p> <ul style="list-style-type: none"> • NodeAddress - Ipv4 Address
<p>IPv4AddressToHWAddress</p> <p>Converts IP address To MacHWAddress</p>	<p>BOOL IPv4AddressToHWAddress (Node* node, int interfaceIndex, Message* msg, NodeAddress ipv4Address)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to Node structure • interfaceIndex - interfcae index of a node • msg - Message pointer • ipv4Address - Ipv4 address from which the <p>Returns:</p> <ul style="list-style-type: none"> • BOOL - Returns False when conversion fails
<p>MacHWAddressToIpv4Address</p>	<p>NodeAddress MacHWAddressToIpv4Address (Node * node, int interfaceIndex, MacHWAddress* macAddr)</p>

<p>This function converts variable length Mac address to IPv4 address. It checks the type of hardware address and based on that conversion is done.</p>	<p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node which indicates the host • interfaceIndex - Interface index of a node • macAddr - Pointer to MacHWAddress Structure. <p>Returns:</p> <ul style="list-style-type: none"> • NodeAddress - IP address
<p>decToHex</p> <p>Convert one byte decimal number to hex number.</p>	<p>char* decToHex (int dec)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • dec - decimal number <p>Returns:</p> <ul style="list-style-type: none"> • char* - return correspondig hex digit string for one byte decimal number
<p>MAC_FourByteMacAddressToVariableHWAddress</p>	<p>void MAC_FourByteMacAddressToVariableHWAddress (Node * node, int interfaceIndex, MacHWAddress * macAddr, NodeAddress nodeAddr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node which indicates the host • interfaceIndex - Interface index of a node • macAddr - Pointer to source MacHWAddress Structure • nodeAddr - Ip address <p>Returns:</p> <ul style="list-style-type: none"> • void - None
<p>MAC_VariableHWAddressToFourByteMacAddress</p> <p>Retrieve IP address from MacHWAddress of type IPV4_LINKADDRESS</p>	<p>NodeAddress MAC_VariableHWAddressToFourByteMacAddress (Node* node, MacHWAddress* macAddr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to Node structure • macAddr - Pointer to hardware address structure <p>Returns:</p> <ul style="list-style-type: none"> • NodeAddress - Ipv4 Address
<p>GetBroadCastAddress</p>	<p>MacHWAddress GetBroadCastAddress (Node* node, int interfaceIndex)</p> <p>Parameters:</p>

	<p>Returns Broadcast Address of an interface</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to a node • <code>interfaceIndex</code> - Interface of a node <p>Returns:</p> <ul style="list-style-type: none"> • <code>MacHWAddress</code> - Broadcast mac address of a interface
<p>GetMacHWAddress</p> <p>Returns MacHWAddress of an interface</p>	<p>MacHWAddress GetMacHWAddress (<code>Node* node, int interfaceIndex</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to a node • <code>interfaceIndex</code> - inetrface of a node <p>Returns:</p> <ul style="list-style-type: none"> • <code>MacHWAddress</code> - Mac address of a interface
<p>MacGetInterfaceIndexFromMacAddress</p> <p>Returns interfaceIndex at which Macaddress is configured</p>	<p>int MacGetInterfaceIndexFromMacAddress (<code>Node* node, MacHWAddress macAddr</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to a node • <code>macAddr</code> - Mac Address of a node <p>Returns:</p> <ul style="list-style-type: none"> • <code>int</code> - interfaceIndex of node
<p>MacGetInterfaceIndexFromMacAddress</p> <p>Returns interfaceIndex at which Macaddress is configured</p>	<p>int MacGetInterfaceIndexFromMacAddress (<code>Node* node, Mac802Address macAddr</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to a node • <code>macAddr</code> - Mac Address of a node <p>Returns:</p> <ul style="list-style-type: none"> • <code>int</code> - interfaceIndex of node
<p>MacGetInterfaceIndexFromMacAddress</p> <p>Returns interfaceIndex at which Macaddress is configured</p>	<p>int MacGetInterfaceIndexFromMacAddress (<code>Node* node, NodeAddress macAddr</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to a node • <code>macAddr</code> - Mac Address of a node <p>Returns:</p>

	<ul style="list-style-type: none"> • <code>int</code> - interfaceIndex of node
MAC_Reset	<p>Reset the Mac protocols use by the node</p> <p><code>void MAC_Reset (Node* node, int InterfaceIndex)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to the node • <code>InterfaceIndex</code> - interface index <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
MAC_AddResetFunctionList	<p>Add which protocols in the Mac layer to be reset to a fuction list pointer.</p> <p><code>void MAC_AddResetFunctionList (Node* node, int InterfaceIndex, void* param)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to the node • <code>InterfaceIndex</code> - interface index • <code>param</code> - pointer to the protocols reset function <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None



Contact [QualNet Support](#) for questions pertaining to the QualNet API Reference. This document is confidential and proprietary.
It may not be reproduced or distributed without the expressed written consent of [Scalable Network Technologies](#).

QualNet® is a Registered Trademark of [Scalable Network Technologies](#).

Copyright © 2001-2012 [Scalable Network Technologies, Inc.](#) All rights reserved.

QualNet 6.1 API Reference

MAIN

This file contains some common definitions.

Constant / Data Structure Summary

Type	Name
CONSTANT	MAX_NUM_PHYS Maximum number of Physical channel
CONSTANT	MAX_NUM_INTERFACES Maximum number of Interfaces.
CONSTANT	PROTOCOL_TYPE_IP Length Field value for protocol IP TYPE
CONSTANT	PROTOCOL_TYPE_ARP ARP type
CONSTANT	ANY_DEST This is a special addresses used in the MAC and network layers. It defines any destination.
CONSTANT	ANY_MAC802 This is a special addresses used in the MAC and network layers. It defines any destination of six byte.
CONSTANT	INVALID_802ADDRESS This is a special addresses used in the MAC and network layers. It is used for invalid address
CONSTANT	ANY_SOURCE_ADDR This is a special addresses used in the MAC and network layers. It defines any source.
CONSTANT	ANY_IP

	This is a special addresses used in the MAC and network layers. It defines any IP.
CONSTANT	ANY_INTERFACE
	This is a special addresses used in the MAC and network layers. It defines any Interface.
CONSTANT	CPU_INTERFACE
	This is a special addresses used in the MAC and network layers. It defines CPU Interface.
CONSTANT	INVALID_ADDRESS
	It defines Invalid Address. Used only by mac/mac_802_11.c.
CONSTANT	MAX_STRING_LENGTH
	Generic maximum length of a string. The maximum length of any line in the input file is 3x this value.
CONSTANT	BIG_STRING_LENGTH
	maximum length of a string.
CONSTANT	MAX_CLOCK_STRING_LENGTH
	Generic maximum length of a clock string.
CONSTANT	MAX_NW_PKT_SIZE
	Defines the Maximum Network Packet Size which can handled by the physical network. In QualNet, its value is 2048. Packets larger than this will be fragmented by IP.
CONSTANT	MIN_NW_PKT_SIZE
	Defines the Minimum Network Packet Size which can be handled by the physical network. In QualNet, its value is 40. Packets smaller than this will not have room for transport headers and most firewall-type devices will drop an initial fragment that does not contain enough data to hold the transport headers.
CONSTANT	MIN_IPV6_PKT_SIZE
	Defines the Minimum Network Packet Size which can be handled by the IPv6 physical network. In QualNet, its value is 60. Packets smaller than this will not have room for transport headers and most firewall-type devices will drop an initial fragment that does not contain enough data to hold the transport headers. The additional space is to allow for IPv6's larger headers.
ENUMERATION	NetworkType

	Enlisted different network type
ENUMERATION	enum for the various layers in QualNet. New layers added to the simulation should be added here as well. Used by models at all layers in the protocol stack to mark newly created messages to be destined to the right layer/module.
STRUCT	in6_addr Describes the IPv6 address
STRUCT	AtmAddress Describes the ATM address
STRUCT	Address Describes the address structure which contains the interface address and network type

Function / Macro Summary

Return Type	Summary
MACRO	MAX(x, y) Utility function MAX. Calculates the Maximum one from two given numbers.
MACRO	MIN(x, y) Utility function MIN. Calculates the Minimum one from two given numbers.
MACRO	ABS(x) Utility function ABS. Return the absolute value of a given number.
MACRO	IN_DB(x) Utility function, decibel converter. Performs the 10 base log operation on the given number and then multiply with 10.
MACRO	NON_DB(x) Utility function, decibel converter. Performs power operation on the given number.
MACRO	MEM_malloc

	Adds filename and line number parameters to the MEM_malloc function
NodeAddress	GetIPv4Address (Address addr)
	Get IPv4 address from generic address
in6_addr	GetIPv6Address (Address addr)
	Get IPv6 address from generic address
void	SetIPv4AddressInfo (Address address, NodeAddress addr)
	Set IPv4 address and network type to generic address
void	SetIPv6AddressInfo (Address address, in6_addr addr)
	Set IPv6 address and network type to generic address
int	RoundToInt (double x)
	Round a float point number to an integer. This function tries to get consistent value on different platforms
void*	MEM_malloc (size_t size, char* filename, int lineno)
	Allocates memory block of a given size.
void	MEM_free (void* ptr)
	Deallocates the memory in turn it calls free().
UInt8	maskChar (UInt8 sposition, UInt8 eposition)
	Return 1's in all bit positions between sposition and eposition
UInt16	maskShort (UInt16 sposition, UInt16 eposition)
	Return 1's in all bit positions between sposition and eposition
UInt32	maskInt (int sposition, int eposition)
	Return 1's in all bit positions between sposition and eposition
UInt8	LshiftChar (UInt8 x, UInt8 eposition)

	Left shifts data where eposition determines the position of the last bit after the shift and (size-eposition) determines the number of bits to be shifted
UInt16	LshiftShort (UInt16 x, UInt16 eposition)
	Left shifts data where eposition determines the position of the last bit after the shift and (size-eposition) determines the number of bits to be shifted
UInt32	LshiftInt (UInt32 x, int eposition)
	Left shifts data where eposition determines the position of the last bit after the shift and (size-eposition) determines the number of bits to be shifted
UInt8	RshiftChar (UInt8 x, UInt8 eposition)
	Right shifts data where eposition determines the position of the last bit after the shift and (size-eposition) determines the number of bits to be shifted
UInt16	RshiftShort (UInt16 x, UInt16 eposition)
	Right shifts data where eposition determines the position of the last bit after the shift and (size-eposition) determines the number of bits to be shifted
UInt32	RshiftInt (UInt32 x, int eposition)
	Right shifts data where eposition determines the position of the last bit after the shift and (size-eposition) determines the number of bits to be shifted

Constant / Data Structure Detail

Constant	MAX_NUM_PHYS 64 Maximum number of Physical channel
Constant	MAX_NUM_INTERFACES 96 Maximum number of Interfaces.
Constant	PROTOCOL_TYPE_IP 0x0800

Length Field value for protocol IP TYPE	
Constant	PROTOCOL_TYPE_ARP 0x0806 ARP type
Constant	ANY_DEST 0xffffffff This is a special addresses used in the MAC and network layers. It defines any destination.
Constant	ANY_MAC802 0xffffffffffff This is a special addresses used in the MAC and network layers. It defines any destination of six byte.
Constant	INVALID_802ADDRESS 0xffffffffffffe This is a special addresses used in the MAC and network layers. It is used for invalid address
Constant	ANY_SOURCE_ADDR 0xffffffff This is a special addresses used in the MAC and network layers. It defines any source.
Constant	ANY_IP 0xffffffff This is a special addresses used in the MAC and network layers. It defines any IP.
Constant	ANY_INTERFACE -1 This is a special addresses used in the MAC and network layers. It defines any Interface.
Constant	CPU_INTERFACE -2 This is a special addresses used in the MAC and network layers. It defines CPU Interface.
Constant	INVALID_ADDRESS 987654321 It defines Invalid Address. Used only by mac/mac_802_11.c.
Constant	MAX_STRING_LENGTH 200 Generic maximum length of a string. The maximum length of any line in the input file is 3x this value.
Constant	BIG_STRING_LENGTH 512

	maximum length of a string.
Constant	<pre>MAX_CLOCK_STRING_LENGTH 24</pre> <p>Generic maximum length of a clock string.</p>
Constant	<pre>MAX_NW_PKT_SIZE 2048</pre> <p>Defines the Maximum Network Packet Size which can handled by the physical network. In QualNet, its value is 2048. Packet larger than this will be fragmented by IP.</p>
Constant	<pre>MIN_NW_PKT_SIZE 40</pre> <p>Defines the Minimum Network Packet Size which can be handled by the physical network. In QualNet, its value is 40. Packets smaller than this will not have room for transport headers and most firewall-type devices will drop an initial fragment that does not contain enough data to hold the transport headers.</p>
Constant	<pre>MIN_IPv6_PKT_SIZE 60</pre> <p>Defines the Minimum Network Packet Size which can be handled by the IPv6 physical network. In QualNet, its value is 60. Packets smaller than this will not have room for transport headers and most firewall-type devices will drop an initial fragment that does not contain enough data to hold the transport headers. The additional space is to allow for IPv6's larger headers.</p>
Enumeration	<p>NetworkType</p> <p>Enlisted different network type</p>
Enumeration	enum for the various layers in QualNet. New layers added to the simulation should be added here as well. Used by models at all layers in the protocol stack to mark newly created messages to be destined to the right layer/module.
Structure	<pre>in6_addr</pre> <p>Describes the IPv6 address</p>
Structure	<pre>AtmAddress</pre> <p>Describes the ATM address</p>
Structure	Address

Describes the address structure which contains the interface address and network type

Function / Macro Detail

Function / Macro	Format
MAX(X, Y)	Utility function MAX. Calculates the Maximum one from two given numbers.
MIN(X, Y)	Utility function MIN. Calculates the Minimum one from two given numbers.
ABS(X)	Utility function ABS. Return the absolute value of a given number.
IN_DB(x)	Utility function, decibel converter. Performs the 10 base log operation on the given number and then multiply with 10.
NON_DB(x)	Utility function, decibel converter. Performs power operation on the given number.
MEM_malloc	Adds filename and line number parameters to the MEM_malloc function
GetIPv4Address Get IPv4 address from generic address	<p>NodeAddress GetIPv4Address (Address addr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>addr</code> - generic address. <p>Returns:</p> <ul style="list-style-type: none"> • <code>NodeAddress</code> - IPv4 address
GetIPv6Address Get IPv6 address from generic address	<p>in6_addr GetIPv6Address (Address addr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>addr</code> - generic address. <p>Returns:</p> <ul style="list-style-type: none"> • <code>in6_addr</code> - IPv6 address
SetIPv4AddressInfo Set IPv4 address and network type to generic address	<p>void SetIPv4AddressInfo (Address address, NodeAddress addr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>address</code> - generic address. • <code>addr</code> - IPv4 interface address.

	<p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
SetIPv6AddressInfo	<p>void SetIPv6AddressInfo (Address address, in6_addr addr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • address - generic address. • addr - IPv6 interface address. <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
RoundToInt	<p>int RoundToInt (double x)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • x - The float point number to be rounded <p>Returns:</p> <ul style="list-style-type: none"> • int - Returns the rounded integer
MEM_malloc	<p>void* MEM_malloc (size_t size, char* filename, int lineno)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • size - Size of the memory block to be allocated. • filename - Name of file allocating the memory • lineno - Line in the file where the API is called <p>Returns:</p> <ul style="list-style-type: none"> • void* - Returns the pointer of allocated memory otherwise NULL if allocation fails.
MEM_free	<p>void MEM_free (void* ptr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • ptr - Pointer of memory to be freed. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
maskChar	<p>UInt8 maskChar (UInt8 sposition, UInt8 eposition)</p> <p>Parameters:</p>

<p>Return 1's in all bit positions between sposition and eposition</p>	<ul style="list-style-type: none"> • <code>sposition</code> - starting bit position • <code>eposition</code> - last bit position set to 1 <p>Returns:</p> <ul style="list-style-type: none"> • <code>UInt8</code> - None
<p>maskShort</p> <p>Return 1's in all bit positions between sposition and eposition</p>	<p><code>UInt16 maskShort (UInt16 sposition, UInt16 eposition)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>sposition</code> - starting bit position • <code>eposition</code> - last bit position set to 1 <p>Returns:</p> <ul style="list-style-type: none"> • <code>UInt16</code> - None
<p>maskInt</p> <p>Return 1's in all bit positions between sposition and eposition</p>	<p><code>UInt32 maskInt (int sposition, int eposition)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>sposition</code> - starting bit position • <code>eposition</code> - last bit position set to 1 <p>Returns:</p> <ul style="list-style-type: none"> • <code>UInt32</code> - None
<p>LshiftChar</p> <p>Left shifts data where eposition determines the position of thelast bit after the shift and (size-eposition) determines the number of bits to be shifted</p>	<p><code>UInt8 LshiftChar (UInt8 x, UInt8 eposition)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>x</code> - the data to be shifted • <code>eposition</code> - last bit position set to 1 <p>Returns:</p> <ul style="list-style-type: none"> • <code>UInt8</code> - None
<p>LshiftShort</p> <p>Left shifts data where eposition determines the position of thelast bit after the shift and (size-eposition) determines the number of bits to be shifted</p>	<p><code>UInt16 LshiftShort (UInt16 x, UInt16 eposition)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>x</code> - the data to be shifted • <code>eposition</code> - last bit position set to 1 <p>Returns:</p> <ul style="list-style-type: none"> • <code>UInt16</code> - None

LshiftInt Left shifts data where eposition determines the position of the last bit after the shift and (size-eposition) determines the number of bits to be shifted	<p>UInt32 LshiftInt (UInt32 x, int eposition)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • x - the data to be shifted • eposition - last bit position set to 1 <p>Returns:</p> <ul style="list-style-type: none"> • UInt32 - None
RshiftChar Right shifts data where eposition determines the position of the last bit after the shift and (size-eposition) determines the number of bits to be shifted	<p>UInt8 RshiftChar (UInt8 x, UInt8 eposition)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • x - the data to be shifted • eposition - last bit position set to 1 <p>Returns:</p> <ul style="list-style-type: none"> • UInt8 - None
RshiftShort Right shifts data where eposition determines the position of the last bit after the shift and (size-eposition) determines the number of bits to be shifted	<p>UInt16 RshiftShort (UInt16 x, UInt16 eposition)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • x - the data to be shifted • eposition - last bit position set to 1 <p>Returns:</p> <ul style="list-style-type: none"> • UInt16 - None
RshiftInt Right shifts data where eposition determines the position of the last bit after the shift and (size-eposition) determines the number of bits to be shifted	<p>UInt32 RshiftInt (UInt32 x, int eposition)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • x - the data to be shifted • eposition - last bit position set to 1 <p>Returns:</p> <ul style="list-style-type: none"> • UInt32 - None



Contact [QualNet Support](#) for questions pertaining to the QualNet API Reference. This document is confidential and proprietary.
It may not be reproduced or distributed without the expressed written consent of [Scalable Network Technologies](#).

QualNet® is a Registered Trademark of [Scalable Network Technologies](#).

Copyright © 2001-2012 [Scalable Network Technologies, Inc.](#). All rights reserved.

QualNet 6.1 API Reference

MAPPING

This file describes data structures and functions for mapping between node pointers, node identifiers, and node addresses.

Constant / Data Structure Summary

Type	Name
CONSTANT	INVALID_MAPPING Indicates Invalid Mapping
CONSTANT	MAX_INTERFACE_ADDRESSES max no of addressees assigned to an interface
CONSTANT	NODE_HASH_SIZE Defines node hash size. Hashes the nodeId using a mod NODE_HASH_SIZE hash.
STRUCT	NetworkProperty Describes the property of a network.
STRUCT	AddressMappingType Describes the type of address mapping.
STRUCT	AddressReverseMappingType Describes the type of reverse address mapping.
STRUCT	SubnetListType Used to determine what the next address counter should be for each subnet address. This is needed to allow different SUBNET/LINK statements to declare the same subnet address.
STRUCT	AddressMapType Describes the detailed information of Node ID <--> IP address mappings.

STRUCT	nodeIdToNodePtr
	Describes the nodeId and corresponding nodePtr.

Function / Macro Summary

Return Type	Summary
MACRO	MADDR6_SCOPE(a) Multicast Address Scope.
MACRO	IS_MULTIADDR6(a) Checks whether an address is multicast address.
MACRO	CLR_ADDR6(a) Set an address with 0 values.
MACRO	IS_CLR_ADDR6(a) Does an address have the value of 0 (Cleared).
MACRO	COPY_ADDR6(from, to) Copies from-ipv6 address to to-ipv6 address.
MACRO	SAME_ADDR6(a, b) Checks if a and b address is same address.
MACRO	IS_ANYADDR6(a) Checks whether the address is any address or not.
MACRO	IS_LOOPADDR6(a) Checks whether it is loopback address.
MACRO	CMP_ADDR6(a, b)

	Compaires two addresses.
MACRO	<u>IS_IPV4ADDR6(a)</u>
	Checks whether it is ipv4 address.
MACRO	<u>IS_LOCALADDR6(a)</u>
	Checks whether it is local address.
MACRO	<u>IS_LINKLADDR6(a)</u>
	Checks whether it is link local address.
MACRO	<u>IS_SITELADDR6(a)</u>
	Checks whether it is site local address.
MACRO	<u>SAME_ADDR4(a, b)</u>
	Checks whether IPv4 addresses match.
MACRO	<u>IS_ANYADDR4(a)</u>
	Checks whether IPv4 address is ANY_DEST.
BOOL	<u>Address_IsSameAddress(Address* addr1 addr1, Address* addr2 addr2)</u>
	Check whether both addresses(i.e. addr1 and addr2) are same.
BOOL	<u>Address_IsAnyAddress(Address* addr addr)</u>
	Check whether addr is any address of the same type
BOOL	<u>Address_IsMulticastAddress(Address* addr addr)</u>
	Check whether addr is a multicast address
BOOL	<u>Address_IsSubnetBroadcastAddress(Node* node node, Address* addr addr)</u>
	Check whether addr is a subnet broadcast address
void	<u>Address_SetToAnyAddress(Address* addr addr, Address* refAddr refAddr)</u>
	Set addr to any address of the same type as refAddr.

void	Address_AddressCopy (Address* dstAddress, Address* srcAddress)
	Copy srcAddress to dstAddress
int	Ipv6CompareAddr6 (in6_addr a, in6_addr b)
	Compairs to ipv6 address. if a is greater than b then returns positive, if equals then 0, a is smaller then b then negative.
BOOL	Ipv6IsAddressInNetwork (const in6_addr* globalAddr, unsigned int tla, unsigned int vla, unsigned int sla)
	Checks whether the address is in the same network. : if in the same network then returns TRUE, otherwise FALSE.
BOOL	Ipv6IsAddressInNetwork (const in6_addr* globalAddr, const in6_addr* ipv6SubnetAddr, unsigned int prefixLenth)
	Checks whether the address is in the same network. : if in the same network then returns TRUE, otherwise FALSE.
BOOL	Ipv6CheckNetworkParams (unsigned int tla tla, unsigned int nla nla, unsigned int sla sla)
	Checks network parameters (tla, nla, sla)
void	MAPPING_HashNodeId (IdToNodePtrMap* hash, NodeAddress nodeId, Node* nodePtr)
	Hashes the nodeIds using a mod NODE_HASH_SIZE hash. This is not thread safe.
Node*	MAPPING_GetNodePtrFromHash (IdToNodePtrMap* hash, NodeAddress nodeId)
	Retrieves the node pointer for nodeId from hash.
AddressMapType*	MAPPING_MallocAddressMap ()
	Allocates memory block of size AddressMapType.
void	MAPPING_InitAddressMap (AddressMapType* map)
	Initializes the AddressMapType structure.
void	MAPPING_BuildAddressMap (const NodeInput* nodeInput, NodeAddress** nodeIdArrayPtr, AddressMapType* map)
	Builds the address map
NodeAddress	MAPPING_GetInterfaceAddressForSubnet (Node* node, NodeAddress nodeId, NodeAddress subnetAddress, int numHostBits)
	Gives Interface address for a Subnet.
NodeAddress	MAPPING_GetInterfaceAddressForSubnet (const AddressMapType* map, NodeAddress nodeId, NodeAddress subnetAddress,

	<pre>int numHostBits)</pre>
	Gives Interface address for a Subnet.
NodeAddress	<code>MAPPING_GetSubnetAddressForInterface</code> (Node* node, NodeAddress nodeId, int interfaceIndex)
	Gives the Subnet address for an interface.
NodeAddress	<code>MAPPING_GetSubnetMaskForInterface</code> (Node* node, NodeAddress nodeId, int interfaceIndex)
	Gives the Subnet mask for an interface.
int	<code>MAPPING_GetNumHostBitsForInterface</code> (Node* node, NodeAddress nodeId, int interfaceIndex)
	Gives the number of host bits for an interface.
void	<code>MAPPING_GetInterfaceInfoForInterface</code> (Node* node, NodeAddress nodeId, int interfaceIndex, NodeAddress* interfaceAddress, NodeAddress* subnetAddress, NodeAddress* subnetMask, int* numHostBits)
	Gives the Interface information for an interface.
NodeAddress	<code>MAPPING_GetInterfaceAddressForInterface</code> (Node* node, NodeAddress nodeId, int interfaceIndex)
	Gives the Interface address for an interface.
Address	<code>MAPPING_GetInterfaceAddressForInterface</code> (NetworkType netType, int relativeInfInx)
	Get node interface Address according to the network specific interface index. Overloaded function for ATM compatibility.
NodeAddress	<code>MAPPING_GetNodeIdFromInterfaceAddress</code> (Node* node, NodeAddress interfaceAddress)
	Gives Node id from an interface address.
NodeAddress	<code>MAPPING_GetNodeIdFromInterfaceAddress</code> (Node* node, Address interfaceAddress)
	Gives Node id from an interface address. Overloaded for IPv6
NodeAddress	<code>MAPPING_GetDefaultInterfaceAddressFromNodeId</code> (Node* node, NodeAddress nodeId)
	Gives default interface address from a node id.
unsigned int	<code>MAPPING_GetNumNodesInSubnet</code> (Node* node, NodeAddress subnetAddress)
	Gives the number of nodes in a subnet.

unsigned int	MAPPING_GetSubnetAddressCounter (AddressMapType* map, NodeAddress subnetAddress)
	Gives the subnet address counter.
void	MAPPING_UpdateSubnetAddressCounter (AddressMapType* map, NodeAddress subnetAddress, int addressCounter)
	Updates the subnet address counter.
int	MAPPING_GetInterfaceIndexFromInterfaceAddress (Node* node, NodeAddress interfaceAddress)
	Gets the node's interface index for the given address.
Address	MAPPING_GetNodeInfoFromAtmNetInfo (unsigned int* index, unsigned int* genIndex)
	Get node interface Address, generic interfaceIndex and Atm related interfaceIndex from ATM Network information.
unsigned int	MAPPING_GetInterfaceIdForDestAddress (Node* node, NodeId nodeId, NodeAddress destAddr)
	For a given destination address find its interface index
NodeAddress	MAPPING_GetSubnetMaskForDestAddress (Node* node, NodeId nodeId, NodeAddress destAddr)
	For a given nodeId & destination address find the subnet mask for the associated network
NodeAddress	MAPPING_GetInterfaceAddrForNodeIdAndIntfId (Node* node, NodeId nodeId, int intfId)
	For a given nodeId & InterfaceId find the associated IP-Address
unsigned int	MAPPING_GetIPv6NetworkAddressCounter (AddressMapType* map, in6_addr subnetAddr, unsigned int subnetPrefixLen)
	Get IPV6 network address counter.
void	MAPPING_UpdateIPv6NetworkAddressCounter (AddressMapType* map, in6_addr subnetAddr, unsigned int subnetPrefixLen, int addressCounter)
	Update IPV6 network address counter.
unsigned int	MAPPING_GetNumNodesInIPv6Network (Node* node, in6_addr subnetAddr, unsigned int subnetPrefixLen)
	Get Num of nodes in IPV6 network.
NetworkType	MAPPING_GetNetworkIPVersion (const char* addrString)
	Get Network version IPv4/IPv6.

NetworkType	<code>MAPPING_GetNetworkType(const char* addrString)</code>
	Identify network type from addrString.
void	<code>MAPPING_GetIpv6InterfaceInfoForInterface(Node *node node, NodeId nodeId nodeId, int interfaceIndex, in6_addr* globalAddr, in6_addr* subnetAddr, unsigned int* subnetPrefixLen)</code>
	Get IPV6 interface information for a interface.
BOOL	<code>MAPPING_GetIpv6GlobalAddress(Node *node node, NodeId nodeId nodeId, unsigned int tla, unsigned int nla, unsigned int sla, in6_addr * addr6)</code>
	Get IPV6 global address.
BOOL	<code>MAPPING_GetIpv6GlobalAddressForInterface(Node * node, NodeId nodeId, int interfaceIndex, in6_addr * addr6, BOOL isDeprecated)</code>
	Get IPV6 global address for a node's nth interface.
void	<code>MAPPING_CreateIpv6GlobalUnicastAddr(unsigned int tla, unsigned int nla, unsigned int sla, int addressCounter, in6_addr* globalAddr)</code>
	Create IPv6 Global Unicast Address from tla nla sla
void	<code>MAPPING_CreateIpv6GlobalUnicastAddr(AddressMapType * map, in6_addr IPv6subnetAddress, uunsiged int IPv6subnetPrefixLen, int addressCounter, in6_addr* globalAddr)</code>
	Create IPv6 Global Unicast Address.
void	<code>MAPPING_CreateIpv6LinkLocalAddr(in6_addr* globalAddr, in6_addr* linkLocalAddr, unsigned int subnetPrefixLen)</code>
	Create IPv6 link local Address.
void	<code>MAPPING_CreateIpv6SiteLocalAddr(in6_addr* globalAddr, in6_addr* siteLocalAddr, unsigned short siteCounter, unsigned int subnetPrefixLen)</code>
	Create IPv6 site local Address.
void	<code>MAPPING_CreateIpv6MulticastAddr(in6_addr* globalAddr, in6_addr* multicastAddr)</code>
	Create ipv6 multicast address.
void	<code>MAPPING_CreateIpv6SubnetAddr(unsigned int tla, unsigned int nla, unsigned int sla, unsigned int* IPv6subnetPrefixLen, in6_addr* IPv6subnetAddress)</code>
	create subnet addr for IPV6 address.
NodeId	<code>MAPPING_GetNodeIdFromGlobalAddr(Node * node, in6_addr* globalAddr)</code>

	Get node id from Global Address.
NodeId	MAPPING_GetNodeIdFromLinkLayerAddr (Node * node, NodeAddress linkLayerAddr)
	Get node id from Link layer Address.
NodeAddress	MAPPING_CreateIpv6LinkLayerAddr (unsigned int nodeId, int interfaceId)
	Create IPv6 link layer Address.
BOOL	MAPPING_IsIpv6AddressOfThisNode (Node* node, const NodeAddress nodeId, in6_addr* globalAddr)
	checks whether the ipv6 address is of this node.
BOOL	MAPPING_IsNodeInThisIpRange (Node* node, NodeId nodeId, NodeAddress startRange, NodeAddress endRange)
	checks whether the node is in given range of : Addresses.
BOOL	MAPPING_IsIpAddressOfThisNode (Node* node, const NodeAddress nodeId, NodeAddress addr)
	checks whether the ipv4 address is of this node.
BOOL	MAPPING_GetInterfaceAddressForSubnet (Node* node, NodeId nodeId, in6_addr* ipv6SubnetAddr, unsigned int prefixLenth, in6_addr* ipv6InterfaceAddr, int* interfaceIndex)
	Get interface address for subnet using ipv6 addr.
BOOL	MAPPING_GetInterfaceAddressForSubnet (const AddressMapType* map, NodeId nodeId, in6_addr* ipv6SubnetAddr, unsigned int prefixLenth, in6_addr* ipv6InterfaceAddr, int* interfaceIndex)
	Get interface address for subnet using ipv6 addr.
BOOL	MAPPING_GetInterfaceAddressForSubnet (Node* node node, NodeId nodeId nodeId, unsigned int tla, unsigned int nla, unsigned int sla, in6_addr* ipv6Addr, int* interfaceIndex)
	Get interface address for subnet using tla nla sla.
BOOL	; MAPPING_GetInterfaceAddressForSubnet (const AddressMapType* map, NodeId nodeId, unsigned int tla, unsigned int nla, unsigned int sla, in6_addr* ipv6Addr, int* interfaceIndex)
	Get interface address for subnet using tla nla sla.
int	MAPPING_GetInterfaceFromLinkLayerAddress (Node* node, const NodeAddress linkLayerAddr)

	Get interface from link layer address.
int	MAPPING_GetInterfaceIndexFromInterfaceAddress (Node* node, Address interfaceAddress)
	Get interface index from interface address.
BOOL	MAPPING_GetIpv6GlobalAddress (Node* node, NodeId nodeId, in6_addr subnetAddr, UInt32 prefixLen, in6_addr* addr6)
	Get ipv6 global address
Address	MAPPING_GetDefaultInterfaceAddressInfoFromNodeId (Node *node node, NodeAddress nodeId nodeId, NetworkType networktype networktype)
	Get default interface address based on network type
void	Mapping_AutoCreateIPv6SubnetAddress (NodeAddress ipAddress, subnetString)
	Create IPv6 Testing Address Prefix (RFC 2471)from : ipv4 address.
NodeAddress	MAPPING_GetSubnetAddressFromInterfaceAddress (Node *node node, NodeAddress interfaceAddress)
	Get subnet address from interface address.
BOOL	MAPPING_GetSubnetAddressFromInterfaceAddress (Node * node, in6_addr* ipv6InterfaceAddr, in6_addr* ipv6SubnetAddr)
	Get ipv6 network Prefix from interface address.
BOOL	MAPPING_GetPrefixLengthForInterfaceAddress (Node* node, in6_addr* ipv6InterfaceAddr, unsigned int prefixLenth)
	Get prefix length for interface address.
NetworkProtocolType	MAPPING_GetNetworkProtocolTypeForNode (NodeAddress nodeId, const NodeInput * nodeInput)
	Get Network Protocol Type for the node.

Constant / Data Structure Detail

Constant	INVALID_MAPPING 0xffffffff Indicates Invalid Mapping
Constant	MAX_INTERFACE_ADDRESSES 2

	max no of addressees assigned to an interface
Constant	NODE_HASH_SIZE 32 Defines node hash size. Hashes the nodelds using a mod NODE_HASH_SIZE hash.
Structure	NetworkProperty Describes the property of a network.
Structure	AddressMappingType Describes the type of address mapping.
Structure	AddressReverseMappingType Describes the type of reverse address mapping.
Structure	SubnetListType Used to determine what the next address counter should be for each subnet address. This is needed to allow different SUBNET/LINK statements to declare the same subnet address.
Structure	AddressMapType Describes the detailed information of Node ID <--> IP address mappings.
Structure	nodeIdToNodePtr Describes the nodeld and corresponding nodePtr.

Function / Macro Detail

Function / Macro	Format
MADDR6_SCOPE(a)	Multicast Address Scope.
IS_MULTIADDR6(a)	Checks whether an address is multicast address.

CLR_ADDR6(a)	Set an address with 0 values.
IS_CLR_ADDR6(a)	Does an address have the value of 0 (Cleared).
COPY_ADDR6(from, to)	Copies from-ipv6 address to to-ipv6 address.
SAME_ADDR6(a, b)	Checks if a and b address is same address.
IS_ANYADDR6(a)	Checks whether the address is any address or not.
IS_LOOPADDR6(a)	Checks whether it is loopback address.
CMP_ADDR6(a, b)	Compaires two addresses.
IS_IPV4ADDR6(a)	Checks whether it is ipv4 address.
IS_LOCALADDR6(a)	Checks whether it is local address.
IS_LINKLADDR6(a)	Checks whether it is link local address.
IS_SITELADDR6(a)	Checks whether it is site local address.
SAME_ADDR4(a, b)	Checks whether IPv4 addresses match.
IS_ANYADDR4(a)	Checks whether IPv4 address is ANY_DEST.
Address_IsSameAddress Check whether both addresses(i.e. addr1 and addr2) are same.	<p>BOOL Address_IsSameAddress (Address* addr1 addr1, Address* addr2 addr2)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • addr1 - Pointer to 1st address • addr2 - Pointer to 2nd address <p>Returns:</p> <ul style="list-style-type: none"> • BOOL - None
Address_IsAnyAddress	BOOL Address_IsAnyAddress (Address* addr addr)

	<p>Check whether addr is any address of the same type</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>addr</code> - Pointer to address <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - None
Address_IsMulticastAddress	<p>Check whether addr is a multicast address</p> <p>BOOL Address_IsMulticastAddress (Address* addr addr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>addr</code> - Pointer to address <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - None
Address_IsSubnetBroadcastAddress	<p>Check whether addr is a subnet broadcast address</p> <p>BOOL Address_IsSubnetBroadcastAddress (Node* node node, Address* addr addr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - pointer to node • <code>addr</code> - Pointer to address <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - None
Address_SetToAnyAddress	<p>Set addr to any address of the same type as refAddr.</p> <p>void Address_SetToAnyAddress (Address* addr addr, Address* refAddr refAddr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>addr</code> - Pointer to address • <code>refAddr</code> - Pointer to refAddr <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
Address_AddressCopy	<p>Copy srcAddress to dstAddress</p> <p>void Address_AddressCopy (Address* dstAddress, Address* srcAddress)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>dstAddress</code> - Destination address • <code>srcAddress</code> - Source address <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL

Ipv6CompareAddr6 Compairs to ipv6 address. if a is greater than b then returns positive, if equals then 0, a is smaller then b then negative.	<p>int Ipv6CompareAddr6 (in6_addr a, in6_addr b)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • a - ipv6 address. • b - ipv6 address. <p>Returns:</p> <ul style="list-style-type: none"> • int - None
Ipv6IsAddressInNetwork Checks whether the address is in the same network. : if in the same network then returns TRUE, otherwise FALSE.	<p>BOOL Ipv6IsAddressInNetwork (const in6_addr* globalAddr, unsigned int tla, unsigned int vla, unsigned int sla)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • globalAddr - Pointer to ipv6 address. • tla - Top level ipv6 address. • vla - Next level ipv6 address. • sla - Site local ipv6 address. <p>Returns:</p> <ul style="list-style-type: none"> • BOOL - None
Ipv6IsAddressInNetwork Checks whether the address is in the same network. : if in the same network then returns TRUE, otherwise FALSE.	<p>BOOL Ipv6IsAddressInNetwork (const in6_addr* globalAddr, const in6_addr* ipv6SubnetAddr, unsigned int prefixLenth)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • globalAddr - Pointer to ipv6 address. • ipv6SubnetAddr - Pointer to ipv6 subnet address. • prefixLenth - prefix length of the address. <p>Returns:</p> <ul style="list-style-type: none"> • BOOL - TRUE if the address is in the same network, FALSE otherwise
Ipv6CheckNetworkParams Checks network parameters (tla, nla, sla)	<p>BOOL Ipv6CheckNetworkParams (unsigned int tla tla, unsigned int nla nla, unsigned int sla sla)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • tla - Top level aggregation. • nla - Next level aggregation. • sla - Site level aggregaton. <p>Returns:</p>

	<ul style="list-style-type: none"> • BOOL - None
MAPPING_HashNodeId	<p>Hashes the nodeIds using a mod NODE_HASH_SIZE hash. This is not thread safe.</p> <p>void MAPPING_HashNodeId (IdToNodePtrMap* hash, NodeAddress nodeId, Node* nodePtr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • hash - IdToNodePtrMap pointer • nodeId - Node id. • nodePtr - Node poniter <p>Returns:</p> <ul style="list-style-type: none"> • void - None
MAPPING_GetNodePtrFromHash	<p>Retrieves the node pointer for nodeId from hash.</p> <p>Node* MAPPING_GetNodePtrFromHash (IdToNodePtrMap* hash, NodeAddress nodeId)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • hash - IdToNodePtrMap pointer • nodeId - Node id. <p>Returns:</p> <ul style="list-style-type: none"> • Node* - Node pointer for nodeId.
MAPPING_MallocAddressMap	<p>Allocates memory block of size AddressMapType.</p> <p>AddressMapType* MAPPING_MallocAddressMap ()</p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> • AddressMapType* - Pointer to a new AddressMapType structure.
MAPPING_InitAddressMap	<p>Initializes the AddressMapType structure.</p> <p>void MAPPING_InitAddressMap (AddressMapType* map)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • map - A pointer of type AddressMapType. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
MAPPING_BuildAddressMap	<p>Builds the address map</p> <p>void MAPPING_BuildAddressMap (const NodeInput* nodeInput, NodeAddress** nodeIdArrayPtr, AddressMapType* map)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • nodeInput - A pointer to const NodeInput. • nodeIdArrayPtr - A pointer to pointer of NodeAddress

	<ul style="list-style-type: none"> • <code>map</code> - A pointer of type <code>AddressMapType</code>. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
MAPPING_GetInterfaceAddressForSubnet	<p>NodeAddress MAPPING_GetInterfaceAddressForSubnet (<code>Node* node, NodeAddress nodeId, NodeAddress subnetAddress, int numHostBits</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - A pointer to node being initialized • <code>nodeId</code> - Node id • <code>subnetAddress</code> - Subnet address • <code>numHostBits</code> - Number of host bits <p>Returns:</p> <ul style="list-style-type: none"> • <code>NodeAddress</code> - Interface address for the subnet.
MAPPING_GetInterfaceAddressForSubnet	<p>NodeAddress MAPPING_GetInterfaceAddressForSubnet (<code>const AddressMapType* map, NodeAddress nodeId, NodeAddress subnetAddress, int numHostBits</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>map</code> - A pointer to address map • <code>nodeId</code> - Node id • <code>subnetAddress</code> - Subnet address • <code>numHostBits</code> - Number of host bits <p>Returns:</p> <ul style="list-style-type: none"> • <code>NodeAddress</code> - Interface address for the subnet.
MAPPING_GetSubnetAddressForInterface	<p>NodeAddress MAPPING_GetSubnetAddressForInterface (<code>Node* node, NodeAddress nodeId, int interfaceIndex</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - A pointer to node being initialized. • <code>nodeId</code> - Node id • <code>interfaceIndex</code> - Interface index <p>Returns:</p> <ul style="list-style-type: none"> • <code>NodeAddress</code> - Subnet address for an interface.
MAPPING_GetSubnetMaskForInterface	<p>NodeAddress MAPPING_GetSubnetMaskForInterface (<code>Node* node, NodeAddress nodeId, int interfaceIndex</code>)</p>

	<p>Gives the Subnet mask for an interface.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - A pointer to node being initialized. • <code>nodeId</code> - Node id • <code>interfaceIndex</code> - Interface index <p>Returns:</p> <ul style="list-style-type: none"> • <code>NodeAddress</code> - Subnet mask for an interface.
MAPPING_GetNumHostBitsForInterface	<p>int MAPPING_GetNumHostBitsForInterface (Node* node, NodeAddress nodeId, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - A pointer to node being initialized. • <code>nodeId</code> - Node id • <code>interfaceIndex</code> - Interface index <p>Returns:</p> <ul style="list-style-type: none"> • <code>int</code> - The number of host bits for an interface.
MAPPING_GetInterfaceInfoForInterface	<p>void MAPPING_GetInterfaceInfoForInterface (Node* node, NodeAddress nodeId, int interfaceIndex, NodeAddress* interfaceAddress, NodeAddress* subnetAddress, NodeAddress* subnetMask, int* numHostBits)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - A pointer to node being initialized. • <code>nodeId</code> - Node id • <code>interfaceIndex</code> - Interface index • <code>interfaceAddress</code> - Interface address, int pointer. • <code>subnetAddress</code> - Subnet address, NodeAddress pointer. • <code>subnetMask</code> - Subnet mask, NodeAddress pointer. • <code>numHostBits</code> - Number of host bits, int pointer. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
MAPPING_GetInterfaceAddressForInterface	<p>NodeAddress MAPPING_GetInterfaceAddressForInterface (Node* node, NodeAddress nodeId, int interfaceIndex)</p> <p>Parameters:</p>

	<ul style="list-style-type: none"> • <code>node</code> - A pointer to the node being initialized. • <code>nodeId</code> - Node id • <code>interfaceIndex</code> - Interface index <p>Returns:</p> <ul style="list-style-type: none"> • <code>NodeAddress</code> - Interface address for an interface.
MAPPING_GetInterfaceAddressForInterface	<p>Address MAPPING_GetInterfaceAddressForInterface (<code>NetworkType netType, int relativeInInx</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>netType</code> - Network type of the interface. • <code>relativeInInx</code> - Inrface index related to networkType. <p>Returns:</p> <ul style="list-style-type: none"> • <code>Address</code> - Return Address.
MAPPING_GetNodeIdFromInterfaceAddress	<p>NodeAddress MAPPING_GetNodeIdFromInterfaceAddress (<code>Node* node, NodeAddress interfaceAddress</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - A pointer to node being initialized. • <code>interfaceAddress</code> - Interface address <p>Returns:</p> <ul style="list-style-type: none"> • <code>NodeAddress</code> - None
MAPPING_GetNodeIdFromInterfaceAddress	<p>NodeAddress MAPPING_GetNodeIdFromInterfaceAddress (<code>Node* node, Address interfaceAddress</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - A pointer to node being initialized. • <code>interfaceAddress</code> - Interface address <p>Returns:</p> <ul style="list-style-type: none"> • <code>NodeAddress</code> - None
MAPPING_GetDefaultInterfaceAddressFromNodeId	<p>NodeAddress MAPPING_GetDefaultInterfaceAddressFromNodeId (<code>Node* node, NodeAddress nodeId</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - A pointer to node being initialized. • <code>nodeId</code> - Node id <p>Returns:</p>

	<ul style="list-style-type: none"> • <code>NodeAddress</code> - Default interface address from the node id.
MAPPING_GetNumNodesInSubnet	<p>unsigned int MAPPING_GetNumNodesInSubnet (Node* node, NodeAddress subnetAddress)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - A pointer to node being initialized. • <code>subnetAddress</code> - Subnet address <p>Returns:</p> <ul style="list-style-type: none"> • <code>unsigned int</code> - Number of nodes in a subnet.
MAPPING_GetSubnetAddressCounter	<p>unsigned int MAPPING_GetSubnetAddressCounter (AddressMapType* map, NodeAddress subnetAddress)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>map</code> - A pointer to AddressMapType. • <code>subnetAddress</code> - Subnet address <p>Returns:</p> <ul style="list-style-type: none"> • <code>unsigned int</code> - The subnet address counter.
MAPPING_UpdateSubnetAddressCounter	<p>void MAPPING_UpdateSubnetAddressCounter (AddressMapType* map, NodeAddress subnetAddress, int addressCounter)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>map</code> - A pointer to AddressMapType. • <code>subnetAddress</code> - Subnet address • <code>addressCounter</code> - Address counter <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
MAPPING_GetInterfaceIndexFromInterfaceAddress	<p>int MAPPING_GetInterfaceIndexFromInterfaceAddress (Node* node, NodeAddress interfaceAddress)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - A pointer to node being initialized. • <code>interfaceAddress</code> - Interface address <p>Returns:</p> <ul style="list-style-type: none"> • <code>int</code> - The interface index.
MAPPING_GetNodeInfoFromAtmNetInfo	Address MAPPING_GetNodeInfoFromAtmNetInfo (unsigned int* index, unsigned int* genIndex)

<p>Get node interface Address, generic interfaceIndex and Atm related interfaceIndex from ATM Network information.</p>	<p>Parameters:</p> <ul style="list-style-type: none"> • <code>index</code> - return atm related interface index of a • <code>genIndex</code> - return generic interface index of a node. <p>Returns:</p> <ul style="list-style-type: none"> • <code>Address</code> - Return valid ATM Address related to Network information if genIndex is not equal to -1.
<p>MAPPING_GetInterfaceIdForDestAddress</p> <p>For a given destination address find its interface index</p>	<p><code>unsigned int MAPPING_GetInterfaceIdForDestAddress (Node* node, NodeId nodeId, NodeAddress destAddr)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - A pointer to node being initialized. • <code>nodeId</code> - Node ID • <code>destAddr</code> - Destination address. <p>Returns:</p> <ul style="list-style-type: none"> • <code>unsigned int</code> - None
<p>MAPPING_GetSubnetMaskForDestAddress</p> <p>For a given nodeId & destination address find the subnet mask for the associated network</p>	<p><code>NodeAddress MAPPING_GetSubnetMaskForDestAddress (Node* node, NodeId nodeId, NodeAddress destAddr)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - A pointer to node being initialized. • <code>nodeId</code> - Node ID • <code>destAddr</code> - Destination address. <p>Returns:</p> <ul style="list-style-type: none"> • <code>NodeAddress</code> - None
<p>MAPPING_GetInterfaceAddrForNodeIdAndIntfId</p> <p>For a given nodeId & InterfaceId find the associated IP-Address</p>	<p><code>NodeAddress MAPPING_GetInterfaceAddrForNodeIdAndIntfId (Node* node, NodeId nodeId, int intfId)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - The pointer to the node. • <code>nodeId</code> - Node ID • <code>intfId</code> - Interface ID. <p>Returns:</p> <ul style="list-style-type: none"> • <code>NodeAddress</code> - None
<p>MAPPING_GetIPv6NetworkAddressCounter</p>	<p><code>unsigned int MAPPING_GetIPv6NetworkAddressCounter (AddressMapType* map, in6_addr subnetAddr,</code></p>

<p>Get IPV6 network address counter.</p>	<p><code>unsigned int subnetPrefixLen)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>map</code> - The address map. • <code>subnetAddr</code> - The IPv6 address. • <code>subnetPrefixLen</code> - The prefix length. <p>Returns:</p> <ul style="list-style-type: none"> • <code>unsigned int</code> - The current counter.
<p>MAPPING_UpdateIPv6NetworkAddressCounter</p> <p>Update IPV6 network address counter.</p>	<p><code>void MAPPING_UpdateIPv6NetworkAddressCounter (AddressMapType* map, in6_addr subnetAddr, unsigned int subnetPrefixLen, int addressCounter)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>map</code> - The address map. • <code>subnetAddr</code> - The IPv6 address. • <code>subnetPrefixLen</code> - The prefix length. • <code>addressCounter</code> - The new counter value. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
<p>MAPPING_GetNumNodesInIPv6Network</p> <p>Get Num of nodes in IPV6 network.</p>	<p><code>unsigned int MAPPING_GetNumNodesInIPv6Network (Node* node, in6_addr subnetAddr, unsigned int subnetPrefixLen)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - The pointer to the node. • <code>subnetAddr</code> - The IPv6 address. • <code>subnetPrefixLen</code> - The prefix length. <p>Returns:</p> <ul style="list-style-type: none"> • <code>unsigned int</code> - None
<p>MAPPING_GetNetworkIPVersion</p> <p>Get Network version IPv4/IPv6.</p>	<p><code>NetworkType MAPPING_GetNetworkIPVersion (const char* addrString)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>addrString</code> - The address string <p>Returns:</p> <ul style="list-style-type: none"> • <code>NetworkType</code> - None

MAPPING_GetNetworkType	<p>Identify network type from addrString.</p> <p>NetworkType MAPPING_GetNetworkType (const char* addrString)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>addrString</code> - The address string <p>Returns:</p> <ul style="list-style-type: none"> • <code>NetworkType</code> - None
MAPPING_GetIpv6InterfaceInfoForInterface	<p>Get IPV6 interface information for a interface.</p> <p>void MAPPING_GetIpv6InterfaceInfoForInterface (Node *node node, NodeId nodeId nodeID, int interfaceIndex, in6_addr* globalAddr, in6_addr* subnetAddr, unsigned int* subnetPrefixLen)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - The node. • <code>nodeID</code> - Node Id • <code>interfaceIndex</code> - The interface index. • <code>globalAddr</code> - The global IPv6 address. • <code>subnetAddr</code> - The subnet IPv6 address. • <code>subnetPrefixLen</code> - THe subnet prefex length. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
MAPPING_GetIpv6GlobalAddress	<p>Get IPV6 global address.</p> <p>BOOL MAPPING_GetIpv6GlobalAddress (Node *node node, NodeId nodeId nodeID, unsigned int tla, unsigned int nla, unsigned int sla, in6_addr * addr6)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - The node • <code>nodeID</code> - The node's id • <code>tla</code> - Top level aggregation • <code>nla</code> - Next level aggregation • <code>sla</code> - Site level aggregation • <code>addr6</code> - The global IPv6 address. <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - None
MAPPING_GetIpv6GlobalAddressForInterface	<p>BOOL MAPPING_GetIpv6GlobalAddressForInterface (Node * node, NodeId nodeId, int interfaceIndex, in6_addr * addr6, BOOL isDeprecated)</p>

<p>Get IPV6 global address for a node's nth interface.</p>	<p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - The node • <code>nodeId</code> - The node's id • <code>interfaceIndex</code> - The interface index. • <code>addr6</code> - The global IPv6 address. • <code>isDeprecated</code> - Return deprecated address (if valid) <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - None
<p>MAPPING_CreateIpv6GlobalUnicastAddr</p> <p>Create IPv6 Global Unicast Address from tla nla sla</p>	<p>void MAPPING_CreateIpv6GlobalUnicastAddr (unsigned int tla, unsigned int nla, unsigned int sla, int addressCounter, in6_addr* globalAddr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>tla</code> - Top level aggregation • <code>nla</code> - Next level aggregation • <code>sla</code> - Site level aggregation • <code>addressCounter</code> - The address counter. • <code>globalAddr</code> - The global IPv6 address. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
<p>MAPPING_CreateIpv6GlobalUnicastAddr</p> <p>Create IPv6 Global Unicast Address.</p>	<p>void MAPPING_CreateIpv6GlobalUnicastAddr (AddressMapType * map, in6_addr IPv6subnetAddress, unsigned int IPv6subnetPrefixLen, int addressCounter, in6_addr* globalAddr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>map</code> - The address map. • <code>IPv6subnetAddress</code> - The subnet address. • <code>IPv6subnetPrefixLen</code> - The prefix length. • <code>addressCounter</code> - The address counter. • <code>globalAddr</code> - The global IPv6 address. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None

MAPPING_CreateIpv6LinkLocalAddr	<p>void MAPPING_CreateIpv6LinkLocalAddr (in6_addr* globalAddr, in6_addr* linkLocalAddr, unsigned int subnetPrefixLen)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>globalAddr</code> - The global IPv6 address. • <code>linkLocalAddr</code> - The subnet IPv6 address. • <code>subnetPrefixLen</code> - The subnet prefix length. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
MAPPING_CreateIpv6SiteLocalAddr	<p>void MAPPING_CreateIpv6SiteLocalAddr (in6_addr* globalAddr, in6_addr* siteLocalAddr, unsigned short siteCounter, unsigned int subnetPrefixLen)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>globalAddr</code> - The global IPv6 address. • <code>siteLocalAddr</code> - The subnet IPv6 address. • <code>siteCounter</code> - The counter to use. • <code>subnetPrefixLen</code> - The subnet prefix length. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
MAPPING_CreateIpv6MulticastAddr	<p>void MAPPING_CreateIpv6MulticastAddr (in6_addr* globalAddr, in6_addr* multicastAddr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>globalAddr</code> - The global IPv6 address. • <code>multicastAddr</code> - The multicast IPv6 address. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
MAPPING_CreateIpv6SubnetAddr	<p>void MAPPING_CreateIpv6SubnetAddr (unsigned int tla, unsigned int nla, unsigned int sla, unsigned int* IPv6subnetPrefixLen, in6_addr* IPv6subnetAddress)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>tla</code> - Top level aggregation. • <code>nla</code> - Next level aggregation. • <code>sla</code> - Site level aggregation.

	<ul style="list-style-type: none"> • <code>IPv6subnetPrefixLen</code> - The IPv6 prefix length. • <code>IPv6subnetAddress</code> - The IPv6 subnet address. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
MAPPING_GetNodeIdFromGlobalAddr	<p>NodeId MAPPING_GetNodeIdFromGlobalAddr (<code>Node * node, in6_addr* globalAddr</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - The node. • <code>globalAddr</code> - The global IPv6 address. <p>Returns:</p> <ul style="list-style-type: none"> • <code>NodeId</code> - None
MAPPING_GetNodeIdFromLinkLayerAddr	<p>NodeId MAPPING_GetNodeIdFromLinkLayerAddr (<code>Node * node, NodeAddress linkLayerAddr</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - The node. • <code>linkLayerAddr</code> - The link layer address. <p>Returns:</p> <ul style="list-style-type: none"> • <code>NodeId</code> - None
MAPPING_CreateIpv6LinkLayerAddr	<p>NodeAddress MAPPING_CreateIpv6LinkLayerAddr (<code>unsigned int nodeId, int interfaceId</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>nodeId</code> - The node's id. • <code>interfaceId</code> - The interface id. <p>Returns:</p> <ul style="list-style-type: none"> • <code>NodeAddress</code> - None
MAPPING_IsIpv6AddressOfThisNode	<p>BOOL MAPPING_IsIpv6AddressOfThisNode (<code>Node* node, const NodeAddress nodeId, in6_addr* globalAddr</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - The node id. • <code>nodeId</code> - The node's address. • <code>globalAddr</code> - The global IPv6 address. <p>Returns:</p>

	<ul style="list-style-type: none"> • BOOL - None
MAPPING_IsNodeInThisIpRange	<p>checks whether the node is in given range of : Addresses.</p> <p>BOOL MAPPING_IsNodeInThisIpRange (Node* node, NodeId nodeId, NodeAddress startRange, NodeAddress endRange)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - The node. • nodeId - The node id. • startRange - The starting address. • endRange - The end address. <p>Returns:</p> <ul style="list-style-type: none"> • BOOL - None
MAPPING_IsIpAddressOfThisNode	<p>checks whether the ipv4 address is of this node.</p> <p>BOOL MAPPING_IsIpAddressOfThisNode (Node* node, const NodeAddress nodeId, NodeAddress addr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - The node. • nodeId - The node id. • addr - The address. <p>Returns:</p> <ul style="list-style-type: none"> • BOOL - None
MAPPING_GetInterfaceAddressForSubnet	<p>Get interface address for subnet using ipv6 addr.</p> <p>BOOL MAPPING_GetInterfaceAddressForSubnet (Node* node, NodeId nodeId, in6_addr* ipv6SubnetAddr, unsigned int prefixLenth, in6_addr* ipv6InterfaceAddr, int* interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - The node. • nodeId - The node id. • ipv6SubnetAddr - The subnet address. • prefixLenth - The subnet prefix length. • ipv6InterfaceAddr - The ipv6 interface address. • interfaceIndex - The interface index. <p>Returns:</p> <ul style="list-style-type: none"> • BOOL - None

MAPPING_GetInterfaceAddressForSubnet	<p>Get interface address for subnet using ipv6 addr.</p> <p>BOOL MAPPING_GetInterfaceAddressForSubnet (const AddressMapType* map, NodeId nodeId, in6_addr* ipv6SubnetAddr, unsigned int prefixLenth, in6_addr* ipv6InterfaceAddr, int* interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>map</code> - The address map. • <code>nodeId</code> - The node id. • <code>ipv6SubnetAddr</code> - The subnet address. • <code>prefixLenth</code> - The subnet prefix length. • <code>ipv6InterfaceAddr</code> - The ipv6 interface address. • <code>interfaceIndex</code> - The interface index. <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - None
MAPPING_GetInterfaceAddressForSubnet	<p>Get interface address for subnet using tla nla sla.</p> <p>BOOL MAPPING_GetInterfaceAddressForSubnet (Node* node node, NodeId nodeId nodeId, unsigned int tla, unsigned int nla, unsigned int sla, in6_addr* ipv6Addr, int* interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - The node. • <code>nodeId</code> - The node id. • <code>tla</code> - Top level aggregation. • <code>nla</code> - Next level aggregation. • <code>sla</code> - Site level aggregation. • <code>ipv6Addr</code> - The ipv6 interface address. • <code>interfaceIndex</code> - The interface index. <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - None
; MAPPING_GetInterfaceAddressForSubnet	<p>Get interface address for subnet using tla nla sla.</p> <p>BOOL ; MAPPING_GetInterfaceAddressForSubnet (const AddressMapType* map, NodeId nodeId, unsigned int tla, unsigned int nla, unsigned int sla, in6_addr* ipv6Addr, int* interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>map</code> - The address map. • <code>nodeId</code> - The node id. • <code>tla</code> - Top level aggregation.

	<ul style="list-style-type: none"> • <code>nla</code> - Next level aggregation. • <code>sla</code> - Site level aggregation. • <code>ipv6Addr</code> - The ipv6 interface address. • <code>interfaceIndex</code> - The interface index. <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - None
MAPPING_GetInterfaceFromLinkLayerAddress	<pre>int MAPPING_GetInterfaceFromLinkLayerAddress (Node* node, const NodeAddress linkLayerAddr)</pre> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - The node. • <code>linkLayerAddr</code> - The link layer address. <p>Returns:</p> <ul style="list-style-type: none"> • <code>int</code> - None
MAPPING_GetInterfaceIndexFromInterfaceAddress	<pre>int MAPPING_GetInterfaceIndexFromInterfaceAddress (Node* node, Address interfaceAddress)</pre> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - The node. • <code>interfaceAddress</code> - The interface address. <p>Returns:</p> <ul style="list-style-type: none"> • <code>int</code> - None
MAPPING_GetIpv6GlobalAddress	<pre>BOOL MAPPING_GetIpv6GlobalAddress (Node* node, NodeId nodeId, in6_addr subnetAddr, UInt32 prefixLen, in6_addr* addr6)</pre> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - The node. • <code>nodeId</code> - The node id • <code>subnetAddr</code> - The subnet address. • <code>prefixLen</code> - The subnet prefix length. • <code>addr6</code> - The IPv6 address. <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - None

MAPPING_GetDefaultInterfaceAddressInfoFromNodeId Get default interface address based on network type	<p>Address MAPPING_GetDefaultInterfaceAddressInfoFromNodeId (Node *node node, NodeAddress nodeId nodeId, NetworkType networktype networktype)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - The node. • <code>nodeId</code> - The node id. • <code>networktype</code> - The network type. <p>Returns:</p> <ul style="list-style-type: none"> • <code>Address</code> - None
Mapping_AutoCreateIPv6SubnetAddress Create IPv6 Testing Address Prefix (RFC 2471)from : ipv4 address.	<p>void Mapping_AutoCreateIPv6SubnetAddress (NodeAddress ipAddress, subnetString)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>ipAddress</code> - The IPv4 address. • <code>subnetString</code> - char* <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NONE
MAPPING_GetSubnetAddressFromInterfaceAddress Get subnet address from interface address.	<p>NodeAddress MAPPING_GetSubnetAddressFromInterfaceAddress (Node *node node, NodeAddress interfaceAddress)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - The node address. • <code>interfaceAddress</code> - The interface address. <p>Returns:</p> <ul style="list-style-type: none"> • <code>NodeAddress</code> - subnet address
MAPPING_GetSubnetAddressFromInterfaceAddress Get ipv6 network Prefix from interface address.	<p>BOOL MAPPING_GetSubnetAddressFromInterfaceAddress (Node * node, in6_addr* ipv6InterfaceAddr, in6_addr* ipv6SubnetAddr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - The node. • <code>ipv6InterfaceAddr</code> - The IPv6 interface address. • <code>ipv6SubnetAddr</code> - The subnet address pointer . <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - None

MAPPING_GetPrefixLengthForInterfaceAddress Get prefix length for interface address.	BOOL MAPPING_GetPrefixLengthForInterfaceAddress (Node* node, in6_addr* ipv6InterfaceAddr, unsigned int prefixLenth) Parameters: <ul style="list-style-type: none"> • node - The node. • ipv6InterfaceAddr - The IPV6 interface address. • prefixLenth - The interface prefix length. Returns: <ul style="list-style-type: none"> • BOOL - None
MAPPING_GetNetworkProtocolTypeForNode Get Network Protocol Type for the node.	NetworkProtocolType MAPPING_GetNetworkProtocolTypeForNode (NodeAddress nodeId, const NodeInput * nodeInput) Parameters: <ul style="list-style-type: none"> • nodeId - The node id. • nodeInput - The node input file Returns: <ul style="list-style-type: none"> • NetworkProtocolType - None



Contact [QualNet Support](#) for questions pertaining to the QualNet API Reference. This document is confidential and proprietary.
It may not be reproduced or distributed without the expressed written consent of [Scalable Network Technologies](#).

QualNet® is a Registered Trademark of [Scalable Network Technologies](#).

Copyright © 2001-2012 [Scalable Network Technologies, Inc.](#) All rights reserved.

QualNet 6.1 API Reference

MEMORY

This file describes the memory management data structures and functions.

Constant / Data Structure Summary

Type	Name
STRUCT	MemoryUsageData <p>Defines the parameters collected by the memory system. Restricted to kernel use.</p>

Function / Macro Summary

Return Type	Summary
void	MEM_CreateThreadData() <p>Creates partition-specific space for collecting memory usage statistics. This is used in threaded versions of QualNet, but not in distributed versions, currently.</p>
void	MEM_InitializeThreadData(MemoryUsageData* data) <p>Sets the partition-specific memory data for this partition.</p>
void	MEM_PrintThreadData() <p>Prints the partition-specific memory data.</p>
void	MEM_ReportPartitionUsage(int partitionId, UInt32 totalAllocatedMemory, UInt32 totalFreedMemory, UInt32 totalPeakUsage) <p>Prints out the total memory used by this partition.</p>
void	MEM_ReportTotalUsage(UInt32 totalAllocatedMemory, UInt32 totalFreedMemory, UInt32 totalPeakUsage) <p>Prints out the total memory usage statistics for the simulation. In a parallel run, the peak usage is the sum of the partition's peak usage and might not be precisely accurate.</p>

Constant / Data Structure Detail

Structure	MemoryUsageData Defines the parameters collected by the memory system. Restricted to kernel use.
-----------	--

Function / Macro Detail

Function / Macro	Format
MEM_CreateThreadData Creates partition-specific space for collecting memory usage statistics. This is used in threaded versions of QualNet, but not in distributed versions, currently.	void MEM_CreateThreadData () Parameters: Returns: <ul style="list-style-type: none">• void - None
MEM_InitializeThreadData Sets the partition-specific memory data for this partition.	void MEM_InitializeThreadData (MemoryUsageData* data) Parameters: <ul style="list-style-type: none">• data - the data Returns: <ul style="list-style-type: none">• void - None
MEM_PrintThreadData Prints the partition-specific memory data.	void MEM_PrintThreadData () Parameters: Returns: <ul style="list-style-type: none">• void - None
MEM_ReportPartitionUsage Prints out the total memory used by this partition.	void MEM_ReportPartitionUsage (int partitionId, UInt32 totalAllocatedMemory, UInt32 totalFreedMemory, UInt32 totalPeakUsage) Parameters: <ul style="list-style-type: none">• partitionId - the partition number• totalAllocatedMemory - sum of all MEM_malloc calls• totalFreedMemory - sum of all MEM_free calls

	<ul style="list-style-type: none"> • <code>totalPeakUsage</code> - peak usage of allocated memory <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
MEM_ReportTotalUsage	<p>Prints out the total memory usage statistics for the simulation. In a parallel run, the peak usage is the sum of the partition's peak usage and might not be precisely accurate.</p> <pre>void MEM_ReportTotalUsage (UInt32 totalAllocatedMemory, UInt32 totalFreedMemory, UInt32 totalPeakUsage)</pre> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>totalAllocatedMemory</code> - sum of all <code>MEM_malloc</code> calls • <code>totalFreedMemory</code> - sum of all <code>MEM_free</code> calls • <code>totalPeakUsage</code> - peak usage of allocated memory <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None



Contact [QualNet Support](#) for questions pertaining to the QualNet API Reference. This document is confidential and proprietary.
It may not be reproduced or distributed without the expressed written consent of [Scalable Network Technologies](#).

QualNet® is a Registered Trademark of [Scalable Network Technologies](#).

Copyright © 2001-2012 [Scalable Network Technologies, Inc.](#). All rights reserved.

QualNet 6.1 API Reference

MESSAGE

This file describes the message structure used to implement events and functions for message operations.

Constant / Data Structure Summary

Type	Name
CONSTANT	MSG_MAX_HDR_SIZE Maximum Header Size
CONSTANT	SMALL_INFO_SPACE_SIZE Size of small Info field. Should be larger than all commonly used info field data structures, especially PropTxInfo and PropRxInfo.
CONSTANT	MSG_PAYLOAD_LIST_MAX Maximum message payload list
CONSTANT	MAX_CACHED_PAYLOAD_SIZE Maximum cached payload size
CONSTANT	MSG_INFO_LIST_MAX Maximum message info list
CONSTANT	MAX_INFO_FIELDS Maximum number of info fields
CONSTANT	MAX_HEADERS Maximum number of headers
ENUMERATION	MessageInfoType Type of information in the info field. One message can only have up to one info field with a specific info type.
STRUCT	MessageInfoHeader

	This is a structure which contains information about a info field.
STRUCT	<p>Message</p> <p>This is the main data strucure that represents a discrete event in qualnet. This is used to represent timer as well as to simulate actual sending of packets across the network.</p>

Function / Macro Summary

Return Type	Summary
void	<p>MESSAGE_PrintMessage(Node* node, Message* msg)</p>
void	<p>MESSAGE_Send(Node* node, Message* msg, clocktype delay, bool isMT)</p> <p>Function call used to send a message within QualNet. When a message is sent using this mechanism, only the pointer to the message is actually sent through the system. So the user has to be careful not to do anything with the content of the pointer once MESSAGE_Send has been called.</p>
void	<p>MESSAGE_SendMT(Node* node, Message* msg, clocktype delay)</p> <p>Function call used to send a message from independent threads running within QualNet, for example those associated with external interfaces.</p>
void	<p>MESSAGE_RemoteSend(Node* node, NodeId destNodeId, Message* msg, clocktype delay)</p> <p>Function used to send a message to a node that might be on a remote partition. The system will make a shallow copy of the message, meaning it can't contain any pointers in the info field or the packet itself. This function is very unsafe. If you use it, your program will probably crash. Only I can use it.</p>
void	<p>MESSAGE_RouteReceivedRemoteEvent(Node* node, Message* msg)</p> <p>Counterpart to MESSAGE_RemoteSend, this function allows models that send remote messages to provide special handling for them on the receiving partition. This function is called in real time as the messages are received, so must be used carefully.</p>
void	<p>MESSAGE_CancelSelfMsg(Node* node, Message* msgToCancelPtr)</p> <p>Function call used to cancel a event message in the QualNet scheduler. The Message must be a self message (timer) i.e. a message a node sent to itself. The msgToCancelPtr must a pointer to the original message that needs to be canceled.</p>
Message*	<p>MESSAGE_Alloc(Node* node, int layerType, int protocol, int eventType)</p>

	Allocate a new Message structure. This is called when a new message has to be sent through the system. The last three parameters indicate the layerType, protocol and the eventType that will be set for this message. MESSAGE_Alloc (PartitionData* partition, int layerType, int protocol, int eventType)
Message*	Allocate a new Message structure. This is called when a new message has to be sent through the system. The last three parameters indicate the layerType, protocol and the eventType that will be set for this message. MESSAGE_AllocMT (PartitionData* partition, int layerType, int protocol, int eventType)
Message*	Mutli-thread safe version of MESSAGE_Alloc for use by worker threads. MESSAGE_InfoFieldAlloc (Node* node, int infoSize)
char*	Allocate space for one "info" field MESSAGE_InfoFieldAlloc (PartitionData* partition, int infoSize)
char*	Allocate space for one "info" field MESSAGE_InfoFieldAllocMT (PartitionData* partition, int infoSize)
void	Multi-thread safe version of MESSAGE_InfoFieldAlloc MESSAGE_InfoFieldFree (Node* node, MessageInfoHeader* hdrPtr)
char*	Free space for one "info" field MESSAGE_AddInfo (Node* node, Message* msg, int infoSize, unsigned short infoType)
char*	Allocate one "info" field with given info type for the message. This function is used for the delivery of data for messages which are NOT packets as well as the delivery of extra information for messages which are packets. If a "info" field with the same info type has previously been allocated for the message, it will be replaced by a new "info" field with the specified size. Once this function has been called, MESSAGE_ReturnInfo function can be used to get a pointer to the allocated space for the info field in the message structure. MESSAGE_AddInfo (PartitionData* partition, Message* msg, int infoSize, unsigned short infoType)
void	Allocate one "info" field with given info type for the message. This function is used for the delivery of data for messages which are NOT packets as well as the delivery of extra information for messages which are packets. If a "info" field with the same info type has previously been allocated for the message, it will be replaced by a new "info" field with the specified size. Once this function has been called, MESSAGE_ReturnInfo function can be used to get a pointer to the allocated space for the info field in the message structure. MESSAGE_RemoveInfo (Node* node, Message* msg, unsigned short infoType)
char *	Remove one "info" field with given info type from the info array of the message. MESSAGE_InfoAlloc (Node* node, Message* msg, int infoSize)

	Allocate the default "info" field for the message. This function is similar to MESSAGE_AddInfo. The difference is that it assumes the type of the info field to be allocated is INFO_TYPE_DEFAULT.
char *	MESSAGE_InfoAlloc (PartitionData* partition, Message* msg, int infoSize)
	Allocate the default "info" field for the message. This function is similar to MESSAGE_AddInfo. The difference is that it assumes the type of the info field to be allocated is INFO_TYPE_DEFAULT.
int	MESSAGE_ReturnInfoSize (Message* msg, unsigned short infoType, int fragmentNumber)
	Returns the size of a "info" field with given info type in the info array of the message.
int	MESSAGE_ReturnInfoSize (Message* msg, unsigned short infoType)
	Returns the size of a "info" field with given info type in the info array of the message.
char*	MESSAGE_ReturnInfo (Message* msg, unsigned short infoType)
	Returns a pointer to the "info" field with given info type in the info array of the message.
void	MESSAGE_CopyInfo (Node* node, Message* dsgMsg, Message* srcMsg)
	Copy the "info" fields of the source message to the destination message.
void	MESSAGE_CopyInfo (Node* node, Message* dsgMsg, MessageInfoHeader* srcInfo)
	Copy the "info" fields of the source info header to the destination message.
void	MESSAGE_FragmentPacket (Node* node, Message* msg, int fragUnit, Message*** fragList, int* numFrags, TraceProtocolType protocolType)
	Fragment one packet into multiple fragments Note: The original packet will be freed in this function. The array for storing pointers to fragments will be dynamically allocated. The caller of this function will need to free the memory.
Message*	MESSAGE_ReassemblePacket (Node* node, Message** fragList, int numFrags, TraceProtocolType protocolType)
	Reassemble multiple fragments into one packet Note: All the fragments will be freed in this function.
Message*	MESSAGE_PackMessage (Node* node, Message* msgList, TraceProtocolType origProtocol, int* actualPktSize)
	Pack a list of messages to be one message structure Whole contents of the list messages will be put as payload of the new message. So the packet size of the new message cannot be directly used now. The original lis of msgs will be freed.
Message*	MESSAGE_UnpackMessage (Node* node, Message* msg, bool copyInfo, bool freeOld)

	<p>Unpack a super message to the original list of messages The list of messages were stored as payload of this super message.</p>
void	<p><code>MESSAGE_PacketAlloc</code>(Node* node, Message* msg, int packetSize, TraceProtocolType originalProtocol)</p> <p>Allocate the "payload" field for the packet to be delivered. Add additional free space in front of the packet for headers that might be added to the packet. This function can be called from the application layer or anywhere else (e.g TCP, IP) that a packet may originate from. The "packetSize" variable will be set to the "packetSize" parameter specified in the function call. Once this function has been called the "packet" variable in the message structure can be used to access this space.</p>
void	<p><code>MESSAGE_PacketAlloc</code>(PartitionData* partition, Message* msg, int packetSize, TraceProtocolType originalProtocol, bool isMT)</p> <p>Allocate the "payload" field for the packet to be delivered. Add additional free space in front of the packet for headers that might be added to the packet. This function can be called from the application layer or anywhere else (e.g TCP, IP) that a packet may originate from. The "packetSize" variable will be set to the "packetSize" parameter specified in the function call. Once this function has been called the "packet" variable in the message structure can be used to access this space.</p>
void	<p><code>MESSAGE_AddHeader</code>(Node* node, Message* msg, int hdrSize, TraceProtocolType traceProtocol)</p> <p>This function is called to reserve additional space for a header of size "hdrSize" for the packet enclosed in the message. The "packetSize" variable in the message structure will be increased by "hdrSize". Since the header has to be prepended to the current packet, after this function is called the "packet" variable in the message structure will point the space occupied by this new header.</p>
void	<p><code>MESSAGE_RemoveHeader</code>(Node* node, Message* msg, int hdrSize, TraceProtocolType traceProtocol)</p> <p>This function is called to remove a header from the packet. The "packetSize" variable in the message will be decreased by "hdrSize".</p>
char*	<p><code>MESSAGE_ReturnHeader</code>(Message* msg, int header)</p> <p>This is kind of a hack so that MAC protocols (dot11) that need to peek at a packet that still has the PHY header can return the contents after the first (N) headers without first removing those headers.</p>
void	<p><code>MESSAGE_ExpandPacket</code>(Node* node, Message* msg, int size)</p> <p>Expand packet by a specified size</p>
void	<p><code>MESSAGE_ShrinkPacket</code>(Node* node, Message* msg, int size)</p> <p>This function is called to shrink packet by a specified size.</p>
void	<p><code>MESSAGE_Free</code>(PartitionData* partition, Message* msg)</p> <p>When the message is no longer needed it can be freed. Firstly the "payload" and "info" fields of the message are freed. Then the message itself is freed. It is important to remember to free the message. Otherwise there will nasty memory leaks in the program.</p>
void	<p><code>MESSAGE_Free</code>(Node* node, Message* msg)</p>

	When the message is no longer needed it can be freed. Firstly the "payload" and "info" fields of the message are freed. Then the message itself is freed. It is important to remember to free the message. Otherwise there will nasty memory leaks in the program.
void	<code>MESSAGE_FreeList</code> (Node* node, Message* msg)
	Free a list of message until the next pointer of the message is NULL.
Message*	<code>MESSAGE_Duplicate</code> (Node* node, Message* msg)
	Create a new message which is an exact duplicate of the message supplied as the parameter to the function and return the new message.
Message*	<code>MESSAGE_Duplicate</code> (PartitionData* partition, Message* msg, bool isMT)
	Create a new message which is an exact duplicate of the message supplied as the parameter to the function and return the new message.
Message*	<code>MESSAGE_DuplicateMT</code> (PartitionData* partition, Message* msg)
	Create a new message which is an exact duplicate of the message supplied as the parameter to the function and return the new message.
char*	<code>MESSAGE_PayloadAlloc</code> (Node* node, int payloadSize)
	Allocate a character payload out of the free list, if possible otherwise via malloc.
char*	<code>MESSAGE_PayloadAlloc</code> (PartitionData* partition, int payloadSize, bool isMT)
	Allocate a character payload out of the free list, if possible otherwise via malloc.
void	<code>MESSAGE_PayloadFree</code> (PartitionData* partition, Char* payload, int payloadSize)
	Return a character payload to the free list, if possible otherwise free it.
void	<code>MESSAGE_PayloadFree</code> (Node* node, Char* payload, int payloadSize)
	Return a character payload to the free list, if possible otherwise free it.
void	<code>MESSAGE_FreeList</code> (Node* node, Message* msg)
	Free a list of messages until the next pointer of the message is NULL.
int	<code>MESSAGE_ReturnNumFrgs</code> (Message* msg)
	Returns the number of fragments used to create a TCP packet.
int	<code>MESSAGE_ReturnFragSeqNum</code> (Message* msg, int fragmentNumber)

	Returns the sequence number of a particular fragments in the TCP packet. MESSAGE_ReturnFragSize (Message* msg, int fragmentNumber)
int	Returns the size of a particular fragment in the TCP packet. MESSAGE_ReturnFragNumInfos (Message* msg, int fragmentNumber)
void	Returns the number of info fields associated with a particular fragment in the TCP packet. MESSAGE_AppendInfo (PartitionData* partitionData, Message* msg, int infosize, short infoType)
void	Appends the "info" fields of the source message to the destination message. MESSAGE_AppendInfo (Node* node, Message* msg, int infosize, short infoType)
void	Appends the "info" fields of the source message to the destination message. MESSAGE_AppendInfo (Node* node, Message* dsgMsg, MessageInfoHeader* srcInfo)
void	Appends the "info" fields of the source message to the destination message. MESSAGE_AppendInfo (Node* node, Message* dsgMsg, Message* srcMsg)
size_t	Appends the "info" fields of the source message to the destination message. MESSAGE_SizeOf ()
BOOL	Returns the size of a message. Used in place of sizeof() in the kernel code to allow for users to add more fields to the message. MESSAGE_FragmentPacket (Node* node, Message*& msg, Message*& fragmentedMsg, Message*& remainingMsg, int fragUnit, TraceProtocolType protocolType, bool freeOriginalMsg)
Message*	Fragment one packet into TWO fragments Note:(i) This API treats the original packet as raw packet and does not take account of fragmentation related information like fragment id. The caller of this API will have to itself put in logic for distinguishing the fragmented packets (ii) Overloaded MESSAGE_FragmentPacket MESSAGE_ReassemblePacket (Node* node, Message* fragMsg1, Message* fragMsg2, TraceProtocolType protocolType)
void	Reassemble TWO fragments into one packet Note: (i) None of the fragments will be freed in this API. The caller of this API will itself have to free the fragments (ii) Overloaded MESSAGE_ReassemblePacket MESSAGE_SendAsEarlyAsPossible (Node* node, Message* msg)

This function is used primarily by external interfaces to inject events into the Simulator as soon as possible without causing problems for parallel execution.

Constant / Data Structure Detail

Constant	<code>MSG_MAX_HDR_SIZE</code> 512 Maximum Header Size
Constant	<code>SMALL_INFO_SPACE_SIZE</code> 112 Size of small Info field. Should be larger than all commonly used info field data structures, especially PropTxInfo and PropRxInfo.
Constant	<code>MSG_PAYLOAD_LIST_MAX</code> 1000 Maximum message payload list
Constant	<code>MAX_CACHED_PAYLOAD_SIZE</code> 1024 Maximum cached payload size
Constant	<code>MSG_INFO_LIST_MAX</code> 1000 Maximum message info list
Constant	<code>MAX_INFO_FIELDS</code> 12 Maximum number of info fields
Constant	<code>MAX_HEADERS</code> 10 Maximum number of headers
Enumeration	<code>MessageInfoType</code> Type of information in the info field. One message can only have up to one info field with a specific info type.
Structure	<code>MessageInfoHeader</code>

	This is a structure which contains information about a info field.
Structure	<p>Message</p> <p>This is the main data structure that represents a discrete event in qualnet. This is used to represent timer as well as to simulate actual sending of packets across the network.</p>

Function / Macro Detail

Function / Macro	Format
MESSAGE_PrintMessage	<p>void MESSAGE_PrintMessage (Node* node, Message* msg)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node which is sending message • <code>msg</code> - message to be printed <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
MESSAGE_Send	<p>void MESSAGE_Send (Node* node, Message* msg, clocktype delay, bool isMT)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node which is sending message • <code>msg</code> - message to be delivered • <code>delay</code> - delay suffered by this message. • <code>isMT</code> - is the function being called from a thread? <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
MESSAGE_SendMT	<p>void MESSAGE_SendMT (Node* node, Message* msg, clocktype delay)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node which is sending message • <code>msg</code> - message to be delivered • <code>delay</code> - delay suffered by this message. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code>

	<p>- NULL</p>
MESSAGE_RemoteSend	<p>void MESSAGE_RemoteSend (Node* node, NodeId destNodeId, Message* msg, clocktype delay)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node which is sending message • destNodeId - nodeId of receiving node • msg - message to be delivered • delay - delay suffered by this message. <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
MESSAGE_RouteReceivedRemoteEvent	<p>void MESSAGE_RouteReceivedRemoteEvent (Node* node, Message* msg)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node which is sending message • msg - message to be delivered <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
MESSAGE_CancelSelfMsg	<p>void MESSAGE_CancelSelfMsg (Node* node, Message* msgToCancelPtr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node which is sending message • msgToCancelPtr - message to be cancelled <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
MESSAGE_Alloc	<p>Message* MESSAGE_Alloc (Node* node, int layerType, int protocol, int eventType)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node which is allocating message • layerType - Layer type to be set for this message • protocol - Protocol to be set for this message • eventType - event type to be set for this message <p>Returns:</p>

	<ul style="list-style-type: none"> • <code>Message*</code> - Pointer to allocated message structure
MESSAGE_Alloc	<p>Allocate a new Message structure. This is called when a new message has to be sent through the system. The last three parameters indicate the layerType, protocol and the eventType that will be set for this message.</p> <p>Message* MESSAGE_Alloc (PartitionData* partition, int layerType, int protocol, int eventType)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>partition</code> - partition that is allocating message • <code>layerType</code> - Layer type to be set for this message • <code>protocol</code> - Protocol to be set for this message • <code>eventType</code> - event type to be set for this message <p>Returns:</p> <ul style="list-style-type: none"> • <code>Message*</code> - Pointer to allocated message structure
MESSAGE_AllocMT	<p>Mutli-thread safe version of MESSAGE_Alloc for use by worker threads.</p> <p>Message* MESSAGE_AllocMT (PartitionData* partition, int layerType, int protocol, int eventType)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>partition</code> - partition that is allocating message • <code>layerType</code> - Layer type to be set for this message • <code>protocol</code> - Protocol to be set for this message • <code>eventType</code> - event type to be set for this message <p>Returns:</p> <ul style="list-style-type: none"> • <code>Message*</code> - Pointer to allocated message structure
MESSAGE_InfoFieldAlloc	<p>Allocate space for one "info" field</p> <p>char* MESSAGE_InfoFieldAlloc (Node* node, int infoSize)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node which is allocating the space. • <code>infoSize</code> - size of the space to be allocated <p>Returns:</p> <ul style="list-style-type: none"> • <code>char*</code> - pointer to the allocated space.
MESSAGE_InfoFieldAlloc	<p>Allocate space for one "info" field</p> <p>char* MESSAGE_InfoFieldAlloc (PartitionData* partition, int infoSize)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>partition</code> - partition which is allocating the space. • <code>infoSize</code> - size of the space to be allocated

	<p>Returns:</p> <ul style="list-style-type: none"> • <code>char*</code> - pointer to the allocated space.
MESSAGE_InfoFieldAllocMT	<p>char* MESSAGE_InfoFieldAllocMT (PartitionData* partition, int infoSize)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>partition</code> - partition which is allocating the space. • <code>infoSize</code> - size of the space to be allocated <p>Returns:</p> <ul style="list-style-type: none"> • <code>char*</code> - pointer to the allocated space.
MESSAGE_InfoFieldFree	<p>void MESSAGE_InfoFieldFree (Node* node, MessageInfoHeader* hdrPtr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node which is allocating the space. • <code>hdrPtr</code> - pointer to the "info" field <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
MESSAGE_AddInfo	<p>char* MESSAGE_AddInfo (Node* node, Message* msg, int infoSize, unsigned short infoType)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node which is allocating the info field. • <code>msg</code> - message for which "info" field • <code>infoSize</code> - size of the "info" field to be allocated • <code>infoType</code> - type of the "info" field to be allocated. <p>Returns:</p> <ul style="list-style-type: none"> • <code>char*</code> - Pointer to the added info field
MESSAGE_AddInfo	<p>char* MESSAGE_AddInfo (PartitionData* partition, Message* msg, int infoSize, unsigned short infoType)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>partition</code> - partition which is allocating the info field. • <code>msg</code> - message for which "info" field • <code>infoSize</code> - size of the "info" field to be allocated

<p>If a "info" field with the same info type has previously been allocated for the message, it will be replaced by a new "info" field with the specified size. Once this function has been called, MESSAGE_ReturnInfo function can be used to get a pointer to the allocated space for the info field in the message structure.</p>	<ul style="list-style-type: none"> • <code>infoType</code> - type of the "info" field to be allocated. <p>Returns:</p> <ul style="list-style-type: none"> • <code>char*</code> - Pointer to the added info field
<p>MESSAGE_RemoveInfo</p> <p>Remove one "info" field with given info type from the info array of the message.</p>	<p><code>void MESSAGE_RemoveInfo (Node* node, Message* msg, unsigned short infoType)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node which is removing info field. • <code>msg</code> - message for which "info" field • <code>infoType</code> - type of the "info" field to be removed. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
<p>MESSAGE_InfoAlloc</p> <p>Allocate the default "info" field for the message. This function is similar to MESSAGE_AddInfo. The difference is that it assumes the type of the info field to be allocated is INFO_TYPE_DEFAULT.</p>	<p><code>char * MESSAGE_InfoAlloc (Node* node, Message* msg, int infoSize)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node which is allocating the info field. • <code>msg</code> - message for which "info" field • <code>infoSize</code> - size of the "info" field to be allocated <p>Returns:</p> <ul style="list-style-type: none"> • <code>char *</code> - None
<p>MESSAGE_InfoAlloc</p> <p>Allocate the default "info" field for the message. This function is similar to MESSAGE_AddInfo. The difference is that it assumes the type of the info field to be allocated is INFO_TYPE_DEFAULT.</p>	<p><code>char * MESSAGE_InfoAlloc (PartitionData* partition, Message* msg, int infoSize)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>partition</code> - partition which is allocating the info field. • <code>msg</code> - message for which "info" field • <code>infoSize</code> - size of the "info" field to be allocated <p>Returns:</p> <ul style="list-style-type: none"> • <code>char *</code> - None
<p>MESSAGE_ReturnInfoSize</p>	<p><code>int MESSAGE_ReturnInfoSize (Message* msg, unsigned short infoType, int fragmentNumber)</code></p> <p>Parameters:</p>

	<p>Returns the size of a "info" field with given info type in the info array of the message.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>msg</code> - message for which "info" field • <code>infoType</code> - type of the "info" field. • <code>fragmentNumber</code> - Location of the fragment in the TCP packet <p>Returns:</p> <ul style="list-style-type: none"> • <code>int</code> - size of the info field.
MESSAGE_ReturnInfoSize	<p>int MESSAGE_ReturnInfoSize (Message* msg, unsigned short infoType)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>msg</code> - message for which "info" field • <code>infoType</code> - type of the "info" field. <p>Returns:</p> <ul style="list-style-type: none"> • <code>int</code> - size of the info field.
MESSAGE_ReturnInfo	<p>char* MESSAGE_ReturnInfo (Message* msg, unsigned short infoType)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>msg</code> - message for which "info" field • <code>infoType</code> - type of the "info" field to be returned. <p>Returns:</p> <ul style="list-style-type: none"> • <code>char*</code> - Pointer to the "info" field with given type. NULL if not found.
MESSAGE_CopyInfo	<p>void MESSAGE_CopyInfo (Node* node, Message* dsgMsg, Message* srcMsg)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Node which is copying the info fields • <code>dsgMsg</code> - Destination message • <code>srcMsg</code> - Source message <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
MESSAGE_CopyInfo	<p>void MESSAGE_CopyInfo (Node* node, Message* dsgMsg, MessageInfoHeader* srcInfo)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Node which is copying the info fields • <code>dsgMsg</code> - Destination message

	<ul style="list-style-type: none"> • <code>srcInfo</code> - Info Header structure <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
MESSAGE_FragmentPacket	<p>Fragment one packet into multiple fragments Note: The original packet will be freed in this function. The array for storing pointers to fragments will be dynamically allocated. The caller of this function will need to free the memory.</p> <p><code>void MESSAGE_FragmentPacket (Node* node, Message* msg, int fragUnit, Message*** fragList, int* numFrags, TraceProtocolType protocolType)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node which is fragmenting the packet • <code>msg</code> - The packet to be fragmented • <code>fragUnit</code> - The unit size for fragmenting the packet • <code>fragList</code> - A list of fragments created. • <code>numFrags</code> - Number of fragments in the fragment list. • <code>protocolType</code> - Protocol type for packet tracing. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
MESSAGE_ReassemblePacket	<p>Reassemble multiple fragments into one packet Note: All the fragments will be freed in this function.</p> <p><code>Message* MESSAGE_ReassemblePacket (Node* node, Message** fragList, int numFrags, TraceProtocolType protocolType)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node which is assembling the packet • <code>fragList</code> - A list of fragments. • <code>numFrags</code> - Number of fragments in the fragment list. • <code>protocolType</code> - Protocol type for packet tracing. <p>Returns:</p> <ul style="list-style-type: none"> • <code>Message*</code> - The reassembled packet.
MESSAGE_PackMessage	<p>Pack a list of messages to be one message structure Whole contents of the list messages will be put as payload of the new message. So the packet size of the new message cannot be directly used now. The original lis of msgs will be freed.</p> <p><code>Message* MESSAGE_PackMessage (Node* node, Message* msgList, TraceProtocolType origProtocol, int* actualPktSize)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node. • <code>msgList</code> - Pointer to a list of messages • <code>origProtocol</code> - Protocol allocating this packet • <code>actualPktSize</code> - For return sum of packet size of msgs in list

	<p>Returns:</p> <ul style="list-style-type: none"> • <code>Message*</code> - The super msg contains a list of msgs as payload
MESSAGE_UnpackMessage	<p><code>Message* MESSAGE_UnpackMessage (Node* node, Message* msg, bool copyInfo, bool freeOld)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to node. • <code>msg</code> - Pointer to the supper msg contains list of msgs • <code>copyInfo</code> - Whether copy info from old msg to first msg • <code>freeOld</code> - Whether the original message should be freed <p>Returns:</p> <ul style="list-style-type: none"> • <code>Message*</code> - A list of messages unpacked from original msg
MESSAGE_PacketAlloc	<p><code>void MESSAGE_PacketAlloc (Node* node, Message* msg, int packetSize, TraceProtocolType originalProtocol)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node which is allocating the packet • <code>msg</code> - message for which packet has to be allocated • <code>packetSize</code> - size of the packet to be allocated • <code>originalProtocol</code> - Protocol allocating this packet <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
MESSAGE_PacketAlloc	<p><code>void MESSAGE_PacketAlloc (PartitionData* partition, Message* msg, int packetSize, TraceProtocolType originalProtocol, bool isMT)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>partition</code> - artition which is allocating the packet • <code>msg</code> - message for which packet has to be allocated • <code>packetSize</code> - size of the packet to be allocated • <code>originalProtocol</code> - Protocol allocating this packet • <code>isMT</code> - Is this packet being created from a worker thread <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL

MESSAGE_AddHeader	<p>This function is called to reserve additional space for a header of size "hdrSize" for the packet enclosed in the message. The "packetSize" variable in the message structure will be increased by "hdrSize". Since the header has to be prepended to the current packet, after this function is called the "packet" variable in the message structure will point the space occupied by this new header.</p>	<p>void MESSAGE_AddHeader (Node* node, Message* msg, int hdrSize, TraceProtocolType traceProtocol)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node which is adding header • msg - message for which header has to be added • hdrSize - size of the header to be added • traceProtocol - protocol name, from trace.h <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
MESSAGE_RemoveHeader	<p>This function is called to remove a header from the packet. The "packetSize" variable in the message will be decreased by "hdrSize".</p>	<p>void MESSAGE_RemoveHeader (Node* node, Message* msg, int hdrSize, TraceProtocolType traceProtocol)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node which is removing the packet header • msg - message for which header is being removed • hdrSize - size of the header being removed • traceProtocol - protocol removing this header. <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
MESSAGE_ReturnHeader	<p>This is kind of a hack so that MAC protocols (dot11) that need to peek at a packet that still has the PHY header can return the contents after the first (N) headers without first removing those headers.</p>	<p>char* MESSAGE_ReturnHeader (Message* msg, int header)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • msg - message containing a packet with headers • header - number of the header to return. <p>Returns:</p> <ul style="list-style-type: none"> • char* - the packet starting at the header'th header
MESSAGE_ExpandPacket	<p>Expand packet by a specified size</p>	<p>void MESSAGE_ExpandPacket (Node* node, Message* msg, int size)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node which is expanding the packet • msg - message which is to be expanded • size - size to expand

	<p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
MESSAGE_ShrinkPacket	<p>This function is called to shrink packet by a specified size.</p> <p>void MESSAGE_ShrinkPacket (Node* node, Message* msg, int size)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node which is shrinking packet • msg - message whose packet is be shrunked • size - size to shrink <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
MESSAGE_Free	<p>When the message is no longer needed it can be freed. Firstly the "payload" and "info" fields of the message are freed. Then the message itself is freed. It is important to remember to free the message. Otherwise there will nasty memory leaks in the program.</p> <p>void MESSAGE_Free (PartitionData* partition, Message* msg)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • partition - partition which is freeing the message • msg - message which has to be freed <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
MESSAGE_Free	<p>When the message is no longer needed it can be freed. Firstly the "payload" and "info" fields of the message are freed. Then the message itself is freed. It is important to remember to free the message. Otherwise there will nasty memory leaks in the program.</p> <p>void MESSAGE_Free (Node* node, Message* msg)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node which is freeing the message • msg - message which has to be freed <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
MESSAGE_FreeList	<p>Free a list of message until the next pointer of the message is NULL.</p> <p>void MESSAGE_FreeList (Node* node, Message* msg)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node which is freeing the message • msg - message which has to be freed <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
MESSAGE_Duplicate	Message* MESSAGE_Duplicate (Node* node, Message* msg)

	<p>Create a new message which is an exact duplicate of the message supplied as the parameter to the function and return the new message.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node is calling message copy • msg - message for which duplicate has to be made <p>Returns:</p> <ul style="list-style-type: none"> • Message* - Pointer to the new message
MESSAGE_Duplicate	<p>Message* MESSAGE_Duplicate (PartitionData* partition, Message* msg, bool isMT)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • partition - partition is calling message copy • msg - message for which duplicate has to be made • isMT - Is this function being called from the context <p>Returns:</p> <ul style="list-style-type: none"> • Message* - Pointer to the new message
MESSAGE_DuplicateMT	<p>Message* MESSAGE_DuplicateMT (PartitionData* partition, Message* msg)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • partition - partition is calling message copy • msg - message for which duplicate has to be made <p>Returns:</p> <ul style="list-style-type: none"> • Message* - Pointer to the new message
MESSAGE_PayloadAlloc	<p>char* MESSAGE_PayloadAlloc (Node* node, int payloadSize)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node which is allocating payload • payloadSize - size of the field to be allocated <p>Returns:</p> <ul style="list-style-type: none"> • char* - pointer to the allocated memory
MESSAGE_PayloadAlloc	<p>char* MESSAGE_PayloadAlloc (PartitionData* partition, int payloadSize, bool isMT)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • partition - partition which is allocating payload

	<ul style="list-style-type: none"> • <code>payloadSize</code> - size of the field to be allocated • <code>isMT</code> - Is this packet being created from a worker thread <p>Returns:</p> <ul style="list-style-type: none"> • <code>char*</code> - pointer to the allocated memory
MESSAGE_PayloadFree	<p>Return a character payload to the free list, if possible otherwise free it.</p> <p><code>void MESSAGE_PayloadFree (PartitionData* partition, Char* payload, int payloadSize)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>partition</code> - partition which is freeing payload • <code>payload</code> - Pointer to the payload field • <code>payloadSize</code> - size of the payload field <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
MESSAGE_PayloadFree	<p>Return a character payload to the free list, if possible otherwise free it.</p> <p><code>void MESSAGE_PayloadFree (Node* node, Char* payload, int payloadSize)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node which is freeing payload • <code>payload</code> - Pointer to the payload field • <code>payloadSize</code> - size of the payload field <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
MESSAGE_FreeList	<p>Free a list of messages until the next pointer of the message is NULL.</p> <p><code>void MESSAGE_FreeList (Node* node, Message* msg)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node which is freeing the message • <code>msg</code> - message which has to be freed <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
MESSAGE_ReturnNumFrags	<p>Returns the number of fragments used to create a TCP packet.</p> <p><code>int MESSAGE_ReturnNumFrags (Message* msg)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>msg</code> - message for which "info" field <p>Returns:</p>

	<ul style="list-style-type: none"> • <code>int</code> - Number of Fragments. 0 if none.
MESSAGE_ReturnFragSeqNum	<p><code>int MESSAGE_ReturnFragSeqNum (Message* msg, int fragmentNumber)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>msg</code> - message for which "info" field • <code>fragmentNumber</code> - fragment location in the TCP message. <p>Returns:</p> <ul style="list-style-type: none"> • <code>int</code> - Sequence number of the fragment. -1 if none.
MESSAGE_ReturnFragSize	<p><code>int MESSAGE_ReturnFragSize (Message* msg, int fragmentNumber)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>msg</code> - message for which "info" field • <code>fragmentNumber</code> - fragment location in the TCP message. <p>Returns:</p> <ul style="list-style-type: none"> • <code>int</code> - Sequence number of the fragment. 0 if none.
MESSAGE_ReturnFragNumInfos	<p><code>int MESSAGE_ReturnFragNumInfos (Message* msg, int fragmentNumber)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>msg</code> - message for which "info" field • <code>fragmentNumber</code> - fragment location in the TCP message. <p>Returns:</p> <ul style="list-style-type: none"> • <code>int</code> - Sequence number of the fragment. 0 if none.
MESSAGE_AppendInfo	<p><code>void MESSAGE_AppendInfo (PartitionData* partitionData, Message* msg, int infosize, short infoType)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>partitionData</code> - Partition which is copying the info fields • <code>msg</code> - Destination message • <code>infosize</code> - size of the info field • <code>infoType</code> - type of info field. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL

MESSAGE_AppendInfo	<p>void MESSAGE_AppendInfo (Node* node, Message* msg, int infosize, short infoType)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Node which is copying the info fields • msg - Destination message • infosize - size of the info field • infoType - type of info field. <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
MESSAGE_AppendInfo	<p>void MESSAGE_AppendInfo (Node* node, Message* dsgMsg, MessageInfoHeader* srcInfo)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Node which is copying the info fields • dsgMsg - Destination message • srcInfo - Source message info vector <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
MESSAGE_AppendInfo	<p>void MESSAGE_AppendInfo (Node* node, Message* dsgMsg, Message* srcMsg)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Node which is copying the info fields • dsgMsg - Destination message • srcMsg - Source message <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
MESSAGE_SizeOf	<p>size_t MESSAGE_SizeOf ()</p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> • size_t - sizeof(msg)
MESSAGE_FragmentPacket	BOOL MESSAGE_FragmentPacket (Node* node, Message*& msg, Message*& fragmentedMsg, Message*& remainingMsg, int fragUnit, TraceProtocolType protocolType, bool freeOriginalMsg)

<p>Fragment one packet into TWO fragments Note:(i) This API treats the original packet as raw packet and does not take account of fragmentation related information like fragment id. The caller of this API will have to itself put in logic for distinguishing the fragmented packets (ii) Overloaded MESSAGE_FragmentPacket</p>	<p>Parameters:</p> <ul style="list-style-type: none"> • node - node which is fragmenting the packet • msg - The packet to be fragmented • fragmentedMsg - First fragment • remainingMsg - Remaining packet • fragUnit - The unit size for fragmenting the packet • protocolType - Protocol type for packet tracing. • freeOriginalMsg - If TRUE, then original msg is set to NULL <p>Returns:</p> <ul style="list-style-type: none"> • BOOL - TRUE if any fragment is created, FALSE otherwise
<p>MESSAGE_ReassemblePacket</p> <p>Reassemble TWO fragments into one packet Note: (i) None of the fragments will be freed in this API. The caller of this API will itself have to free the fragments (ii) Overloaded MESSAGE_ReassemblePacket</p>	<p>Message* MESSAGE_ReassemblePacket (Node* node, Message* fragMsg1, Message* fragMsg2, TraceProtocolType protocolType)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node which is assembling the packet • fragMsg1 - First fragment • fragMsg2 - Second fragment • protocolType - Protocol type for packet tracing. <p>Returns:</p> <ul style="list-style-type: none"> • Message* - The reassembled packet.
<p>MESSAGE_SendAsEarlyAsPossible</p> <p>This function is used primarily by external interfaces to inject events into the Simulator as soon as possible without causing problems for parallel execution.</p>	<p>void MESSAGE_SendAsEarlyAsPossible (Node* node, Message* msg)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node which is sending message • msg - message to be delivered <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL



Contact [QualNet Support](#) for questions pertaining to the QualNet API Reference. This document is confidential and proprietary.
It may not be reproduced or distributed without the expressed written consent of [Scalable Network Technologies](#).

QualNet® is a Registered Trademark of [Scalable Network Technologies](#).

Copyright © 2001-2012 [Scalable Network Technologies, Inc.](#) All rights reserved.

QualNet 6.1 API Reference

MOBILITY

This file describes data structures and functions used by mobility models.

Constant / Data Structure Summary

Type	Name
CONSTANT	DEFAULT_DISTANCE_GRANULARITY Defines the default distance granularity
CONSTANT	NUM_NODE_PLACEMENT_TYPES Defines the number of node placement schemes
CONSTANT	NUM_MOBILITY_TYPES Defines the number of mobility models
CONSTANT	NUM_PAST_MOBILITY_EVENTS Number of past mobility models stored
ENUMERATION	NodePlacementType Specifies different node placement schemes
ENUMERATION	MobilityType Specifies different mobility models
STRUCT	MobilityHeap A Heap that determines the earliest time
STRUCT	MobilityElement Defines all the element of mobility model.
STRUCT	MobilityRemainder

	A structure that defines the next states of the elements of mobility model.
STRUCT	<p><u>MobilityData</u></p> <p>This structure keeps the data related to mobility model. It also holds the variables which are static and variable during the simulation. Buffer caches future position updates as well.</p>

Function / Macro Summary

Return Type	Summary
void	<p><u>MOBILITY_InsertEvent</u>(MobilityHeap* heapPtr, Node* node)</p> <p>Inserts an event.</p>
void	<p><u>MOBILITY_DeleteEvent</u>(MobilityHeap* heapPtr, Node* node)</p> <p>Deletes an event.</p>
void	<p><u>MOBILITY_HeapFixDownEvent</u>(MobilityHeap* heapPtr, int i)</p> <p>Inserts an event and sort out the heap downwards</p>
void	<p><u>MOBILITY_AllocateNodePositions</u>(int numNodes, NodeAddress* nodeIdArray, NodePositions** nodePositions, int** nodePlacementTypeCounts, NodeInput* nodeInput, int seedVal)</p> <p>Allocates memory for nodePositions and mobilityData Note: This function is called before NODE_CreateNode(). It cannot access Node structure</p>
void	<p><u>MOBILITY_PreInitialize</u>(NodeAddress nodeId, MobilityData* mobilityData, NodeInput* nodeInput, int seedVal)</p> <p>Initializes most variables in mobilityData. (Node positions are set in MOBILITY_SetNodePositions().) Note: This function is called before NODE_CreateNode(). It cannot access Node structure</p>
void	<p><u>MOBILITY_PostInitialize</u>(Node* node, NodeInput* nodeInput)</p> <p>Initializes variables in mobilityData not initialized by MOBILITY_PreInitialize().</p>
void	<p><u>MOBILITY_UpdatePathProfiles</u>(MobilityHeap* pathProfileHeap, clocktype nextEventTime, clocktype* upperBoundTime)</p> <p>Updates the path profiles.</p>
void	<p><u>MOBILITY_Finalize</u>(Node* node)</p>

	Called at the end of simulation to collect the results of the simulation of the mobility data.
void	<code>MOBILITY_ProcessEvent</code> (Node* node)
	Models the behaviour of the mobility models on receiving a message.
void	<code>MOBILITY_AddANewDestination</code> (MobilityData* mobilityData, clocktype arrivalTime, Coordinates dest, Orientation orientation, double zValue)
	Adds a new destination.
BOOL	<code>MOBILITY_NextPosition</code> (Node* node, MobilityElement* element)
	Update next node position for static mobility models
clocktype	<code>MOBILITY_NextMoveTime</code> (Node* node)
	Determines the time of next movement.
MobilityElement*	<code>MOBILITY_ReturnMobilityElement</code> (Node* node, int sequenceNum)
	Used to get the mobility element.
void	<code>MOBILITY_InsertANewEvent</code> (Node* node, clocktype nextMoveTime, Coordinates position, Orientation orientation, double speed)
	Inserts a new event.
bool	<code>MOBILITY_NodeIsIndoors</code> (Node* node)
	Returns whether the node is indoors.
void	<code>MOBILITY_SetIndoors</code> (Node* node, bool indoors)
	Sets the node's indoor variable.
void	<code>MOBILITY_ReturnCoordinates</code> (Node* node, Coordinates position)
	Returns the coordinate.
void	<code>MOBILITY_ReturnOrientation</code> (Node* node, Orientation* orientation)
	Returns the node orientation.
void	<code>MOBILITY_ReturnInstantaneousSpeed</code> (Node* node, double* speed)

	Returns instantaneous speed of a node. MOBILITY_ReturnSequenceNum (Node* node, int* sequenceNum)
void	Returns a sequence number for the current position. MOBILITY_SetNodePositions (int numNodes, NodePositions* nodePositions, int* nodePlacementTypeCounts, TerrainData* terrainData, NodeInput* nodeInput, RandomSeed seed, clocktype maxSimTime)
void	Set positions of nodes MOBILITY_PostInitializePartition (PartitionData* partitionData)
void	Initialization of mobility models that must be done after partition is created; MOBILITY_SetNodePositions would be too early MOBILITY_NodePlacementFinalize (PartitionData* partitionData)
void	Finalize mobility models MOBILITY_ChangeGroundNode (Node* node, BOOL before, BOOL after)
void	Change GroundNode value.. MOBILITY_ChangePositionGranularity (Node* node)
	Change Mobility-Position-Granularity value..

Constant / Data Structure Detail

Constant	DEFAULT_DISTANCE_GRANULARITY 1 Defines the default distance granularity
Constant	NUM_NODE_PLACEMENT_TYPES 7 Defines the number of node placement schemes
Constant	NUM_MOBILITY_TYPES 5

	Defines the number of mobility models
Constant	NUM_PAST_MOBILITY_EVENTS 2 Number of past mobility models stored
Enumeration	NodePlacementType Specifies different node placement schemes
Enumeration	MobilityType Specifies different mobility models
Structure	MobilityHeap A Heap that determines the earliest time
Structure	MobilityElement Defines all the element of mobility model.
Structure	MobilityRemainder A structure that defines the next states of the elements of mobility model.
Structure	MobilityData This structure keeps the data related to mobility model. It also holds the variables which are static and variable during the simulation. Buffer caches future position updates as well.

Function / Macro Detail

Function / Macro	Format
MOBILITY_InsertEvent Inserts an event.	Format <pre>void MOBILITY_InsertEvent (MobilityHeap* heapPtr, Node* node)</pre> <p>Parameters:</p> <ul style="list-style-type: none"> heapPtr - A pointer of type MobilityHeap. node - A pointer to node. <p>Returns:</p>

	<ul style="list-style-type: none"> • void - None
MOBILITY_DeleteEvent	<p>Deletes an event.</p> <p>void MOBILITY_DeleteEvent (MobilityHeap* heapPtr, Node* node)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • heapPtr - A pointer of type MobilityHeap. • node - A pointer to node. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
MOBILITY_HeapFixDownEvent	<p>Inserts an event and sort out the heap downwards</p> <p>void MOBILITY_HeapFixDownEvent (MobilityHeap* heapPtr, int i)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • heapPtr - A pointer of type MobilityHeap. • i - index <p>Returns:</p> <ul style="list-style-type: none"> • void - None
MOBILITY_AllocateNodePositions	<p>Allocates memory for nodePositions and mobilityData Note: This function is called before NODE_CreateNode(). It cannot access Node structure</p> <p>void MOBILITY_AllocateNodePositions (int numNodes, NodeAddress* nodeIdArray, NodePositions** nodePositions, int** nodePlacementTypeCounts, NodeInput* nodeInput, int seedVal)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • numNodes - number of nodes • nodeIdArray - array of nodeId • nodePositions - pointer to the array • nodePlacementTypeCounts - array of placement type counts • nodeInput - configuration input • seedVal - seed for random number seeds <p>Returns:</p> <ul style="list-style-type: none"> • void - None
MOBILITY_PreInitialize	<p>Initializes most variables in mobilityData. (Node positions are set in MOBILITY_SetNodePositions().) Note: This</p> <p>void MOBILITY_PreInitialize (NodeAddress nodeId, MobilityData* mobilityData, NodeInput* nodeInput, int seedVal)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • nodeId - nodeId • mobilityData - mobilityData to be initialized

<p>function is called before NODE_CreateNode(). It cannot access Node structure</p>	<ul style="list-style-type: none"> • nodeInput - configuration input • seedVal - seed for random number seeds <p>Returns:</p> <ul style="list-style-type: none"> • void - None
<p>MOBILITY_PostInitialize</p> <p>Initializes variables in mobilityData not initialized by MOBILITY_PreInitialize().</p>	<p>void MOBILITY_PostInitialize (Node* node, NodeInput* nodeInput)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node being initialized • nodeInput - structure containing contents of input file <p>Returns:</p> <ul style="list-style-type: none"> • void - None
<p>MOBILITY_UpdatePathProfiles</p> <p>Updates the path profiles.</p>	<p>void MOBILITY_UpdatePathProfiles (MobilityHeap* pathProfileHeap, clocktype nextEventTime, clocktype* upperBoundTime)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • pathProfileHeap - MobilityHeap structure. • nextEventTime - Next event time. • upperBoundTime - Upper bound time. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
<p>MOBILITY_Finalize</p> <p>Called at the end of simulation to collect the results of the simulation of the mobility data.</p>	<p>void MOBILITY_Finalize (Node* node)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Node for which results are to be collected. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
<p>MOBILITY_ProcessEvent</p> <p>Models the behaviour of the mobility models on receiving a message.</p>	<p>void MOBILITY_ProcessEvent (Node* node)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Node which received the message <p>Returns:</p> <ul style="list-style-type: none"> • void - None

MOBILITY_AddANewDestination	<p>void MOBILITY_AddANewDestination (MobilityData* mobilityData, clocktype arrivalTime, Coordinates dest, Orientation orientation, double zValue)</p> <p>Adds a new destination.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • mobilityData - MobilityData of the node • arrivalTime - Arrival time • dest - Destination • orientation - Orientation • zValue - original zValue <p>Returns:</p> <ul style="list-style-type: none"> • void - None
MOBILITY_NextPosition	<p>BOOL MOBILITY_NextPosition (Node* node, MobilityElement* element)</p> <p>Update next node position for static mobility models</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Node to be updated • element - next mobility update <p>Returns:</p> <ul style="list-style-type: none"> • BOOL - None
MOBILITY_NextMoveTime	<p>clocktype MOBILITY_NextMoveTime (Node* node)</p> <p>Determines the time of next movement.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. <p>Returns:</p> <ul style="list-style-type: none"> • clocktype - Next time of movement.
MOBILITY_ReturnMobilityElement	<p>MobilityElement* MOBILITY_ReturnMobilityElement (Node* node, int sequenceNum)</p> <p>Used to get the mobility element.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • sequenceNum - Sequence number. <p>Returns:</p> <ul style="list-style-type: none"> • MobilityElement* - None
MOBILITY_InsertANewEvent	void MOBILITY_InsertANewEvent (Node* node, clocktype nextMoveTime, Coordinates position, Orientation orientation,

	<p>Inserts a new event.</p> <p>double speed)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • nextMoveTime - Time of next movement. • position - Position of the node. • orientation - Node orientation. • speed - Speed of the node. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
MOBILITY_NodeIsIndoors Returns whether the node is indoors.	<p>bool MOBILITY_NodeIsIndoors (Node* node)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. <p>Returns:</p> <ul style="list-style-type: none"> • bool - returns true if indoors.
MOBILITY_SetIndoors Sets the node's indoor variable.	<p>void MOBILITY_SetIndoors (Node* node, bool indoors)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • indoors - true if the node is indoors. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
MOBILITY_ReturnCoordinates Returns the coordinate.	<p>void MOBILITY_ReturnCoordinates (Node* node, Coordinates position)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • position - Position of the node. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
MOBILITY_ReturnOrientation	<p>void MOBILITY_ReturnOrientation (Node* node, Orientation* orientation)</p> <p>Parameters:</p>

	<ul style="list-style-type: none"> • node - Pointer to node. • orientation - Pointer to Orientation. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
MOBILITY_ReturnInstantaneousSpeed Returns instantaneous speed of a node.	<p>void MOBILITY_ReturnInstantaneousSpeed (Node* node, double* speed)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • speed - Speed of the node, double pointer. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
MOBILITY_ReturnSequenceNum Returns a sequence number for the current position.	<p>void MOBILITY_ReturnSequenceNum (Node* node, int* sequenceNum)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • sequenceNum - Sequence number. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
MOBILITY_SetNodePositions Set positions of nodes	<p>void MOBILITY_SetNodePositions (int numNodes, NodePositions* nodePositions, int* nodePlacementTypeCounts, TerrainData* terrainData, NodeInput* nodeInput, RandomSeed seed, clocktype maxSimTime)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • numNodes - Defines the number of nodes to be distributed. • nodePositions - Pointer to NodePositionInfo. States • nodePlacementTypeCounts - Array of placement type counts • terrainData - Terrain data. • nodeInput - Pointer to NodeInput, defines the • seed - Stores the seed value. • maxSimTime - Maximum simulation time. <p>Returns:</p> <ul style="list-style-type: none"> • void - None

MOBILITY_PostInitializePartition Initialization of mobility models that must be done after partition is created; MOBILITY_SetNodePositions would be too early	<p>void MOBILITY_PostInitializePartition (PartitionData* partitionData)</p> <p>Parameters:</p> <ul style="list-style-type: none"> partitionData - Pointer to the partition data <p>Returns:</p> <ul style="list-style-type: none"> void - None
MOBILITY_NodePlacementFinalize Finalize mobility models	<p>void MOBILITY_NodePlacementFinalize (PartitionData* partitionData)</p> <p>Parameters:</p> <ul style="list-style-type: none"> partitionData - Pointer to the partition data <p>Returns:</p> <ul style="list-style-type: none"> void - None
MOBILITY_ChangeGroundNode Change GroundNode value..	<p>void MOBILITY_ChangeGroundNode (Node* node, BOOL before, BOOL after)</p> <p>Parameters:</p> <ul style="list-style-type: none"> node - Pointer to node being initialized. before - Orginal value for Ground-Node variable after - new value for Ground-Node variable. <p>Returns:</p> <ul style="list-style-type: none"> void - None
MOBILITY_ChangePositionGranularity Change Mobility-Position-Granularity value..	<p>void MOBILITY_ChangePositionGranularity (Node* node)</p> <p>Parameters:</p> <ul style="list-style-type: none"> node - Pointer to node being initialized. <p>Returns:</p> <ul style="list-style-type: none"> void - None



QualNet® is a Registered Trademark of [Scalable Network Technologies, Inc.](#)

Copyright © 2001-2012 [Scalable Network Technologies, Inc.](#). All rights reserved.

QualNet 6.1 API Reference

MUTEX

This file describes objects for use in creating critical regions (synchronized access) for global variables or data structures that have to be shared between threads.

Function / Macro Summary

Return Type	Summary
None	<p>QNThreadLock(QNThreadMutex mutex)</p> <p>This constructor is used to begin a critical region.</p>
None	<p>QNPartitionLock(QNPartitionMutex mutex)</p> <p>This constructor is used to begin a critical region.</p>

Function / Macro Detail

Function / Macro	Format
QNThreadLock	<p>None QNThreadLock (QNThreadMutex mutex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • mutex - Pointer to the Thread mutex to lock for this region. <p>Returns:</p> <ul style="list-style-type: none"> • None - None
QNPartitionLock	<p>None QNPartitionLock (QNPartitionMutex mutex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • mutex - Pointer to the Partition mutex to lock <p>Returns:</p> <ul style="list-style-type: none"> • None - None



Contact [QualNet Support](#) for questions pertaining to the QualNet API Reference. This document is confidential and proprietary.
It may not be reproduced or distributed without the expressed written consent of [Scalable Network Technologies](#).

[QualNet®](#) is a Registered Trademark of [Scalable Network Technologies](#).

Copyright © 2001-2012 [Scalable Network Technologies, Inc.](#) All rights reserved.

QualNet 6.1 API Reference

NETWORK LAYER

This file describes the data structures and functions used by the Network Layer.

Constant / Data Structure Summary

Type	Name
CONSTANT	DEFAULT_IP_QUEUE_COUNT
	Default number of output queue per interface
CONSTANT	DEFAULT_CPU_QUEUE_SIZE
	Default size of CPU queue (in byte)
CONSTANT	DEFAULT_NETWORK_INPUT_QUEUE_SIZE
	Default size in bytes of an input queue, if it's not specified in the input file with the IP-QUEUE-PRIORITY-INPUT-QUEUE-SIZE parameter.
CONSTANT	DEFAULT_NETWORK_OUTPUT_QUEUE_SIZE
	Default size in bytes of an output queue, if it's not specified in the input file with the IP-QUEUE-PRIORITY-QUEUE-SIZE parameter.
CONSTANT	DEFAULT_ETHERNET_MTU
	Default Ethernet MTU(Maximum transmission unit) in bytes. QualNet does not model Ethernet yet, but this value is used (in the init functions in network/fifoqueue.c and network/redqueue.c) to compute the initial number of Message * instances that are used to store packets in queues.Regardless, the buffer capacity of a queue is not the number of Message * instances, but a certain number of bytes, as expected.
CONSTANT	IP_MAXPACKET
	Maximum IP packet size
CONSTANT	NETWORK_IP_UNLIMITED_BACKPLANE_THROUGHPUT
	Maximum throughput of backplane of network.
ENUMERATION	NetworkIpBackplaneStatusType

	Status of backplane (either busy or idle) NetworkRoutingAdminDistanceType
ENUMERATION	Administrative distance of different routing protocol NetworkRoutingProtocolType
ENUMERATION	Enlisted different network/routing protocol ManagementReportType
ENUMERATION	Type of management report message ManagementResponseType
ENUMERATION	Type of management response message ManagementResponseType
STRUCT	Main data structure of network layer NetworkData
STRUCT	data structure of management report ManagementReport
STRUCT	data structure of management response ManagementResponse

Function / Macro Summary

Return Type	Summary
void	NETWORK_ManagementReport (Node* node, int interfaceIndex, ManagementReport* report, ManagementReportResponse* resp) Deliver a MAC management request to the NETWORK layer
void	NetworkGetInterfaceInfo() (Node* node, int interfaceIndex, Address* address) Returns interface information for a interface. Information means its address and type

NETWORK LAYER

void	NetworkIpGetInterfaceAddressString (Node* node, int interfaceIndex, const char* ipAddrString)
	ipAddrString is filled in by interface's ipv6 address in character format.
NetworkType	NetworkIpGetInterfaceType (Node* node, int interfaceIndex)
	Returns type of network (ipv4 or ipv6) the interface.
void	NETWORK_ReceivePacketFromMacLayer (Node* node, Message* message, NodeAddress lastHopAddress, int interfaceIndex)
	Network-layer receives packets from MAC layer, now check Overloaded Function to support Mac Address type of IP and call proper function
void	NETWORK_Reset (Node* node, int interfaceIndex)
	Reset Network protocols and/or layer.
void	NETWORK_AddResetFunctionList (Node* node, int interfaceIndex)
	Add which protocols to be reset to a fuction list pointer.

Constant / Data Structure Detail

Constant	DEFAULT_IP_QUEUE_COUNT 3 Default number of output queue per interface
Constant	DEFAULT_CPU_QUEUE_SIZE 640000 Default size of CPU queue (in byte)
Constant	DEFAULT_NETWORK_INPUT_QUEUE_SIZE 150000 Default size in bytes of an input queue, if it's not specified in the input file with the IP-QUEUE-PRIORITY-INPUT-QUEUE-SIZE parameter.
Constant	DEFAULT_NETWORK_OUTPUT_QUEUE_SIZE 150000 Default size in bytes of an output queue, if it's not specified in the input file with the IP-QUEUE-PRIORITY-QUEUE-SIZE parameter.

Constant	<code>DEFAULT_ETHERNET_MTU 1500</code>
	Default Ethernet MTU(Maximum transmission unit) in bytes. QualNet does not model Ethernet yet, but this value is used (in the init functions in network/fifoqueue.c and network/redqueue.c) to compute the initial number of <code>Message *</code> instances that are used to store packets in queues.Regardless, the buffer capacity of a queue is not the number of <code>Message *</code> instances, but a certain number of bytes, as expected.
Constant	<code>IP_MAXPACKET 65535</code>
	Maximum IP packet size
Constant	<code>NETWORK_IP_UNLIMITED_BACKPLANE_THROUGHPUT 0</code>
	Maximum throughput of backplane of network.
Enumeration	<code>NetworkIpBackplaneStatusType</code>
	Status of backplane (either busy or idle)
Enumeration	<code>NetworkRoutingAdminDistanceType</code>
	Administrative distance of different routing protocol
Enumeration	<code>NetworkRoutingProtocolType</code>
	Enlisted different network/routing protocol
Enumeration	<code>ManagementReportType</code>
	Type of management report message
Enumeration	<code>ManagementResponseType</code>
	Type of management response message
Structure	<code>NetworkData</code>
	Main data structure of network layer
Structure	<code>ManagementReport</code>

	data structure of management report
Structure	ManagementResponse
	data structure of management response

Function / Macro Detail

Function / Macro	Format
NETWORK_ManagementReport Deliver a MAC management request to the NETWORK layer	<p>void NETWORK_ManagementReport (Node* node, int interfaceIndex, ManagementReport* report, ManagementReportResponse* resp)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to a network node • interfaceIndex - index of interface • report - Pointer to a management report • resp - Pointer to a management response <p>Returns:</p> <ul style="list-style-type: none"> • void - None
NetworkGetInterfaceInfo() Returns interface information for a interface. Information means its address and type	<p>void NetworkGetInterfaceInfo() (Node* node, int interfaceIndex, Address* address)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • interfaceIndex - interface index for which info required. • address - interface info returned <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
NetworkIpGetInterfaceAddressString ipAddrString is filled in by interface's ipv6 address in character format.	<p>void NetworkIpGetInterfaceAddressString (Node* node, int interfaceIndex, const char* ipAddrString)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • interfaceIndex - Interface index. • ipAddrString - Pointer to string ipv6 address. <p>Returns:</p>

	<ul style="list-style-type: none"> • void - None
NetworkIpGetInterfaceType	<p>NetworkType NetworkIpGetInterfaceType (Node* node, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • interfaceIndex - Interface index. <p>Returns:</p> <ul style="list-style-type: none"> • NetworkType - None
NETWORK_ReceivePacketFromMacLayer	<p>void NETWORK_ReceivePacketFromMacLayer (Node* node, Message* message, NodeAddress lastHopAddress, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node • message - Message received • lastHopAddress - last hop address • interfaceIndex - incoimg interface <p>Returns:</p> <ul style="list-style-type: none"> • void - None
NETWORK_Reset	<p>void NETWORK_Reset (Node* node, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • interfaceIndex - Interface index. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
NETWORK_AddResetFunctionList	<p>void NETWORK_AddResetFunctionList (Node* node, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Pointer to node. • interfaceIndex - Interface index. <p>Returns:</p> <ul style="list-style-type: none"> • void - None



Contact [QualNet Support](#) for questions pertaining to the QualNet API Reference. This document is confidential and proprietary.
It may not be reproduced or distributed without the expressed written consent of [Scalable Network Technologies](#).

QualNet® is a Registered Trademark of [Scalable Network Technologies](#).

Copyright © 2001-2012 [Scalable Network Technologies, Inc.](#) All rights reserved.

QualNet 6.1 API Reference

NODE

This file defines the Node data structure and some generic operations on nodes.

Constant / Data Structure Summary

Type	Name
ENUMERATION	NodeGlobalIndex
	<p>This enumeration contains indexes into the nodeGlobal array used for module data. For example, nodeGlobal[NodeGlobal_JNE] points to a JneData structure that stores data related to JNE interfaces and applications.</p>
STRUCT	Node
	<p>This struct includes all the information for a particular node. State information for each layer can be accessed from this structure.</p>
STRUCT	NodePositions
	<p>Contains information about the initial positions of nodes.</p>

Function / Macro Summary

Return Type	Summary
void	<p>NODE_CreateNode(PartitionData* partitionData, NodeId nodeId, int index)</p> <p>Function used to allocate and initialize a node.</p>
void	<p>NODE_ProcessEvent(Node* node, Message* msg)</p> <p>Function used to call the appropriate layer to execute instructions for the message</p>
void	<p>NODE_PrintLocation(Node* node, int coordinateSystemType)</p> <p>Prints the node's three dimensional coordinates.</p>
TerrainData*	<p>NODE_GetTerrainPtr(Node* node)</p>

	Get terrainData pointer.
--	--------------------------

Constant / Data Structure Detail

Enumeration	NodeGlobalIndex This enumeration contains indexes into the nodeGlobal array used for module data. For example, nodeGlobal[NodeGlobal_JNE] points to a JneData structure that stores data related to JNE interfaces and applications.
Structure	Node This struct includes all the information for a particular node. State information for each layer can be accessed from this structure.
Structure	NodePositions Contains information about the initial positions of nodes.

Function / Macro Detail

Function / Macro	Format
NODE_CreateNode Function used to allocate and initialize a node.	void NODE_CreateNode (PartitionData* partitionData, NodeId nodeId, int index) Parameters: <ul style="list-style-type: none"> • partitionData - the partition that owns the node • nodeId - the node's ID • index - the node's index within the partition Returns: <ul style="list-style-type: none"> • void - None
NODE_ProcessEvent Function used to call the appropriate layer to execute instructions for the message	void NODE_ProcessEvent (Node* node, Message* msg) Parameters: <ul style="list-style-type: none"> • node - node for which message is to be delivered • msg - message for which instructions are to be executed

	<p>Returns:</p> <ul style="list-style-type: none"> • void - None
NODE_PrintLocation Prints the node's three dimensional coordinates.	<p>void NODE_PrintLocation (Node* node, int coordinateSystemType)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - the node • coordinateSystemType - Cartesian or LatLonAlt <p>Returns:</p> <ul style="list-style-type: none"> • void - None
NODE_GetTerrainPtr Get terrainData pointer.	<p>TerrainData* NODE_GetTerrainPtr (Node* node)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - the node <p>Returns:</p> <ul style="list-style-type: none"> • TerrainData* - TerrainData pointer



Contact [QualNet Support](#) for questions pertaining to the QualNet API Reference. This document is confidential and proprietary.
It may not be reproduced or distributed without the expressed written consent of [Scalable Network Technologies](#).

QualNet® is a Registered Trademark of [Scalable Network Technologies](#).

Copyright © 2001-2012 [Scalable Network Technologies, Inc.](#) All rights reserved.

QualNet 6.1 API Reference

PARALLEL

This file describes data structures and functions used for parallel programming.

Constant / Data Structure Summary

Type	Name
CONSTANT	MAX_THREADS
	The maximum number of processes that can be used in parallel QualNet. Customers do not receive parallel.cpp, so cannot effectively change this value.
ENUMERATION	SynchronizationAlgorithm
	Possible algorithms to use in the parallel runtime. Synchronous is used by default.
STRUCT	LookaheadLocator
	This struct is allows us to be able to remove from the LookaheadCalculator's heap. This way lookahead handles can request they be removed. Internally, as the heap re-heapifies these locators are updated.
STRUCT	EotHeapElement
	Basic data structure for simplifying lookahead calculation.
STRUCT	LookaheadCalculator
	Stores a heap of EOT elements to calculate lookahead.

Function / Macro Summary

Return Type	Summary
LookaheadHandle	PARALLEL_AllocateLookaheadHandle (Node* node)
void	Obtains a new lookahead handle that allows a protocol to indicate minimum delay values for output. This minimum delay is called EOT - earliest output time. PARALLEL_AddLookaheadHandleToLookaheadCalculator (Node* node, LookaheadHandle lookaheadHandle,

	<pre>clocktype eotOfNode)</pre> <p>Adds a new LookaheadHandle to the lookahead calculator.</p>
void	<p><code>PARALLEL_SetLookaheadHandleEOT</code>(Node* node, LookaheadHandle lookaheadHandle, clocktype eot)</p> <p>Protocols that use EOT will make use of this function more than any other to update the earliest output time as the simulation progresses. Use of EOT is an all-or-nothing option. If your protocol uses EOT, it <u>must</u> use EOT pervasively.</p>
void	<p><code>PARALLEL_RemoveLookaheadHandleFromLookaheadCalculator</code>(Node* node, LookaheadHandle lookaheadHandle, clocktype* eotOfNode)</p> <p>Removes a LookaheadHandle from the lookahead calculator.</p>
void	<p><code>PARALLEL_SetMinimumLookaheadForInterface</code>(Node* node, clocktype minLookahead)</p> <p>Sets a minimum delay for messages going out on this interface. This is typically set by the protocol running on that interface.</p>
void	<p><code>PARALLEL_InitLookaheadCalculator</code>(LookaheadCalculator lookaheadCalculator)</p> <p>Initializes lookahead calculation. For kernel use only.</p>
int	<p><code>PARALLEL_AssignNodesToPartitions</code>(int numNodes, int numberOfPartitions, NodeInput* nodeInput, NodePosition* nodePos, AddressMapType* map)</p> <p>Using their positions or other information, assigns each node to a partition. For kernel use only.</p>
int	<p><code>PARALLEL_GetPartitionForNode</code>(NodeId nodeId)</p> <p>Allows parallel code to determine to what partition a node is assigned. If a Node* is available, it's much quicker to just look it up directly</p>
void	<p><code>PARALLEL_InitializeParallelRuntime</code>(int numberOfThreads)</p> <p>Sets global variables and stuff. For kernel use only.</p>
void	<p><code>PARALLEL_CreatePartitionThreads</code>(int numberOfThreads, NodeInput* nodeInput, PartitionData* partitionArray)</p> <p>Creates the threads for parallel execution and starts them running. For kernel use only.</p>
void	<p><code>PARALLEL_GetRemoteMessages</code>(PartitionData* partitionData)</p> <p>Collects all the messages received from other partitions. For kernel use only.</p>
void	<p><code>PARALLEL_GetRemoteMessagesAndBarrier</code>(PartitionData* partitionData)</p>

	<p>Collects all the messages received from other partitions. This function also acts as a barrier. For kernel use only.</p> <p><code>PARALLEL_SendRemoteMessages</code>(Message* msgList, PartitionData* partitionData, int partitionId)</p>
	<p>Sends one or more messages to a remote partition. For kernel use only.</p> <p><code>PARALLEL_DeliverRemoteMessages</code>(PartitionData* partitionData)</p>
	<p>Delivers cached messages to all remote partitions. For kernel use only.</p>
void	<p><code>PARALLEL_SendRemoteMessagesOob</code>(Message* msgList, PartitionData* partitionData, int partitionId, bool isResponse)</p>
	<p>Sends one or more messages to a remote partition. These messages are oob messages and will be processed immediately. For kernel use only.</p>
void	<p><code>PARALLEL_SendMessageToAllPartitions</code>(Message* msg, PartitionData* partitionData, bool freeMsg)</p>
	<p>Sends a message to all remote partitions, but not the current one. By default, duplicates will be sent to all remote partitions and the original freed, but if freeMsg is false, the original message will not be freed.</p>
	<p><code>PARALLEL_SendRemoteLinkMessage</code>(Node* node, Message* msg, LinkData* link, clocktype txDelay)</p>
	<p>Sends one LINK message to a remote partition.</p>
void	<p><code>PARALLEL_UpdateSafeTime</code>(PartitionData* partitionData)</p>
	<p>A generic function for calculating the window of safe events For kernel use only.</p>
clocktype	<p><code>PARALLEL_ReturnEarliestGlobalEventTime</code>(PartitionData* partitionData)</p>
	<p>Returns the earliest global event time. Required for interfacing to time-sensitive external programs. For kernel use only.</p>
void	<p><code>PARALLEL_Exit</code>(PartitionData* partitionData)</p>
	<p>Exits from the parallel system, killing threads, etc. For kernel use only.</p>
void	<p><code>PARALLEL_SetProtocolIsNotEOTCapable</code>(Node* node)</p>
	<p>Currently, EOT can only be used if supported by all protocols running in the scenario. If any protocol is not capable, only the minimum lookahead is used.</p>
void	<p><code>PARALLEL_EnableDynamicMobility</code>()</p>
	<p>Forces the runtime to consider mobility events when calculating EOT/ECOT. Mobility events are ignored by default. This function</p>

	should be called during the initialization of models where changes in position or direction of one node may affect the behavior of other nodes.
void	PARALLEL_SetGreedy (bool greedy) Tells the kernel to use spin locks on barriers if true, or to use blocking barriers otherwise. In greedy mode, the Simulator needs a dedicated CPU per partition.
bool	PARALLEL_IsGreedy () Checks whether SetGreedy has been called.
void	PARALLEL_Preflight (PartitionData* partitionData) Initializes parallel operation.
void	PARALLEL_ScheduleMessagesOnPartition (PartitionData* partitionData, Message* msgList, Message** oobMessage, bool* gotOobMessage, bool isMT) Takes a list of messages or an OOB message and schedules them for execution on the current partition. Typically these messages have arrived from a remote partition.
void	PARALLEL_EndSimulation (PartitionData* partitionData) Shuts down the parallel engine, including whatever synchronization is required.
void	PARALLEL_BuildStatFile (int numPartitions, char* statFileName, char* experimentPrefix) Builds the final stat file when running in parallel node. Should only be called once from partition 0.
void	PARALLEL_NumberOfSynchronizations () Return the number of synchronizations performed per partition
void	PARALLEL_StartRealTimeThread (PartitionData* partitionData) Tells the kernel to use an independent thread to constantly update realtime.

Constant / Data Structure Detail

Constant	MAX_THREADS 512

	The maximum number of processes that can be used in parallel QualNet. Customers do not receive parallel.cpp, so cannot effectively change this value.
Enumeration	SynchronizationAlgorithm Possible algorithms to use in the parallel runtime. Synchronous is used by default.
Structure	LookaheadLocator This struct is allows us to be able to remove from the LookaheadCalculator's heap. This way lookahead handles can request they be removed. Internally, as the heap re-heapifies these locators are updated.
Structure	EotHeapElement Basic data structure for simplifying lookahead calculation.
Structure	LookaheadCalculator Stores a heap of EOT elements to calculate lookahead.

Function / Macro Detail

Function / Macro	Format
PARALLEL_AllocateLookaheadHandle Obtains a new lookahead handle that allows a protocol to indicate minimum delay values for output. This minimum delay is called EOT - earliest output time.	LookaheadHandle PARALLEL_AllocateLookaheadHandle (Node* node) Parameters: <ul style="list-style-type: none">• node - the active node Returns: <ul style="list-style-type: none">• LookaheadHandle - Returns a reference to the node's lookahead data.
PARALLEL_AddLookaheadHandleToLookaheadCalculator Adds a new LookaheadHandle to the lookahead calculator.	void PARALLEL_AddLookaheadHandleToLookaheadCalculator (Node* node, LookaheadHandle lookaheadHandle, clocktype eotOfNode) Parameters: <ul style="list-style-type: none">• node - the active node• lookaheadHandle - the node's lookahead handle• eotOfNode - the node's EOT Returns: void

	<ul style="list-style-type: none"> - None
PARALLEL_SetLookaheadHandleEOT	<p>Protocols that use EOT will make use of this function more than any other to update the earliest output time as the simulation progresses. Use of EOT is an all-or-nothing option. If your protocol uses EOT, it <u>must</u> use EOT pervasively.</p>
	<p>void PARALLEL_SetLookaheadHandleEOT (Node* node, LookaheadHandle lookaheadHandle, clocktype eot)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - the active node • lookaheadHandle - the node's lookahead handle • eot - the node's current EOT <p>Returns:</p> <ul style="list-style-type: none"> • void - None
PARALLEL_RemoveLookaheadHandleFromLookaheadCalculator	<p>Removes a LookaheadHandle from the lookahead calculator.</p>
	<p>void PARALLEL_RemoveLookaheadHandleFromLookaheadCalculator (Node* node, LookaheadHandle lookaheadHandle, clocktype* eotOfNode)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - the active node • lookaheadHandle - the node's lookahead handle • eotOfNode - the node's current EOT <p>Returns:</p> <ul style="list-style-type: none"> • void - None
PARALLEL_SetMinimumLookaheadForInterface	<p>Sets a minimum delay for messages going out on this interface. This is typically set by the protocol running on that interface.</p>
	<p>void PARALLEL_SetMinimumLookaheadForInterface (Node* node, clocktype minLookahead)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - the active node • minLookahead - the protocol's minimum lookahead <p>Returns:</p> <ul style="list-style-type: none"> • void - None
PARALLEL_InitLookaheadCalculator	<p>Initializes lookahead calculation. For kernel use only.</p>
	<p>void PARALLEL_InitLookaheadCalculator (LookaheadCalculator lookaheadCalculator)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • lookaheadCalculator - the lookahead calculator <p>Returns:</p> <ul style="list-style-type: none"> • void - None

PARALLEL_AssignNodesToPartitions	<p>Using their positions or other information, assigns each node to a partition. For kernel use only.</p>	<p>int PARALLEL_AssignNodesToPartitions (int numNodes, int numberOfPartitions, NodeInput* nodeInput, NodePosition* nodePos, AddressMapType* map)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>numNodes</code> - the number of nodes • <code>numberOfPartitions</code> - the number of partitions • <code>nodeInput</code> - the input configuration file • <code>nodePos</code> - the node positions • <code>map</code> - node ID <--> IP address mappings <p>Returns:</p> <ul style="list-style-type: none"> • <code>int</code> - the number of partitions used
PARALLEL_GetPartitionForNode	<p>Allows parallel code to determine to what partition a node is assigned. If a <code>Node*</code> is available, it's much quicker to just look it up directly</p>	<p>int PARALLEL_GetPartitionForNode (NodeId nodeId)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>nodeId</code> - the node's ID <p>Returns:</p> <ul style="list-style-type: none"> • <code>int</code> - the partition to which the node is assigned
PARALLEL_InitializeParallelRuntime	<p>Sets global variables and stuff. For kernel use only.</p>	<p>void PARALLEL_InitializeParallelRuntime (int numberOfThreads)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>numberOfThreads</code> - the number of processors to use. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
PARALLEL_CreatePartitionThreads	<p>Creates the threads for parallel execution and starts them running. For kernel use only.</p>	<p>void PARALLEL_CreatePartitionThreads (int numberOfThreads, NodeInput* nodeInput, PartitionData* partitionArray)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>numberOfThreads</code> - the number of threads to create. • <code>nodeInput</code> - the input configuration • <code>partitionArray</code> - an array containing the partition data <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
PARALLEL_GetRemoteMessages		void PARALLEL_GetRemoteMessages (PartitionData* partitionData)

	<p>Collects all the messages received from other partitions. For kernel use only.</p> <p>Parameters:</p> <ul style="list-style-type: none"> partitionData - a pointer to the partition <p>Returns:</p> <ul style="list-style-type: none"> void - None
PARALLEL_GetRemoteMessagesAndBarrier	<p>Collects all the messages received from other partitions. This function also acts as a barrier. For kernel use only.</p> <p>void PARALLEL_GetRemoteMessagesAndBarrier (PartitionData* partitionData)</p> <p>Parameters:</p> <ul style="list-style-type: none"> partitionData - a pointer to the partition <p>Returns:</p> <ul style="list-style-type: none"> void - None
PARALLEL_SendRemoteMessages	<p>Sends one or more messages to a remote partition. For kernel use only.</p> <p>PARALLEL_SendRemoteMessages (Message* msgList, PartitionData* partitionData, int partitionId)</p> <p>Parameters:</p> <ul style="list-style-type: none"> msgList - a linked list of Messages partitionData - a pointer to the partition partitionId - the partition's ID <p>Returns:</p> <ul style="list-style-type: none"> - None
PARALLEL_DeliverRemoteMessages	<p>Delivers cached messages to all remote partitions. For kernel use only.</p> <p>PARALLEL_DeliverRemoteMessages (PartitionData* partitionData)</p> <p>Parameters:</p> <ul style="list-style-type: none"> partitionData - a pointer to the partition <p>Returns:</p> <ul style="list-style-type: none"> - None
PARALLEL_SendRemoteMessagesOob	<p>Sends one or more messages to a remote partition. These messages are oob messages and will be processed immediately. For kernel use only.</p> <p>void PARALLEL_SendRemoteMessagesOob (Message* msgList, PartitionData* partitionData, int partitionId, bool isResponse)</p> <p>Parameters:</p> <ul style="list-style-type: none"> msgList - a linked list of Messages partitionData - a pointer to the partition partitionId - the partition's ID isResponse - if it's a response to an OOB message

	<p>Returns:</p> <ul style="list-style-type: none"> • void - None
PARALLEL_SendMessageToAllPartitions	<p>Sends a message to all remote partitions, but not the current one. By default, duplicates will be sent to all remote partitions and the original freed, but if freeMsg is false, the original message will not be freed.</p> <p>void PARALLEL_SendMessageToAllPartitions (Message* msg, PartitionData* partitionData, bool freeMsg)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • msg - the message(s) to send • partitionData - the sending partition • freeMsg - whether or not to free the original <p>Returns:</p> <ul style="list-style-type: none"> • void - None
PARALLEL_SendRemoteLinkMessage	<p>Sends one LINK message to a remote partition.</p> <p>PARALLEL_SendRemoteLinkMessage (Node* node, Message* msg, LinkData* link, clocktype txDelay)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - the sending node • msg - the message to be sent • link - info about the link • txDelay - the transmission delay, not including propagation <p>Returns:</p> <ul style="list-style-type: none"> • - None
PARALLEL_UpdateSafeTime	<p>A generic function for calculating the window of safe events For kernel use only.</p> <p>void PARALLEL_UpdateSafeTime (PartitionData* partitionData)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • partitionData - a pointer to the partition <p>Returns:</p> <ul style="list-style-type: none"> • void - None
PARALLEL_ReturnEarliestGlobalEventTime	<p>Returns the earliest global event time. Required for interfacing to time-sensitive external programs. For kernel use only.</p> <p>clocktype PARALLEL_ReturnEarliestGlobalEventTime (PartitionData* partitionData)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • partitionData - a pointer to the partition <p>Returns:</p>

	<ul style="list-style-type: none"> • <code>clocktype</code> - the time of the earliest event across all partitions
PARALLEL_Exit	<p>Exits from the parallel system, killing threads, etc. For kernel use only.</p> <p>void PARALLEL_Exit (PartitionData* partitionData)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>partitionData</code> - a pointer to the partition <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
PARALLEL_SetProtocolIsNotEOTCapable	<p>Currently, EOT can only be used if supported by all protocols running in the scenario. If any protocol is not capable, only the minimum lookahead is used.</p> <p>void PARALLEL_SetProtocolIsNotEOTCapable (Node* node)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - the node's data <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
PARALLEL_EnableDynamicMobility	<p>Forces the runtime to consider mobility events when calculating EOT/ECOT. Mobility events are ignored by default. This function should be called during the initialization of models where changes in position or direction of one node may affect the behavior of other nodes.</p> <p>void PARALLEL_EnableDynamicMobility ()</p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
PARALLEL_SetGreedy	<p>Tells the kernel to use spin locks on barriers if true, or to use blocking barriers otherwise. In greedy mode, the Simulator needs a dedicated CPU per partition.</p> <p>void PARALLEL_SetGreedy (bool greedy)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>greedy</code> - should it be greedy or not? <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
PARALLEL_IsGreedy	<p>Checks whether SetGreedy has been called.</p> <p>bool PARALLEL_IsGreedy ()</p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> • <code>bool</code> - true if greedy mode is enabled.
PARALLEL_Preflight	<p>Initializes parallel operation.</p> <p>void PARALLEL_Preflight (PartitionData* partitionData)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>partitionData</code> - the partition to initialize.

	<p>Returns:</p> <ul style="list-style-type: none"> • void - None
PARALLEL_ScheduleMessagesOnPartition	<p>Takes a list of messages or an OOB message and schedules them for execution on the current partition. Typically these messages have arrived from a remote partition.</p> <p>void PARALLEL_ScheduleMessagesOnPartition (PartitionData* partitionData, Message* msgList, Message*** oobMessage, bool* gotOobMessage, bool isMT)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • partitionData - the partition. • msgList - a list of normal simulation messages. • oobMessage - an out of bounds message. • gotoOobMessage - returns true if Oob response is received • isMT - is this called from a worker thread <p>Returns:</p> <ul style="list-style-type: none"> • void - None
PARALLEL_EndSimulation	<p>Shuts down the parallel engine, including whatever synchronization is required.</p> <p>void PARALLEL_EndSimulation (PartitionData* partitionData)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • partitionData - the partition to terminate. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
PARALLEL_BuildStatFile	<p>Builds the final stat file when running in parallel node. Should only be called once from partition 0.</p> <p>void PARALLEL_BuildStatFile (int numPartitions, char* statFileName, char* experimentPrefix)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • numPartitions - number of partitions • statFileName - name of stat file • experimentPrefix - experiment prefix <p>Returns:</p> <ul style="list-style-type: none"> • void - None
PARALLEL_NumberOfSynchronizations	<p>Return the number of synchronizations performed per partition</p> <p>void PARALLEL_NumberOfSynchronizations ()</p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> • void - None

PARALLEL_StartRealTimeThread	Tells the kernel to use an independent thread to constantly update realtime.	void PARALLEL_StartRealTimeThread (PartitionData* partitionData)
		<p>Parameters:</p> <ul style="list-style-type: none">partitionData - a pointer to the partition <p>Returns:</p> <ul style="list-style-type: none">void - None



Contact [QualNet Support](#) for questions pertaining to the QualNet API Reference. This document is confidential and proprietary.
It may not be reproduced or distributed without the expressed written consent of [Scalable Network Technologies](#).

QualNet® is a Registered Trademark of [Scalable Network Technologies](#).

Copyright © 2001-2012 [Scalable Network Technologies, Inc.](#). All rights reserved.

QualNet 6.1 API Reference

PARTITION

This file contains declarations of some functions for partition threads.

Constant / Data Structure Summary

Type	Name
CONSTANT	NUM_SIM_TIME_STATUS_PRINTS The number of percentage complete statements to print
CONSTANT	COMMUNICATION_ID_INVALID A default uninitialized communication ID.
CONSTANT	COMMUNICATION_DELAY_REAL_TIME A value to indicate real time interpartition communication
STRUCT	SubnetMemberData Data structure containing interfaceIndex and Node* for a node in a single subnet
STRUCT	PartitionSubnetMemberList Data structure containing member data info for all nodes in a single subnet
STRUCT	PartitionSubnetData Data structure containing subnet member data for all subnets
STRUCT	PartitionData Contains global information for this partition.
STRUCT	SimulationProperties Global properties of the simulation for all partitions.

Function / Macro Summary

Return Type	Summary
void	<p><code>PARTITION_GetTerrainPtr</code>(PartitionData* partitionData)</p> <p>Inline function used to get terrainData pointer.</p>
void	<p><code>PARTITION_CreateEmptyPartition</code>(int partitionId, int numPartitions)</p> <p>Function used to allocate and perform initilaization of an empty partition data structure.</p>
void	<p><code>PARTITION_InitializePartition</code>(PartitionData* partitionData, TerrainData* terrainData, clocktype maxSimClock, double startRealTime, int numNodes, BOOL traceEnabled, AddressMapType* addressMapPtr, NodePositions* nodePositions, NodeInput* nodeInput, int seedVal, int* nodePlacementTypeCounts, char* experimentPrefix, clocktype startsSimClock)</p> <p>Function used to initialize a partition.</p>
void	<p><code>PARTITION_InitializeNodes</code>(PartitionData* partitionData)</p> <p>Function used to allocate and initialize the nodes on a partition.</p>
void	<p><code>PARTITION_Finalize</code>(PartitionData* partitionData)</p> <p>Finalizes the nodes on the partition.</p>
void	<p><code>PARTITION_ProcessPartition</code>(PartitionData* partitionData)</p> <p>Creates and initializes the nodes, then processes events on this partition.</p>
void	<p><code>PARTITION_ProcessSendMT</code>(PartitionData* partitionData)</p> <p>Messages sent by worker threads outside of the main simulation event loop MUST call MESSAGE_SendMT (). This funciton then is the other half - where the multi-thread messages are properly added to the event list.</p>
BOOL	<p><code>PARTITION_ReturnNodePointer</code>(PartitionData* partitionData, Node** node, NodeId nodeId, BOOL remoteOK)</p> <p>Returns a pointer to the node or NULL if the node is not on this partition. If remoteOK is TRUE, returns a pointer to this partition's proxy for a remote node if the node does not belong to this partition. This feature should be used with great care, as the proxy is incomplete. Returns TRUE if the node was successfully found.</p>
void	<p><code>PARTITION_NodeExists</code>(PartitionData* partitionData, NodeId nodeId)</p> <p>Determines whether the node ID exists in the scenario. Must follow node creation.</p>

PARTITION

void	<code>PARTITION_PrintRunTimeStats</code> (PartitionData* partitionData)
	If dynamic statistics reporting is enabled, generates statistics for enabled layers.
void	<code>PARTITION_SchedulePartitionEvent</code> (PartitionData* partitionData, Message* msg, clocktype eventTime, bool scheduleBeforeNodes)
	Schedules a generic partition-level event.
void	<code>PARTITION_HandlePartitionEvent</code> (PartitionData* partitionData, Message* msg)
	An empty function for protocols to use that need to schedule and handle partition-level events.
void	<code>PARTITION_ClientStateSet</code> (PartitionData* partitionData, const char* stateName, void* clientState)
	Sets or replaces a pointer to client-state, identified by name, in the indicated partition. Allows client code, like external interfaces, to store their own data in the partition. The client's state pointer is set and found by name. If the caller passes a name for client state that is already being stored, the state pointer replaces what was already there.
void*	<code>PARTITION_ClientStateFind</code> (PartitionData* partitionData, const char* stateName)
	Looks up the requested client-state by name. Returns NULL if the state isn't present.
CommunicatorId	<code>PARTITION_COMMUNICATION_RegisterCommunicator</code> (PartitionData* partitionData, const char* name, PartitionCommunicationHandler handler)
	Allocates a message id and registers the handler that will be invoked to receive callbacks when messages are with the id are sent.
CommunicatorId	<code>PARTITION_COMMUNICATION_FindCommunicator</code> (PartitionData* partitionData, std name)
	Locate an already registered commincator.
void	<code>PARTITION_COMMUNICATION_SendToPartition</code> (PartitionData* partitionData, int partitionId, Message* msg, clocktype delay)
	Transmit a message to a partition.
void	<code>PARTITION_COMMUNICATION_SendToAllPartitions</code> (PartitionData* partitionData, Message* msg, clocktype delay)
	Transmit a message to all partitions.
std	<code>IO_Return_Qualnet_Directory</code> ()
	This will return in a string the current directory qualnet is executing from
boolean true/false if file exists	<code>IO_SourceFileExists</code> ()

	This will return a boolean true if file exists, and false if not IO_CheckSourceLibrary()
std	This will return in a string the formatted yes/no line for whether the fingerprint file exists for given library IO_ReturnSourceAndCompiledLibraries()
std	This will return in a string a list of libraries currently compiled into product as well as those which have source code available. IO_ReturnExpirationDateFromLicenseFeature()
std	This will return in a string a list of libraries currently compiled into product as well as those which have source code available. IO_ReturnExpirationDateFromNumericalDate()
std	This will return in a string the expiration date of the library IO_ReturnExpirationDateFromNumericalDate()
std	This will return in a string the expiration date of the library IO_ParseFlexLMDate()
UInt64 containing the date	Parse a FlexLM date in a platform safe way IO_ReturnStatusMessageFromLibraryInfo()
std	This will return in a string the status message for the library used with the -print_libraries option IO_ReturnStatusMessageFromLibraryInfo()
	This will return in a string the library name from its index :: because flexlm won't allow std strucsts in main.cpp :: but main.cpp is the only place flex structs are allowed PARTITION_ShowProgress()
	Print standard QualNet progress log

Constant / Data Structure Detail

Constant	<code>NUM_SIM_TIME_STATUS_PRINTS 100</code> The number of percentage complete statements to print
Constant	<code>COMMUNICATION_ID_INVALID 0</code> A default uninitialized communication ID.
Constant	<code>COMMUNICATION_DELAY_REAL_TIME -1</code> A value to indicate real time interpartition communication
Structure	<code>SubnetMemberData</code> Data structure containing interfaceIndex and Node* for a node in a single subnet
Structure	<code>PartitionSubnetMemberList</code> Data structure containing member data info for all nodes in a single subnet
Structure	<code>PartitionSubnetData</code> Data structure containing subnet member data for all subnets
Structure	<code>PartitionData</code> Contains global information for this partition.
Structure	<code>SimulationProperties</code> Global properties of the simulation for all partitions.

Function / Macro Detail

Function / Macro	Format
<code>PARTITION_GetTerrainPtr</code> Inline function used to get terrainData pointer.	Format <code>void PARTITION_GetTerrainPtr (PartitionData* partitionData)</code> Parameters: <ul style="list-style-type: none"> partitionData - pointer to partitionData

	<p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
PARTITION_CreateEmptyPartition	<p>Function used to allocate and perform initialization of an empty partition data structure.</p> <p><code>void PARTITION_CreateEmptyPartition (int partitionId, int numPartitions)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>partitionId</code> - the partition ID, used for parallel • <code>numPartitions</code> - for parallel <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
PARTITION_InitializePartition	<p>Function used to initialize a partition.</p> <p><code>void PARTITION_InitializePartition (PartitionData* partitionData, TerrainData* terrainData, clocktype maxSimClock, double startRealTime, int numNodes, BOOL traceEnabled, AddressMapType* addressMapPtr, NodePositions* nodePositions, NodeInput* nodeInput, int seedVal, int* nodePlacementTypeCounts, char* experimentPrefix, clocktype startSimClock)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>partitionData</code> - an empty partition data structure • <code>terrainData</code> - dimensions, terrain database, etc. • <code>maxSimClock</code> - length of the scenario • <code>startRealTime</code> - for synchronizing with the realtime • <code>numNodes</code> - number of nodes in the simulation • <code>traceEnabled</code> - is packet tracing enabled? • <code>addressMapPtr</code> - contains Node ID <--> IP address mappings • <code>nodePositions</code> - initial node locations and partition assignments • <code>nodeInput</code> - contains all the input parameters • <code>seedVal</code> - the global random seed • <code>nodePlacementTypeCounts</code> - gives information about node placement • <code>experimentPrefix</code> - the experiment name • <code>startSimClock</code> - the simulation starting time <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
PARTITION_InitializeNodes	<code>void PARTITION_InitializeNodes (PartitionData* partitionData)</code>

	<p>Function used to allocate and initialize the nodes on a partition.</p> <p>Parameters:</p> <ul style="list-style-type: none"> partitionData - an pre-initialized partition data structure <p>Returns:</p> <ul style="list-style-type: none"> void - None
PARTITION_Finalize	<p>Finalizes the nodes on the partition.</p> <p>void PARTITION_Finalize (PartitionData* partitionData)</p> <p>Parameters:</p> <ul style="list-style-type: none"> partitionData - an pre-initialized partition data structure <p>Returns:</p> <ul style="list-style-type: none"> void - None
PARTITION_ProcessPartition	<p>Creates and initializes the nodes, then processes events on this partition.</p> <p>void PARTITION_ProcessPartition (PartitionData* partitionData)</p> <p>Parameters:</p> <ul style="list-style-type: none"> partitionData - an pre-initialized partition data structure <p>Returns:</p> <ul style="list-style-type: none"> void - None
PARTITION_ProcessSendMT	<p>Messages sent by worker threads outside of the main simulation event loop MUST call MESSAGE_SendMT (). This function then is the other half - where the multi-thread messages are properly added to the event list.</p> <p>void PARTITION_ProcessSendMT (PartitionData* partitionData)</p> <p>Parameters:</p> <ul style="list-style-type: none"> partitionData - an pre-initialized partition data structure <p>Returns:</p> <ul style="list-style-type: none"> void - None
PARTITION_ReturnNodePointer	<p>Returns a pointer to the node or NULL if the node is not on this partition. If remoteOK is TRUE, returns a pointer to this partition's proxy for a remote node if the node does not belong to this partition. This feature should be used with great care, as the proxy is incomplete. Returns TRUE if the node was successfully found.</p> <p>BOOL PARTITION_ReturnNodePointer (PartitionData* partitionData, Node** node, NodeId nodeId, BOOL remoteOK)</p> <p>Parameters:</p> <ul style="list-style-type: none"> partitionData - an pre-initialized partition data structure node - for returning the node pointer nodeId - the node's ID remoteOK - is it ok to return a pointer to proxy node? <p>Returns:</p> <ul style="list-style-type: none"> BOOL - returns TRUE if the node was successfully found

PARTITION_NodeExists	<p>Determines whether the node ID exists in the scenario. Must follow node creation.</p>	<p>void PARTITION_NodeExists (PartitionData* partitionData, NodeId nodeId)</p> <p>Parameters:</p> <ul style="list-style-type: none"> partitionData - an pre-initialized partition data structure nodeId - the node's ID <p>Returns:</p> <ul style="list-style-type: none"> void - None
PARTITION_PrintRunTimeStats	<p>If dynamic statistics reporting is enabled, generates statistics for enabled layers.</p>	<p>void PARTITION_PrintRunTimeStats (PartitionData* partitionData)</p> <p>Parameters:</p> <ul style="list-style-type: none"> partitionData - an pre-initialized partition data structure <p>Returns:</p> <ul style="list-style-type: none"> void - None
PARTITION_SchedulePartitionEvent	<p>Schedules a generic partition-level event.</p>	<p>void PARTITION_SchedulePartitionEvent (PartitionData* partitionData, Message* msg, clocktype eventTime, bool scheduleBeforeNodes)</p> <p>Parameters:</p> <ul style="list-style-type: none"> partitionData - an pre-initialized partition data structure msg - an event eventTime - the time the event should occur scheduleBeforeNodes - process event before or after node events <p>Returns:</p> <ul style="list-style-type: none"> void - None
PARTITION_HandlePartitionEvent	<p>An empty function for protocols to use that need to schedule and handle partition-level events.</p>	<p>void PARTITION_HandlePartitionEvent (PartitionData* partitionData, Message* msg)</p> <p>Parameters:</p> <ul style="list-style-type: none"> partitionData - an pre-initialized partition data structure msg - an event <p>Returns:</p> <ul style="list-style-type: none"> void - None
PARTITION_ClientStateSet		<p>void PARTITION_ClientStateSet (PartitionData* partitionData, const char* stateName, void* clientState)</p> <p>Parameters:</p>

<p>Sets or replaces a pointer to client-state, identified by name, in the indicated partition. Allows client code, like external interfaces, to store their own data in the partition. The client's state pointer is set and found by name. If the caller passes a name for client state that is already being stored, the state pointer replaces what was already there.</p>	<ul style="list-style-type: none"> partitionData - an pre-initialized partition data structure stateName - Name used to locate this client state clientState - Pointer to whatever data-structure the <p>Returns:</p> <ul style="list-style-type: none"> void - None
<p>PARTITION_ClientStateFind</p> <p>Looks up the requested client-state by name. Returns NULL if the state isn't present.</p>	<p>void* PARTITION_ClientStateFind (PartitionData* partitionData, const char* stateName)</p> <p>Parameters:</p> <ul style="list-style-type: none"> partitionData - an pre-initialized partition data structure stateName - Name used to locate this client state <p>Returns:</p> <ul style="list-style-type: none"> void* - returns the client state
<p>PARTITION_COMMUNICATION_RegisterCommunicator</p> <p>Allocates a message id and registers the handler that will be invoked to receive callbacks when messages are with the id are sent.</p>	<p>CommunicatorId PARTITION_COMMUNICATION_RegisterCommunicator (PartitionData* partitionData, const char* name, PartitionCommunicationHandler handler)</p> <p>Parameters:</p> <ul style="list-style-type: none"> partitionData - an pre-initialized partition data structure name - Your name for this type of message. handler - Function <p>Returns:</p> <ul style="list-style-type: none"> CommunicatorId - used to later when calling MESSAGE_Alloc().
<p>PARTITION_COMMUNICATION_FindCommunicator</p> <p>Locate an already registered commincator.</p>	<p>CommunicatorId PARTITION_COMMUNICATION_FindCommunicator (PartitionData* partitionData, std name)</p> <p>Parameters:</p> <ul style="list-style-type: none"> partitionData - an pre-initialized partition data structure name - string <p>Returns:</p> <ul style="list-style-type: none"> CommunicatorId - found communicator Id or COMMUNICATION_ID_INVALID if not found.
<p>PARTITION_COMMUNICATION_SendToPartition</p> <p>Transmit a message to a partition.</p>	<p>void PARTITION_COMMUNICATION_SendToPartition (PartitionData* partitionData, int partitionId, Message* msg, clocktype delay)</p> <p>Parameters:</p>

	<ul style="list-style-type: none"> partitionData - an pre-initialized partition data structure partitionId - partition to send the message to msg - Message to send. You are required to follow delay - When the message should execute. Special delay <p>Returns:</p> <ul style="list-style-type: none"> void - None
PARTITION_COMMUNICATION_SendToAllPartitions	<p>Transmit a message to all partitions.</p> <p>void PARTITION_COMMUNICATION_SendToAllPartitions (PartitionData* partitionData, Message* msg, clocktype delay)</p> <p>Parameters:</p> <ul style="list-style-type: none"> partitionData - an pre-initialized partition data structure msg - Message to send. You are required to follow delay - When the message should execute. Special delay <p>Returns:</p> <ul style="list-style-type: none"> void - None
IO_Return_Qualnet_Directory	<p>This will return in a string the current directory qualnet is executing from</p> <p>std IO_Return_Qualnet_Directory ()</p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> std - string containing current qualnet directory
IO_SourceFileExists	<p>This will return a boolean true if file exists, and false if not</p> <p>boolean true/false if file exists IO_SourceFileExists ()</p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> boolean true/false if file exists - None
IO_CheckSourceLibrary	<p>This will return in a string the formatted yes/no line for whether the fingerprint file exists for given library</p> <p>std IO_CheckSourceLibrary ()</p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> std - string containing list of libraries
IO_ReturnSourceAndCompiledLibraries	<p>std IO_ReturnSourceAndCompiledLibraries ()</p> <p>Parameters:</p>

This will return in a string a list of libraries currently compiled into product as well as those which have source code available.	Returns: <ul style="list-style-type: none">• std - string containing list of libraries
IO_ReturnExpirationDateFromLicenseFeature This will return in a string a list of libraries currently compiled into product as well as those which have source code available.	std IO_ReturnExpirationDateFromLicenseFeature () Parameters: Returns: <ul style="list-style-type: none">• std - string containing expiration date for this feature
IO_ReturnExpirationDateFromNumericalDate This will return in a string the expiration date of the library	std IO_ReturnExpirationDateFromNumericalDate () Parameters: Returns: <ul style="list-style-type: none">• std - string containing expiration date for this feature
IO_ReturnExpirationDateFromNumericalDate This will return in a string the expiration date of the library	std IO_ReturnExpirationDateFromNumericalDate () Parameters: Returns: <ul style="list-style-type: none">• std - string containing expiration date for this feature
IO_ParseFlexLMDate Parse a FlexLM date in a platform safe way	UInt64 containing the date IO_ParseFlexLMDate () Parameters: Returns: <ul style="list-style-type: none">• UInt64 containing the date - None
IO_ReturnStatusMessageFromLibraryInfo This will return in a string the status message for the library used with the -print_libraries option	std IO_ReturnStatusMessageFromLibraryInfo () Parameters: Returns: <ul style="list-style-type: none">• std - string containing status message for library
IO_ReturnStatusMessageFromLibraryInfo This will return in a string the library name from its index :: because flexlm won't allow std strucsts in main.cpp :: but main.cpp is the only place flex structs are allowed	std IO_ReturnStatusMessageFromLibraryInfo () Parameters: Returns: <ul style="list-style-type: none">• std - string containing library name
PARTITION_ShowProgress	PARTITION_ShowProgress ()

Print standard QualNet progress log

Parameters:

Returns:

- -



Contact [QualNet Support](#) for questions pertaining to the QualNet API Reference. This document is confidential and proprietary.
It may not be reproduced or distributed without the expressed written consent of [Scalable Network Technologies](#).

QualNet® is a Registered Trademark of [Scalable Network Technologies](#).

Copyright © 2001-2012 [Scalable Network Technologies, Inc.](#). All rights reserved.

QualNet 6.1 API Reference

PHYSICAL LAYER

This file describes data structures and functions used by the Physical Layer. Most of this functionality is enabled/used in the Wireless library.

Constant / Data Structure Summary

Type	Name
CONSTANT	PHY_DEFAULT_NOISE_FACTOR Default noise factor in physical medium
CONSTANT	PHY_DEFAULT_TEMPERATURE Default temperature of physical medium.
CONSTANT	PHY_DEFAULT_MIN_PCOM_VALUE Default minimum pcom value threshold
CONSTANT	PHY_DEFAULT_SYNC_COLLISION_WINDOW Default minimum pcom value threshold
ENUMERATION	PhyModel Different phy types supported.
ENUMERATION	PhyRxModel Different types of packet reception model
STRUCT	PhyBerEntry SNR/BER curve entry
STRUCT	PhyBerTable Bit Error Rate table.
STRUCT	PhyPerEntry

	SNR/PER curve entry
STRUCT	PhyPerTable
	Packet Error Rate table.
STRUCT	PhySerEntry
	SNR/PER curve entry
STRUCT	PhySerTable
	Symbol Error Rate table.
STRUCT	PhySignalMeasurement
	Measurement of the signal of received pkt
STRUCT	AntennaModel
	Structure for classifying different types of antennas.
STRUCT	AntennaOmnidirectional
	Structure for an omnidirectional antenna.
STRUCT	PhyPcomItem
	Used by Phy layer to store PCOM values
STRUCT	PhyData
	Structure for phy layer
STRUCT	PacketPhyStatus
	Used by Phy layer to report channel status to mac layer

Function / Macro Summary

Return Type	Summary
-------------	---------

PHYSICAL LAYER

void	<code>PHY_GlobalBerInit</code> (const NodeInput* nodeInput)
	Pre-load all the BER files.
void	<code>PHY_GetSnrBerTableByName</code> (char* tableName)
	Get a pointer to a specific BER table.
int	<code>PHY_GetSnrBerTableIndex</code> (Node* node, int phyIndex)
	Get a index of BER table used by PHY.
int	<code>PHY_SetSnrBerTableIndex</code> (Node* node, int phyIndex)
	Set index of BER table to be used by PHY.
void	<code>PHY_GlobalPerInit</code> (const NodeInput* nodeInput)
	Pre-load all the PER files.
void	<code>PHY_GetSnrPerTableByName</code> (char* tableName)
	Get a pointer to a specific PER table.
void	<code>PHY_GlobalSerInit</code> (const NodeInput* nodeInput)
	Pre-load all the SER files.
void	<code>PHY_GetSnrSerTableByName</code> (char* tableName)
	Get a pointer to a specific SER table.
void	<code>PHY_Init</code> (Node* node, const NodeInput* nodeInput)
	Initialize physical layer
void	<code>PHY_CreateAPhyForMac</code> (Node* node, const NodeInput* nodeInput, int interfaceIndex, int networkAddress, PhyModel phyModel, int* phyNumber)
	Initialization function for the phy layer
void	<code>PHY_Finalize</code> (Node * node)
	Called at the end of simulation to collect the results of the simulation of the Phy Layer.

PHYSICAL LAYER

void	PHY_ProcessEvent (Node* node, Message* msg)	Models the behaviour of the Phy Layer on receiving the message encapsulated in msgHdr
PhyStatusType	PHY_GetStatus (Node * node, int phyNum)	Retrieves the Phy's current status
void	PHY_SetTransmitPower (Node * node, int phyIndex, double newTxPower_mW)	Sets the Radio's transmit power in mW
void	PHY_SetRxSNRThreshold (Node * node, int phyIndex, double snr)	Sets the Radio's Rx SNR Threshold
void	PHY_SetDataRate (Node * node, int phyIndex, int dataRate)	Sets the Radio's Data Rate for both Tx and Rx
void	PHY_SetTxDataRate (Node * node, int phyIndex, int dataRate)	For radios that support different Tx and Rx data rates, this will set the Rx Data Rate. For others, it will call PHY_SetDataRate.
void	PHY_SetRxDataRate (Node * node, int phyIndex, int dataRate)	For radios that support different Tx and Rx data rates, this will set the Rx Data Rate. For others, it will call PHY_SetDataRate.
void	PHY_GetTransmitPower (Node * node, int phyIndex, double* txPower_mW)	Gets the Radio's transmit power in mW.
clocktype	PHY_GetTransmissionDelay (Node * node, int phyIndex, int size)	Get transmission delay based on the first (usually lowest) data rate WARNING: This function call is to be replaced with PHY_GetTransmissionDuration() with an appropriate data rate
clocktype	PHY_GetTransmissionDuration (Node * node, int phyIndex, int dataRateIndex, int size)	Get transmission duration of a structured signal fragment.
PhyModel	PHY_GetFrameModel (Node * node, int phyNum)	Get Physical Model

PHYSICAL LAYER

PhyModel	<code>PHY_GetAntennaModelType</code> (Node * node, int phyNum)
	Get Antenna Model type
void	<code>PHY_StartTransmittingSignal</code> (Node * node, int phyNum, Message* msg, BOOL useMacLayerSpecifiedDelay, clocktype delayUntilAirborne)
	Starts transmitting a packet.
void	<code>PHY_StartTransmittingSignal</code> (Node * node, int phyNum, Message* msg, clocktype duration, BOOL useMacLayerSpecifiedDelay, clocktype delayUntilAirborne)
	Starts transmitting a packet. Function is being overloaded
void	<code>PHY_StartTransmittingSignal</code> (Node * node, int phyNum, Message* msg, int bitSize, BOOL useMacLayerSpecifiedDelay, clocktype delayUntilAirborne)
	Starts transmitting a packet.
void	<code>PHY_SignalArrivalFromChannel</code> (Node * node, int phyIndex, int channelIndex, PropRxInfo* propRxInfo)
	Called when a new signal arrives
void	<code>PHY_SignalEndFromChannel</code> (Node * node, int phyIndex, int channelIndex, PropRxInfo* propRxInfo)
	Called when the current signal ends
int	<code>PHY_GetTxDataRate</code> (Node * node, int phyIndex)
	Get transmission data rate
int	<code>PHY_GetRxDataRate</code> (Node * node, int phyIndex)
	Get reception data rate
int	<code>PHY_GetTxDataRateType</code> (Node * node, int phyIndex)
	Get transmission data rate type
int	<code>PHY_GetRxDataRateType</code> (Node * node, int phyIndex)
	Get reception data rate type
void	<code>PHY_SetTxDataRateType</code> (Node * node, int phyIndex, int dataRateType)

	Set transmission data rate type <code>PHY_SetTxDataRateType(Node* node, int phyIndex, int* dataRateType)</code>
void	Get the lowest transmission data rate type <code>PHY_GetLowestTxDataRateType(Node* node, int phyIndex)</code>
void	Set the lowest transmission data rate type <code>PHY_SetLowestTxDataRateType(Node* node, int phyIndex, int* dataRateType)</code>
void	Get the highest transmission data rate type <code>PHY_SetHighestTxDataRateType(Node* node, int phyIndex)</code>
void	Set the highest transmission data rate type <code>PHY_SetHighestTxDataRateTypeForBC(Node* node, int phyIndex, int* dataRateType)</code>
void	Get the highest transmission data rate type for broadcast <code>PHY_SetHighestTxDataRateTypeForBC(Node* node, int phyIndex)</code>
double	Compute SINR <code>PHY_ComputeSINR(PhyData * phyData, double * signalPower_mW, double* interferencePower_mW, int bandwidth)</code>
void	Compute SINR <code>PHY_SignalInterference(Node* node, int phyIndex, int channelIndex, Message * msg, double * signalPower_mW, double* interferencePower_mW)</code>
double	Compute Power from the desired signal and interference <code>PHY_BER(PhyData * phyData, int berTableIndex, double sinr)</code>
double	Get BER <code>PHY_SER(PhyData * phyData, int perTableIndex, double sinr)</code>
void	Get SER <code>PHY_StopListeningToChannel(Node* node, int phyIndex, int channelIndex)</code>
BOOL	Can Listen To Channel <code>PHY_CanListenToChannel(Node* node, int phyIndex, int channelIndex)</code>

	Check if it can listen to the channel <code>PHY_IsListeningToChannel(Node* node, int phyIndex, int channelIndex)</code>
BOOL	Check if it is listening to the channel <code>PHY_SetTransmissionChannel(Node* node, int phyIndex, int channelIndex)</code>
void	Set the channel index used for transmission <code>PHY_GetTransmissionChannel(Node* node, int phyIndex, int channelIndex)</code>
void	Get the channel index used for transmission <code>PHY_MediumIsIdle(Node* node, int phyNum)</code>
BOOL	Check if the medium is idle <code>PHY_MediumIsIdleInDirection(Node* node, int phyNum, double azimuth)</code>
void	Check if the medium is idle if sensed directionally <code>PHY_SetSensingDirection(Node* node, int phyNum, double azimuth)</code>
void	Set the sensing direction <code>PHY_StartTransmittingSignalDirectionally(Node* node, int phyNum, Message* msg, BOOL useMacLayerSpecifiedDelay, clocktype delayUntilAirborne, double directionAzimuth)</code>
void	Start transmitting a signal directionally <code>PHY_LockAntennaDirection(Node* node, int phyNum)</code>
void	Lock the direction of antenna <code>PHY_UnlockAntennaDirection(Node* node, int phyNum)</code>
double	Unlock the direction of antenna <code>PHY_GetLastSignalsAngleOfArrival(Node* node, int phyNum)</code>
void	Get the AOA of the last signal <code>PHY_TerminateCurrentReceive(Node* node, int phyNum, const BOOL terminateOnlyOnReceiveError, BOOL* receiveError, clocktype* endSignalTime)</code>

		Terminate the current signal reception
double		<code>PHY_PropagationRange(Node* txnode, Node* node, int txInterfaceIndex, int interfaceIndex, int channelIndex, BOOL printAll)</code>
		Calculates an estimated radio range for the PHY. Supports only TWO-RAY and FREE-SPACE.
void		<code>ENERGY_Init(Node* node, const int phyIndex, NodeInput* nodeInput)</code>
		This function declares energy model variables and initializes them. Moreover, the function read energy model specifications and configures the parameters which are configurable.
void		<code>ENERGY_PrintStats(Node* node, const int phyIndex)</code>
		To print the statistic of Energy Model.
void		<code>Phy_ReportStatusToEnergyModel(Node* node, const int phyIndex, PhyStatusType prevStatus, PhyStatusType newStatus)</code>
		This function should be called whenever a state transition occurs in any place in PHY layer. As input parameters, the function reads the current state and the new state of PHY layer and based on the new states calculates the cost of the load that should be taken off the battery. The function then interacts with battery model and updates the charge of battery.
void		<code>Generic_UpdateCurrentLoad(Node* node, const int phyIndex)</code>
		To update the current load of generic energy model.
void		<code>PHY_NotificationOfPacketDrop(Node* node, int phyIndex, int channelIndex, const Message* msg, const string& dropType, double rxPower_mW, double interferencePower_mW, double passloss_dB)</code>
		To Notify the StatsDB module and other modules of the packet dropping event.
void		<code>PHY_NotificationOfSignalReceived(Node* node, int phyIndex, int channelIndex, const Message* msg, double rxPower_mW, double interferencePower_mW, double passloss_dB, int controlSize)</code>
		To Notify the StatsDB module and other modules of the signal received event .
void		<code>PHY_GetSteeringAngle(Node* node, int phyIndex)</code>
		Gets the current steering angle for a directional antenna from PHY models that support this.
double		<code>PHY_GetBandwidth(Node* node, int phyIndex)</code>
		To get the bandwidth for the given PHY model.
double		<code>PHY_GetFrequency(Node* node, int channelIndex)</code>

	To get the frequency for the given signal. <code>PHY_GetPhyModel</code> (Node* node, int phyIndex)
PhyModel	To get the PhyModel for the node. <code>PHY_isSignalFeatureMatchReceiverPhyModel</code> (Node* node, int phyIndex)
BOOL	To check if the signal feature matches the receiver's phyModel.
double	<code>PHY_ComputeInbandPower</code> (double signalPower_mW, double signalFrequency, double signalBandwidth, double rxFrequency, double rxBandwidth)
void	To estimate the inband signal power for given signal and receiver parameters. <code>PHY_InterferenceArrivalFromChannel</code> (Node * node, int phyIndex, int channelIndex, PropRxInfo* propRxInfo, double sigPower_mW)
void	Called when a interference signal arrives <code>PHY_InterferenceEndFromChannel</code> (Node * node, int phyIndex, int channelIndex, PropRxInfo* propRxInfo, double sigPower_mW)
	Called when a interference signal ends

Constant / Data Structure Detail

Constant	PHY_DEFAULT_NOISE_FACTOR PHY_DEFAULT_NOISE_FACTOR 10.0 Default noise factor in physical medium
Constant	PHY_DEFAULT_TEMPERATURE PHY_DEFAULT_TEMPERATURE 290.0 Default temperature of physical medium.
Constant	PHY_DEFAULT_MIN_PCOM_VALUE 0.0 Default minimum pcom value threshold
Constant	PHY_DEFAULT_SYNC_COLLISION_WINDOW 1ms

	Default minimum pcom value threshold
Enumeration	<p>PhyModel</p> <p>Different phy types supported.</p>
Enumeration	<p>PhyRxModel</p> <p>Different types of packet reception model</p>
Structure	<p>PhyBerEntry</p> <p>SNR/BER curve entry</p>
Structure	<p>PhyBerTable</p> <p>Bit Error Rate table.</p>
Structure	<p>PhyPerEntry</p> <p>SNR/PER curve entry</p>
Structure	<p>PhyPerTable</p> <p>Packet Error Rate table.</p>
Structure	<p>PhySerEntry</p> <p>SNR/PER curve entry</p>
Structure	<p>PhySerTable</p> <p>Symbol Error Rate table.</p>
Structure	<p>PhySignalMeasurement</p> <p>Measurement of the signal of received pkt</p>
Structure	AntennaModel

	Structure for classifying different types of antennas.
Structure	AntennaOmnidirectional Structure for an omnidirectional antenna.
Structure	PhyPcomItem Used by Phy layer to store PCOM values
Structure	PhyData Structure for phy layer
Structure	PacketPhyStatus Used by Phy layer to report channel status to mac layer

Function / Macro Detail

Function / Macro	Format
PHY_GlobalBerInit Pre-load all the BER files.	void PHY_GlobalBerInit (const NodeInput* nodeInput) Parameters: <ul style="list-style-type: none">• nodeInput - structure containing contents of input file Returns: <ul style="list-style-type: none">• void - None
PHY_GetSnrBerTableByName Get a pointer to a specific BER table.	void PHY_GetSnrBerTableByName (char* tableName) Parameters: <ul style="list-style-type: none">• tableName - name of the BER file Returns: <ul style="list-style-type: none">• void - None
PHY_GetSnrBerTableIndex Get a index of BER table used by PHY.	int PHY_GetSnrBerTableIndex (Node* node, int phyIndex) Parameters: <ul style="list-style-type: none">• node - Node pointer

	<ul style="list-style-type: none"> • phyIndex - interface Index <p>Returns:</p> <ul style="list-style-type: none"> • int - None
PHY_SetSnrBerTableIndex	<p>int PHY_SetSnrBerTableIndex (Node* node, int phyIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Node pointer • phyIndex - interface Index <p>Returns:</p> <ul style="list-style-type: none"> • int - None
PHY_GlobalPerInit	<p>void PHY_GlobalPerInit (const NodeInput* nodeInput)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • nodeInput - structure containing contents of input file <p>Returns:</p> <ul style="list-style-type: none"> • void - None
PHY_GetSnrPerTableByName	<p>void PHY_GetSnrPerTableByName (char* tableName)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • tableName - name of the PER file <p>Returns:</p> <ul style="list-style-type: none"> • void - None
PHY_GlobalSerInit	<p>void PHY_GlobalSerInit (const NodeInput* nodeInput)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • nodeInput - structure containing contents of input file <p>Returns:</p> <ul style="list-style-type: none"> • void - None
PHY_GetSnrSerTableByName	<p>void PHY_GetSnrSerTableByName (char* tableName)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • tableName - name of the SER file <p>Returns:</p>

	<ul style="list-style-type: none"> • void - None
PHY_Init	<p>void PHY_Init (Node* node, const NodeInput* nodeInput)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node being initialized • nodeInput - structure containing contents of input file <p>Returns:</p> <ul style="list-style-type: none"> • void - None
PHY_CreateAPhyForMac	<p>void PHY_CreateAPhyForMac (Node* node, const NodeInput* nodeInput, int interfaceIndex, int networkAddress, PhyModel phyModel, int* phyNumber)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node being initialized • nodeInput - structure containing contents of input file • interfaceIndex - interface being initialized. • networkAddress - address of the interface. • phyModel - Which phisical model is used. • phyNumber - returned value to be used as phyIndex <p>Returns:</p> <ul style="list-style-type: none"> • void - None
PHY_Finalize	<p>void PHY_Finalize (Node * node)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node for which results are to be collected <p>Returns:</p> <ul style="list-style-type: none"> • void - None
PHY_ProcessEvent	<p>void PHY_ProcessEvent (Node* node, Message* msg)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node which received the message • msg - message received by the layer <p>Returns:</p>

	<ul style="list-style-type: none"> • void - None
PHY_GetStatus	<p>PhyStatusType PHY_GetStatus (Node * node, int phyNum)</p> <p>Parameters:</p> <p>Retrieves the Phy's current status</p> <ul style="list-style-type: none"> • node - node for which stats are to be collected • phyNum - interface for which stats are to be collected <p>Returns:</p> <ul style="list-style-type: none"> • PhyStatusType - status of interface.
PHY_SetTransmitPower	<p>void PHY_SetTransmitPower (Node * node, int phyIndex, double newTxPower_mW)</p> <p>Parameters:</p> <p>Sets the Radio's transmit power in mW</p> <ul style="list-style-type: none"> • node - node for which transmit power is to be set • phyIndex - interface for which transmit power is to be set • newTxPower_mW - transmit power(mW) <p>Returns:</p> <ul style="list-style-type: none"> • void - None
PHY_SetRxSNRThreshold	<p>void PHY_SetRxSNRThreshold (Node * node, int phyIndex, double snr)</p> <p>Parameters:</p> <p>Sets the Radio's Rx SNR Threshold</p> <ul style="list-style-type: none"> • node - node for which transmit power is to be set • phyIndex - interface for which transmit power is to be set • snr - threshold value to be set <p>Returns:</p> <ul style="list-style-type: none"> • void - None
PHY_SetDataRate	<p>void PHY_SetDataRate (Node * node, int phyIndex, int dataRate)</p> <p>Parameters:</p> <p>Sets the Radio's Data Rate for both Tx and Rx</p> <ul style="list-style-type: none"> • node - node for which transmit power is to be set • phyIndex - interface for which transmit power is to be set • dataRate - dataRate value to be set <p>Returns:</p>

	<ul style="list-style-type: none"> • void - None
PHY_SetTxDataRate	<p>For radios that support different Tx and Rx data rates, this will set the Rx Data Rate. For others, it will call <code>PHY_SetDataRate</code>.</p> <p>void <code>PHY_SetTxDataRate</code> (Node * node, int phyIndex, int dataRate)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node for which transmit power is to be set • phyIndex - interface for which transmit power is to be set • dataRate - dataRate value to be set <p>Returns:</p> <ul style="list-style-type: none"> • void - None
PHY_SetRxDataRate	<p>For radios that support different Tx and Rx data rates, this will set the Rx Data Rate. For others, it will call <code>PHY_SetDataRate</code>.</p> <p>void <code>PHY_SetRxDataRate</code> (Node * node, int phyIndex, int dataRate)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node for which transmit power is to be set • phyIndex - interface for which transmit power is to be set • dataRate - dataRate value to be set <p>Returns:</p> <ul style="list-style-type: none"> • void - None
PHY_GetTransmitPower	<p>Gets the Radio's transmit power in mW.</p> <p>void <code>PHY_GetTransmitPower</code> (Node * node, int phyIndex, double* txPower_mW)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Node that is • phyIndex - interface index. • txPower_mW - transmit power(mW) <p>Returns:</p> <ul style="list-style-type: none"> • void - None
PHY_GetTransmissionDelay	<p>Get transmission delay based on the first (usually lowest) data rate WARNING: This function call is to be replaced with <code>PHY_GetTransmissionDuration()</code> with an appropriate data rate</p> <p>clocktype <code>PHY_GetTransmissionDelay</code> (Node * node, int phyIndex, int size)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node pointer to node • phyIndex - interface index • size - size of the frame in bytes <p>Returns:</p>

	<ul style="list-style-type: none"> • <code>clocktype</code> - transmission delay.
PHY_GetTransmissionDuration	<p>clocktype PHY_GetTransmissionDuration (Node * node, int phyIndex, int dataRateIndex, int size)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node pointer to node • <code>phyIndex</code> - interface index. • <code>dataRateIndex</code> - data rate. • <code>size</code> - size of frame in bytes. <p>Returns:</p> <ul style="list-style-type: none"> • <code>clocktype</code> - transmission duration
PHY_GetFrameModel	<p>PhyModel PHY_GetFrameModel (Node * node, int phyNum)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node pointer to node • <code>phyNum</code> - interface index <p>Returns:</p> <ul style="list-style-type: none"> • <code>PhyModel</code> - Physical Model
PHY_GetAntennaModelType	<p>PhyModel PHY_GetAntennaModelType (Node * node, int phyNum)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node pointer to node • <code>phyNum</code> - interface index <p>Returns:</p> <ul style="list-style-type: none"> • <code>PhyModel</code> - Physical Model
PHY_StartTransmittingSignal	<p>void PHY_StartTransmittingSignal (Node * node, int phyNum, Message* msg, BOOL useMacLayerSpecifiedDelay, clocktype delayUntilAirborne)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node pointer to node • <code>phyNum</code> - interface index • <code>msg</code> - packet to be sent • <code>useMacLayerSpecifiedDelay</code> - use delay specified by MAC

	<ul style="list-style-type: none"> • <code>delayUntilAirborne</code> - delay until airborne <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
PHY_StartTransmittingSignal	<p>Starts transmitting a packet. Function is being overloaded</p> <p><code>void PHY_StartTransmittingSignal (Node * node, int phyNum, Message* msg, clocktype duration, BOOL useMacLayerSpecifiedDelay, clocktype delayUntilAirborne)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node pointer to node • <code>phyNum</code> - interface index • <code>msg</code> - packet to be sent • <code>duration</code> - specified transmission delay • <code>useMacLayerSpecifiedDelay</code> - use delay specified by MAC • <code>delayUntilAirborne</code> - delay until airborne <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
PHY_StartTransmittingSignal	<p>Starts transmitting a packet.</p> <p><code>void PHY_StartTransmittingSignal (Node * node, int phyNum, Message* msg, int bitSize, BOOL useMacLayerSpecifiedDelay, clocktype delayUntilAirborne)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node pointer to node • <code>phyNum</code> - interface index • <code>msg</code> - packet to be sent • <code>bitSize</code> - specified size of the packet in bits • <code>useMacLayerSpecifiedDelay</code> - use delay specified by MAC • <code>delayUntilAirborne</code> - delay until airborne <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
PHY_SignalArrivalFromChannel	<p>Called when a new signal arrives</p> <p><code>void PHY_SignalArrivalFromChannel (Node * node, int phyIndex, int channelIndex, PropRxInfo* propRxInfo)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node pointer to node

	<p>phyIndex - interface index</p> <ul style="list-style-type: none"> • channelIndex - channel index • propRxInfo - information on the arrived signal <p>Returns:</p> <ul style="list-style-type: none"> • void - None
PHY_SignalEndFromChannel	<p>void PHY_SignalEndFromChannel (Node * node, int phyIndex, int channelIndex, PropRxInfo* propRxInfo)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node pointer to node • phyIndex - interface index • channelIndex - channel index • propRxInfo - information on the arrived signal <p>Returns:</p> <ul style="list-style-type: none"> • void - None
Called when the current signal ends	
PHY_GetTxDataRate	<p>int PHY_GetTxDataRate (Node * node, int phyIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node pointer to node • phyIndex - interface index <p>Returns:</p> <ul style="list-style-type: none"> • int - None
Get transmission data rate	
PHY_GetRxDataRate	<p>int PHY_GetRxDataRate (Node * node, int phyIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node pointer to node • phyIndex - interface index <p>Returns:</p> <ul style="list-style-type: none"> • int - None
Get reception data rate	
PHY_GetTxDataRateType	<p>int PHY_GetTxDataRateType (Node * node, int phyIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node pointer to node
Get transmission data rate type	

	<ul style="list-style-type: none"> • phyIndex - interface index <p>Returns:</p> <ul style="list-style-type: none"> • int - None
PHY_GetRxDataRateType	<p>int PHY_GetRxDataRateType (Node * node, int phyIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node pointer to node • phyIndex - interface index <p>Returns:</p> <ul style="list-style-type: none"> • int - None
PHY_SetTxDataRateType	<p>void PHY_SetTxDataRateType (Node * node, int phyIndex, int dataRateType)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node pointer to node • phyIndex - interface index • dataRateType - rate of data <p>Returns:</p> <ul style="list-style-type: none"> • void - None
PHY_GetLowestTxDataRateType	<p>void PHY_GetLowestTxDataRateType (Node* node, int phyIndex, int* dataRateType)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node pointer to node • phyIndex - interface index • dataRateType - rate of data <p>Returns:</p> <ul style="list-style-type: none"> • void - None
PHY_SetLowestTxDataRateType	<p>void PHY_SetLowestTxDataRateType (Node* node, int phyIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node pointer to node • phyIndex - interface index

	<p>Returns:</p> <ul style="list-style-type: none"> • void - None
PHY_GetHighestTxDataRateType	<p>Get the highest transmission data rate type</p> <p>void PHY_GetHighestTxDataRateType (Node* node, int phyIndex, int* dataRateType)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node pointer to node • phyIndex - interface index • dataRateType - rate of data <p>Returns:</p> <ul style="list-style-type: none"> • void - None
PHY_SetHighestTxDataRateType	<p>Set the highest transmission data rate type</p> <p>void PHY_SetHighestTxDataRateType (Node* node, int phyIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node pointer to node • phyIndex - interface index <p>Returns:</p> <ul style="list-style-type: none"> • void - None
PHY_GetHighestTxDataRateTypeForBC	<p>Get the highest transmission data rate type for broadcast</p> <p>void PHY_GetHighestTxDataRateTypeForBC (Node* node, int phyIndex, int* dataRateType)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node pointer to node • phyIndex - interface index • dataRateType - rate of data <p>Returns:</p> <ul style="list-style-type: none"> • void - None
PHY_SetHighestTxDataRateTypeForBC	<p>Set the highest transmission data rate type for broadcast</p> <p>void PHY_SetHighestTxDataRateTypeForBC (Node* node, int phyIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - node pointer to node • phyIndex - interface index <p>Returns:</p> <ul style="list-style-type: none"> • void - None

PHY_ComputeSINR	<p>double PHY_ComputeSINR (PhyData * phyData, double * signalPower_mW, double* interferencePower_mW, int bandwidth)</p> <p>Compute SINR</p> <p>Parameters:</p> <ul style="list-style-type: none"> • phyData - PHY layer data • signalPower_mW - Signal power • interferencePower_mW - Interference power • bandwidth - Bandwidth <p>Returns:</p> <ul style="list-style-type: none"> • double - Signal to Interference and Noise Ratio
PHY_SignalInterference	<p>void PHY_SignalInterference (Node* node, int phyIndex, int channelIndex, Message * msg, double * signalPower_mW, double* interferencePower_mW)</p> <p>Compute Power from the desired signal and interference</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Node that is being • phyIndex - interface number • channelIndex - channel index • msg - message including desired signal • signalPower_mW - power from the desired signal • interferencePower_mW - power from interfering signals <p>Returns:</p> <ul style="list-style-type: none"> • void - None
PHY_BER	<p>double PHY_BER (PhyData * phyData, int berTableIndex, double sinr)</p> <p>Get BER</p> <p>Parameters:</p> <ul style="list-style-type: none"> • phyData - PHY layer data • berTableIndex - index for BER tables • sinr - Signal to Interference and Noise Ratio <p>Returns:</p> <ul style="list-style-type: none"> • double - Bit Error Rate
PHY_SER	double PHY_SER (PhyData * phyData, int perTableIndex, double sinr)

Get SER	<p>Parameters:</p> <ul style="list-style-type: none"> • phyData - PHY layer data • perTableIndex - index for SER tables • sinr - Signal to Interference and Noise Ratio <p>Returns:</p> <ul style="list-style-type: none"> • double - Packet Error Rate
PHY_StopListeningToChannel	<p>void PHY_StopListeningToChannel (Node* node, int phyIndex, int channelIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Node that is being • phyIndex - interface number • channelIndex - channel index <p>Returns:</p> <ul style="list-style-type: none"> • void - None
Check if it can listen to the channel PHY_CanListenToChannel	<p>BOOL PHY_CanListenToChannel (Node* node, int phyIndex, int channelIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Node that is being • phyIndex - interface number • channelIndex - channel index <p>Returns:</p> <ul style="list-style-type: none"> • BOOL - None
Check if it is listening to the channel PHY_IsListeningToChannel	<p>BOOL PHY_IsListeningToChannel (Node* node, int phyIndex, int channelIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Node that is being • phyIndex - interface number • channelIndex - channel index <p>Returns:</p> <ul style="list-style-type: none"> • BOOL - None
PHY_SetTransmissionChannel	<p>void PHY_SetTransmissionChannel (Node* node, int phyIndex, int channelIndex)</p>

	<p>Set the channel index used for transmission</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Node that is being • phyIndex - interface number • channelIndex - channel index <p>Returns:</p> <ul style="list-style-type: none"> • void - None
PHY_GetTransmissionChannel	<p>Get the channel index used for transmission</p> <p>void PHY_GetTransmissionChannel (Node* node, int phyIndex, int channelIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Node that is being • phyIndex - interface number • channelIndex - channel index <p>Returns:</p> <ul style="list-style-type: none"> • void - None
PHY_MediumIsIdle	<p>Check if the medium is idle</p> <p>BOOL PHY_MediumIsIdle (Node* node, int phyNum)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Node that is being • phyNum - interface number <p>Returns:</p> <ul style="list-style-type: none"> • BOOL - None
PHY_MediumIsIdleInDirection	<p>Check if the medium is idle if sensed directionally</p> <p>BOOL PHY_MediumIsIdleInDirection (Node* node, int phyNum, double azimuth)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Node that is being • phyNum - interface number • azimuth - azimuth (in degrees) <p>Returns:</p> <ul style="list-style-type: none"> • BOOL - None
PHY_SetSensingDirection	<p>void PHY_SetSensingDirection (Node* node, int phyNum, double azimuth)</p>

	<p>Set the sensing direction</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Node that is being • phyNum - interface number • azimuth - azimuth (in degrees) <p>Returns:</p> <ul style="list-style-type: none"> • void - None
PHY_StartTransmittingSignalDirectionally	<p>Start transmitting a signal directionally</p> <pre>void PHY_StartTransmittingSignalDirectionally (Node* node, int phyNum, Message* msg, BOOL useMacLayerSpecifiedDelay, clocktype delayUntilAirborne, double directionAzimuth)</pre> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Node that is • phyNum - interface number • msg - signal to transmit • useMacLayerSpecifiedDelay - use delay specified by MAC • delayUntilAirborne - delay until airborne • directionAzimuth - azimuth to transmit the signal <p>Returns:</p> <ul style="list-style-type: none"> • void - None
PHY_LockAntennaDirection	<p>Lock the direction of antenna</p> <pre>void PHY_LockAntennaDirection (Node* node, int phyNum)</pre> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Node that is being • phyNum - interface number <p>Returns:</p> <ul style="list-style-type: none"> • void - None
PHY_UnlockAntennaDirection	<p>Unlock the direction of antenna</p> <pre>void PHY_UnlockAntennaDirection (Node* node, int phyNum)</pre> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Node that is being • phyNum - interface number <p>Returns:</p>

	<ul style="list-style-type: none"> • void - None
PHY_GetLastSignalsAngleOfArrival Get the AOA of the last signal	<p>double PHY_GetLastSignalsAngleOfArrival (Node* node, int phyNum)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Node that is being • phyNum - interface number <p>Returns:</p> <ul style="list-style-type: none"> • double - AOA
PHY_TerminateCurrentReceive Terminate the current signal reception	<p>void PHY_TerminateCurrentReceive (Node* node, int phyNum, const BOOL terminateOnlyOnReceiveError, BOOL* receiveError, clocktype* endSignalTime)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - Node pointer that the • phyNum - interface number • terminateOnlyOnReceiveError - terminate only when • receiveError - if error happened • endSignalTime - end of signal <p>Returns:</p> <ul style="list-style-type: none"> • void - None
PHY_PropagationRange Calculates an estimated radio range for the PHY. Supports only TWO-RAY and FREE-SPACE.	<p>double PHY_PropagationRange (Node* txnode, Node* node, int txInterfaceIndex, int interfaceIndex, int channelIndex, BOOL printAll)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • txnode - the Tx node of interest • node - the Rx node of interest • txInterfaceIndex - the interface for the TX node • interfaceIndex - the interface for the Rx node • channelIndex - the index of the channel • printAll - if TRUE, prints the range for all data <p>Returns:</p> <ul style="list-style-type: none"> • double - the range in meters
ENERGY_Init	void ENERGY_Init (Node* node, const int phyIndex, NodeInput* nodeInput)

	<p>This function declares energy model variables and initializes them. Moreover, the function read energy model specifications and configures the parameters which are configurable.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - the node of interest. • phyIndex - the PHY index. • nodeInput - the node input. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
ENERGY_PrintStats	<p>To print the statistic of Energy Model.</p> <p>void ENERGY_PrintStats (Node* node, const int phyIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - the node of interest. • phyIndex - the PHY index. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
Phy_ReportStatusToEnergyModel	<p>This function should be called whenever a state transition occurs in any place in PHY layer. As input parameters, the function reads the current state and the new state of PHY layer and based on the new states calculates the cost of the load that should be taken off the battery. The function then interacts with battery model and updates the charge of battery.</p> <p>void Phy_ReportStatusToEnergyModel (Node* node, const int phyIndex, PhyStatusType prevStatus, PhyStatusType newStatus)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - the node of interest. • phyIndex - the PHY index. • prevStatus - the the previous status. • newStatus - the the new status. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
Generic_UpdateCurrentLoad	<p>To update the current load of generic energy model.</p> <p>void Generic_UpdateCurrentLoad (Node* node, const int phyIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - the node of interest. • phyIndex - the PHY index. <p>Returns:</p> <ul style="list-style-type: none"> • void - None
PHY_NotificationOfPacketDrop	void PHY_NotificationOfPacketDrop (Node* node, int phyIndex, int channelIndex, const Message* msg, const

	<p>To Notify the StatsDB module and other modules of the packet dropping event.</p> <p><code>string& dropType, double rxPower_mW, double interferencePower_mW, double passloss_dB)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - the node of interest. • <code>phyIndex</code> - the PHY index. • <code>channelIndex</code> - the channelIndex • <code>msg</code> - The dropped message • <code>dropType</code> - the reason for the drop • <code>rxPower_mW</code> - receiving power of the signal • <code>interferencePower_mW</code> - interference power of the signal • <code>passloss_dB</code> - pathloss value of the signal <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
PHY_NotificationOfSignalReceived	<p>To Notify the StatsDB module and other modules of the signal received event .</p> <p><code>void PHY_NotificationOfSignalReceived (Node* node, int phyIndex, int channelIndex, const Message* msg, double rxPower_mW, double interferencePower_mW, double passloss_dB, int controlSize)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - the node of interest. • <code>phyIndex</code> - the PHY index. • <code>channelIndex</code> - the channelIndex • <code>msg</code> - The dropped message • <code>rxPower_mW</code> - receiving power of the signal • <code>interferencePower_mW</code> - interference power of the signal • <code>passloss_dB</code> - pathloss value of the signal • <code>controlSize</code> - size of control header <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
PHY_GetSteeringAngle	<p>Gets the current steering angle for a directional antenna from PHY models that support this.</p> <p><code>void PHY_GetSteeringAngle (Node* node, int phyIndex)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node being used

	<ul style="list-style-type: none"> • <code>phyIndex</code> - physical to be initialized <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
PHY_GetBandwidth	<p><code>double PHY_GetBandwidth (Node* node, int phyIndex)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - The node of interest. • <code>phyIndex</code> - The PHY index. <p>Returns:</p> <ul style="list-style-type: none"> • <code>double</code> - The bandwidth
PHY_GetFrequency	<p><code>double PHY_GetFrequency (Node* node, int channelIndex)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - The node of interest. • <code>channelIndex</code> - Index of the propagation channel <p>Returns:</p> <ul style="list-style-type: none"> • <code>double</code> - The frequency
PHY_GetPhyModel	<p><code>PhyModel PHY_GetPhyModel (Node* node, int phyIndex)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - The node of interest. • <code>phyIndex</code> - The PHY index. <p>Returns:</p> <ul style="list-style-type: none"> • <code>PhyModel</code> - The PhyModel
PHY_isSignalFeatureMatchReceiverPhyModel	<p><code>BOOL PHY_isSignalFeatureMatchReceiverPhyModel (Node* node, int phyIndex)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - The node of interest. • <code>phyIndex</code> - The PHY index. <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - if the signal feature matches the receiver's phyModel
PHY_ComputeInbandPower	<code>double PHY_ComputeInbandPower (double signalPower_mW, double signalFrequency, double signalBandwidth,</code>

	<p>To estimate the inband signal power for given signal and receiver parameters.</p> <p>double rxFrequency, double rxBandwidth)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>signalPower_mW</code> - The signal power in mW. • <code>signalFrequency</code> - The signal frequency in Hz. • <code>signalBandwidth</code> - The signal bandwidth in Hz • <code>rxFrequency</code> - The receiver frequency in Hz • <code>rxBandwidth</code> - The receiver bandwidth in Hz <p>Returns:</p> <ul style="list-style-type: none"> • <code>double</code> - The inband signal power in mW
PHY_InterferenceArrivalFromChannel Called when a interference signal arrives	<p>void <code>PHY_InterferenceArrivalFromChannel</code> (Node * node, int phyIndex, int channelIndex, PropRxInfo* propRxInfo, double sigPower_mW)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node pointer to node • <code>phyIndex</code> - interface index • <code>channelIndex</code> - channel index • <code>propRxInfo</code> - information on the arrived signal • <code>sigPower_mW</code> - The inband interference power in mW <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
PHY_InterferenceEndFromChannel Called when a interference signal ends	<p>void <code>PHY_InterferenceEndFromChannel</code> (Node * node, int phyIndex, int channelIndex, PropRxInfo* propRxInfo, double sigPower_mW)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - node pointer to node • <code>phyIndex</code> - interface index • <code>channelIndex</code> - channel index • <code>propRxInfo</code> - information on the arrived signal • <code>sigPower_mW</code> - The inband interference power in mW <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None



Contact [QualNet Support](#) for questions pertaining to the QualNet API Reference. This document is confidential and proprietary.
It may not be reproduced or distributed without the expressed written consent of [Scalable Network Technologies](#).

QualNet® is a Registered Trademark of [Scalable Network Technologies](#).

Copyright © 2001-2012 [Scalable Network Technologies, Inc.](#). All rights reserved.

QualNet 6.1 API Reference

PROPAGATION

This file describes data structures and functions used by propagation models.

Constant / Data Structure Summary

Type	Name
CONSTANT	BOLTZMANN_CONSTANT Boltzmann constant
CONSTANT	NEGATIVE_PATHLOSS_DB Path loss in dB (used as an invalid value)
CONSTANT	SPEED_OF_LIGHT Defines the value of speed of light
CONSTANT	PROP_DEFAULT_PROPAGATION_LIMIT_dBm Default value for propagation limit.
CONSTANT	PROP_DEFAULT_SHADOWING_MEAN_db Default mean value for shadowing in dB
CONSTANT	MAX_NUM_ELEVATION_SAMPLES Maximum number of sample would be taken.
CONSTANT	PROP_DEFAULT_BANDWIDTH_FACTOR The bandwidth factor that is used to get the half sum bandwidth.
ENUMERATION	PathlossModel Different type of path loss.
ENUMERATION	ShadowingModel

	Different type of shadowing used.
ENUMERATION	FadingModel
	Different type of fading used.
ENUMERATION	propagationEnvironment
	Different type of propagation environment.
ENUMERATION	LoSIndicator
	Indicated if the path is Line of sight OR non-Line of sight
ENUMERATION	SuburbanTerrainType
	Terrain types for Suburban-foliage model
ENUMERATION	IndoorLinkType
	Link types for Indoor model
ENUMERATION	LinkType
	Link types for model
STRUCT	PropPathProfile
	Structure that keeps track of all propertice of a path.
STRUCT	PropChannel
	structure of a channel.
STRUCT	PropProfile
	Main structure of propagation profile
STRUCT	PropData
	Main structure of propagation data.
STRUCT	PropTxInfo

	This structure is used for fields related to channel layer information that need to be sent with a message.
STRUCT	PropRxInfo This structure is used for fields related to channel layer information that need to be received with a message.

Function / Macro Summary

Return Type	Summary
MACRO	PROP_NumberChannels(node) Get the number of channel.
MACRO	PROP_ChannelWavelength(node, channelIndex) Get wavelength of channel having index channelIndex
void	PROP_GlobalInit (PartitionData* partitionData, NodeInput* nodeInput) Initialization function for propagation This function is called from each partition, not from each node
void	PROP_PartitionInit (PartitionData* partitionData, NodeInput* nodeInput) Initialize some partition specific data structures. This function is called from each partition, not from each node This function is only called for non-MPI
void	PROP_Init (Node* node, int channelIndex, NodeInput* nodeInput) Initialization function for propagation functions. This function is called from each node.
void	PROP_ProcessEvent (Node* node, Message* msg) To receive message.
void	PROP_Finalize (Node* node) To collect various result.
double	PROP_PathlossFreeSpace (double distance, double wavelength) Calculates pathloss using free space model.

PROPAGATION

double	<code>PROP_PathlossTwoRay(double distance, double wavelength, float txAntennaHeight, float rxAntennaHeight)</code>
	To calculate path loss of a channel.
double	<code>PROP_PathlossOpar(double distance, double OverlappingDistance, double frequency, ObstructionType obstructiontype)</code>
	Calculates extra path attenuation using opar model.
void	<code>PROP_CalculatePathloss(Node* node, NodeId txNodeId, NodeId rxNodeId, int channelIndex, double wavelength, float txAntennaHeight, float rxAntennaHeight, PropPathProfile* pathProfile, bool forBinning)</code>
	To calculate path loss of a channel.
void	<code>PROP_CalculateFading(PropTxInfo* propTxInfo, Node* node2, int channelIndex, clocktype currentTime, float* fading_dB, double* channelReal, double* channelImag)</code>
	To calculate fading between two node.
BOOL	<code>PROP_CalculateRxPowerAndPropagationDelay(Message* msg, int channelIndex, PropChannel* propChannel, PropTxInfo* propTxInfo, Node* txNode, Node* rxNode, PropPathProfile* pathProfile)</code>
	This function will be called by QualNet wireless propagation code to calculate rxPower and prop delay for a specific signal from a specific tx node to a specific rx node.
BOOL	<code>PROP_CalculateRxPowerAndPropagationDelay(Message* msg, int channelIndex, PropChannel* propChannel, PropTxInfo* propTxInfo, Node* txNode, Node* rxNode, PropPathProfile* pathProfile)</code>
	This function will be called by QualNet wireless propagation code to calculate rxPower and prop delay for a specific signal from a specific tx node to a specific rx node.
void	<code>PROP_MotionObtainfadingStretchingFactor(PropTxInfo* propTxInfo, Node* receiver, int channelIndex)</code>
	Get a stretching factor for fast moving objects.
void	<code>PROP_UpdatePathProfiles(Node* node)</code>
	UpdatePathProfiles
void	<code>PROP_ReleaseSignal(Node* node, Message* msg, int phyIndex, int channelIndex, float txPower_dBm, clocktype duration, clocktype delayUntilAirborne)</code>
	Release (transmit) the signal
void	<code>PROP_SubscribeChannel(Node* node, int phyIndex, int channelIndex)</code>

PROPAGATION

	Start subscribing (listening to) a channel PROP_UnsubscribeChannel (Node* node, int phyIndex, int channelIndex)
void	Stop subscription of (listening to) a channel PROP_UnreferenceSignal (Node* node, Message* msg)
void	Unreference a signal (internal use) PROP_CalculateInterNodePathLossOnChannel (Node* node, int channelIndex, int* numNodesOnChannel, NodeAddress* nodeIdList, float** pathloss_dB, float** distance)
clocktype	Calculate inter-node pathloss, distance values between all the nodes on a given channel PROP_CalculatePropagationDelay (double distance, double propSpeed, PartitionData* partitionData, int channelIndex, int coordinateSystemType, Coordinates* fromPosition, Coordinates* toPosition)
void	Calculate the wireless propagation delay for the given distance and propagation speed. PROP_Reset (Node* node, int phyIndex, char* newChannelListenable)
void	Reset previous channel remove/add node to propChannel for signal delivery, in propagation_private. PROP_AddNodeToList (Node* node, int channelIndex)
void	add node to propChannel nodeList need to make sure that node is not already exists in list before adding. PROP_RemoveNodeFromList (Node* node, int channelIndex)
double	remove node from propChannel nodeList need to make sure that all the interface from that node is not listing on that channel before removing. PROP_GetChannelFrequency (Node* node, int channelIndex)
void	Get channel frequency from profile for PropChannel. PROP_SetChannelFrequency (Node* node, int channelIndex, double channelFrequency)
double	Set channel frequency from profile for PropChannel. PROP_GetChannelWavelength (Node* node, int channelIndex)
void	Get channel wavelength from profile for PropChannel. PROP_SetChannelWavelength (Node* node, int channelIndex, double channelWavelength)

	Set channel wavelength from profile for PropChannel. PROP_SetChannelDopplerFrequency (Node* node, int channelIndex)
double	Get channel doppler freq from profile for PropChannel. PROP_GetChannelDopplerFrequency (Node* node, int channelIndex, double channelDopplerFrequency)
void	Set channel doppler freq from profile for PropChannel. PROP_SetChannelDopplerFrequency (Node* node, int channelIndex, double channelDopplerFrequency)

Constant / Data Structure Detail

Constant	BOLTZMANN_CONSTANT 1.379e-23 Boltzmann constant
Constant	NEGATIVE_PATHLOSS_dB -1.0 Path loss in dB (used as an invalid value)
Constant	SPEED_OF_LIGHT 3.0e8 Defines the value of speed of light
Constant	PROP_DEFAULT_PROPAGATION_LIMIT_dBm -111.0 Default value for propagation limit.
Constant	PROP_DEFAULT_SHADOWING_MEAN_dB 4.0 Default mean value for shadowing in dB
Constant	MAX_NUM_ELEVATION_SAMPLES 16384

	Maximum number of sample would be taken.
Constant	PROP_DEFAULT_BANDWIDTH_FACTOR 2.0 The bandwidth factor that is used to get the half sum bandwidth.
Enumeration	PathlossModel Different type of path loss.
Enumeration	ShadowingModel Different type of shadowing used.
Enumeration	FadingModel Different type of fading used.
Enumeration	propagationEnvironment Different type of propagation environment.
Enumeration	LoSIndicator Indicated if the path is Line of sight OR non-Line of sight
Enumeration	SuburbanTerrainType Terrain types for Suburban-foliage model
Enumeration	IndoorLinkType Link types for Indoor model
Enumeration	LinkType Link types for model
Structure	PropPathProfile Structure that keeps track of all propertice of a path.

Structure	PropChannel structure of a channel.
Structure	PropProfile Main structure of propagation profile
Structure	PropData Main structure of propagation data.
Structure	PropTxInfo This structure is used for fields related to channel layer information that need to be sent with a message.
Structure	PropRxInfo This structure is used for fields related to channel layer information that need to be received with a message.

Function / Macro Detail

Function / Macro	Format
PROP_NumberChannels(node)	Get the number of channel.
PROP_ChannelWavelength(node, channelIndex)	Get wavelength of channel having index channelIndex
PROP_GlobalInit Initialization function for propagation This function is called from each partition, not from each node	void PROP_GlobalInit (PartitionData* partitionData, NodeInput* nodeInput) Parameters: <ul style="list-style-type: none">• partitionData - structure shared among nodes• nodeInput - structure containing contents of input file Returns: <ul style="list-style-type: none">• void - None
PROP_PartitionInit	void PROP_PartitionInit (PartitionData* partitionData, NodeInput* nodeInput)

	<p>Initialize some partition specific data structures. This function is called from each partition, not from each node. This function is only called for non-MPI.</p> <p>Parameters:</p> <ul style="list-style-type: none"> partitionData - structure shared among nodes nodeInput - structure containing contents of input file <p>Returns:</p> <ul style="list-style-type: none"> void - None
PROP_Init	<p>Initialization function for propagation functions. This function is called from each node.</p> <p>void PROP_Init (Node* node, int channelIndex, NodeInput* nodeInput)</p> <p>Parameters:</p> <ul style="list-style-type: none"> node - node being initialized. channelIndex - channel being initialized. nodeInput - structure containing contents of input file <p>Returns:</p> <ul style="list-style-type: none"> void - None
PROP_ProcessEvent	<p>To receive message.</p> <p>void PROP_ProcessEvent (Node* node, Message* msg)</p> <p>Parameters:</p> <ul style="list-style-type: none"> node - Node that is msg - message received by the layer <p>Returns:</p> <ul style="list-style-type: none"> void - None
PROP_Finalize	<p>To collect various result.</p> <p>void PROP_Finalize (Node* node)</p> <p>Parameters:</p> <ul style="list-style-type: none"> node - node for which results are to be collected <p>Returns:</p> <ul style="list-style-type: none"> void - None
PROP_PathlossFreeSpace	<p>Calculates pathloss using free space model.</p> <p>double PROP_PathlossFreeSpace (double distance, double wavelength)</p> <p>Parameters:</p> <ul style="list-style-type: none"> distance - distance (meters) between two nodes wavelength - wavelength used for propagation. <p>Returns:</p>

	<ul style="list-style-type: none"> • <code>double</code> - pathloss in db
PROP_PathlossTwoRay	<p>double PROP_PathlossTwoRay (double distance, double wavelength, float txAntennaHeight, float rxAntennaHeight)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>distance</code> - distance (meters) between two nodes • <code>wavelength</code> - wavelength used for propagation. • <code>txAntennaHeight</code> - transmitting antenna hight. • <code>rxAntennaHeight</code> - receiving antenna hight. <p>Returns:</p> <ul style="list-style-type: none"> • <code>double</code> - pathloss in db
PROP_PathlossOpar	<p>double PROP_PathlossOpar (double distance, double OverlappingDistance, double frequency, ObstructionType obstructiontype)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>distance</code> - distance (meters) between two nodes • <code>OverlappingDistance</code> - overlapping distance • <code>frequency</code> - frequency used for propagation. • <code>obstructiontype</code> - obstruction type <p>Returns:</p> <ul style="list-style-type: none"> • <code>double</code> - extra path attenuation in db
PROP_CalculatePathloss	<p>void PROP_CalculatePathloss (Node* node, NodeId txNodeId, NodeId rxNodeId, int channelIndex, double wavelength, float txAntennaHeight, float rxAntennaHeight, PropPathProfile* pathProfile, bool forBinning)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Node that is • <code>txNodeId</code> - including for debugging • <code>rxNodeId</code> - including for debugging • <code>channelIndex</code> - channel number. • <code>wavelength</code> - wavelength used for propagation. • <code>txAntennaHeight</code> - transmitting antenna hight. • <code>rxAntennaHeight</code> - receiving antenna hight.

	<ul style="list-style-type: none"> • <code>pathProfile</code> - characteristics of path. • <code>forBinning</code> - disables some features to support <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
PROP_CalculateFading	<p>To calculate fading between two node.</p> <p><code>void PROP_CalculateFading (PropTxInfo* propTxInfo, Node* node2, int channelIndex, clocktype currentTime, float* fading_dB, double* channelReal, double* channelImag)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>propTxInfo</code> - Information about the transmitter • <code>node2</code> - receiver • <code>channelIndex</code> - channel number • <code>currentTime</code> - current simulation time • <code>fading_dB</code> - calculated fading store here. • <code>channelReal</code> - for cooperative comm • <code>channelImag</code> - for cooperative comm <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
PROP_CalculateRxPowerAndPropagationDelay	<p>This function will be called by QualNet wireless propagation code to calculate rxPower and prop delay for a specific signal from a specific tx node to a specific rx node.</p> <p><code>BOOL PROP_CalculateRxPowerAndPropagationDelay (Message* msg, int channelIndex, PropChannel* propChannel, PropTxInfo* propTxInfo, Node* txNode, Node* rxNode, PropPathProfile* pathProfile)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>msg</code> - Signal to be propagated • <code>channelIndex</code> - Channel that the signal is propagated • <code>propChannel</code> - Info of the propagation channel • <code>propTxInfo</code> - Transmission parameers of the tx node • <code>txNode</code> - Point to the Tx node • <code>rxNode</code> - Point to the Rx node • <code>pathProfile</code> - For returning results <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - If FALSE, indicate the two nodes cannot comm TRUE means two nodes can communicate
PROP_CalculateRxPowerAndPropagationDelay	<code>BOOL PROP_CalculateRxPowerAndPropagationDelay (Message* msg, int channelIndex, PropChannel* propChannel,</code>

	<p>This function will be called by QualNet wireless propagation code to calculate rxPower and prop delay for a specific signal from a specific tx node to a specific rx node.</p> <p>PropTxInfo* propTxInfo, Node* txNode, Node* rxNode, PropPathProfile* pathProfile)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • msg - Signal to be propagated • channelIndex - Channel that the signal is propagated • propChannel - Info of the propagation channel • propTxInfo - Transmission parameters of the tx node • txNode - Point to the Tx node • rxNode - Point to the Rx node • pathProfile - For returning results <p>Returns:</p> <ul style="list-style-type: none"> • BOOL - If FALSE, indicate the two nodes cannot comm TRUE means two nodes can communicate
PROP_MotionObtainfadingStretchingFactor Get a stretching factor for fast moving objects.	void PROP_MotionObtainfadingStretchingFactor (PropTxInfo* propTxInfo, Node* receiver, int channelIndex) <p>Parameters:</p> <ul style="list-style-type: none"> • propTxInfo - Transmitter information • receiver - Receiver node. • channelIndex - channel number <p>Returns:</p> <ul style="list-style-type: none"> • void - None
PROP_UpdatePathProfiles UpdatePathProfiles	void PROP_UpdatePathProfiles (Node* node) <p>Parameters:</p> <ul style="list-style-type: none"> • node - Node that is <p>Returns:</p> <ul style="list-style-type: none"> • void - None
PROP_ReleaseSignal Release (transmit) the signal	void PROP_ReleaseSignal (Node* node, Message* msg, int phyIndex, int channelIndex, float txPower_dBm, clocktype duration, clocktype delayUntilAirborne) <p>Parameters:</p> <ul style="list-style-type: none"> • node - Node that is • msg - Signal to be transmitted

	<ul style="list-style-type: none"> • <code>phyIndex</code> - PHY data index • <code>channelIndex</code> - channel index • <code>txPower_dBm</code> - transmitting power • <code>duration</code> - transmission duration • <code>delayUntilAirborne</code> - delay until airborne <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
PROP_SubscribeChannel	<p>Start subscribing (listening to) a channel</p> <p><code>void PROP_SubscribeChannel (Node* node, int phyIndex, int channelIndex)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Node that is • <code>phyIndex</code> - interface index • <code>channelIndex</code> - channel index <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
PROP_UnsubscribeChannel	<p>Stop subscription of (listening to) a channel</p> <p><code>void PROP_UnsubscribeChannel (Node* node, int phyIndex, int channelIndex)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Node that is • <code>phyIndex</code> - interface index • <code>channelIndex</code> - channel index <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
PROP_UnreferenceSignal	<p>Unreference a signal (internal use)</p> <p><code>void PROP_UnreferenceSignal (Node* node, Message* msg)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Node that is • <code>msg</code> - Signal to be unreferenced <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
PROP_CalculateInterNodePathLossOnChannel	<code>void PROP_CalculateInterNodePathLossOnChannel (Node* node, int channelIndex, int* numNodesOnChannel,</code>

<p>Calculate inter-node pathloss, distance values between all the nodes on a given channel</p>	<p><code>NodeAddress* nodeIdList, float** pathloss_dB, float** distance)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - any valid node • <code>channelIndex</code> - selected channel instance • <code>numNodesOnChannel</code> - number of nodes using this channel • <code>nodeIdList</code> - list of (<code>numNodesOnChannel</code>) <code>nodeIds</code> • <code>pathloss_dB</code> - 2D pathloss array for nodes in • <code>distance</code> - 2D array of inter-node distances <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
<p>PROP_CalculatePropagationDelay</p> <p>Calculate the wireless propagation delay for the given distance and propagation speed.</p>	<p><code>clocktype PROP_CalculatePropagationDelay (double distance, double propSpeed, PartitionData* partitionData, int channelIndex, int coordinateSystemType, Coordinates* fromPosition, Coordinates* toPosition)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>distance</code> - Propagation distance • <code>propSpeed</code> - Propagation speed • <code>partitionData</code> - Partition data • <code>channelIndex</code> - Channel index or -1 for p2p • <code>coordinateSystemType</code> - Coordinate system type • <code>fromPosition</code> - Source position • <code>toPosition</code> - Destination position <p>Returns:</p> <ul style="list-style-type: none"> • <code>clocktype</code> - Calculated propagation delay COMMENTS :: + <code>partitionData</code> can be used to get the simulation time or terrain data + <code>channelIndex</code> indicates the channel for scenarios with multiple channels Wireless p2p link or microwave links don't use propagation channels. -1 will be passed in which indicate p2p/microwave links. + <code>fromPosition</code> and <code>toPosition</code> are not used right now. They can be used to calculate location specific delay.
<p>PROP_Reset</p> <p>Reset previous channel remove/add node to <code>propChannel</code> for signal delivery, in <code>propagation_private</code>.</p>	<p><code>void PROP_Reset (Node* node, int phyIndex, char* newChannelListenable)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Node that is being instantiated in • <code>phyIndex</code> - interface index • <code>newChannelListenable</code> - new channel

	<p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
PROP_AddNodeToList	<p>void PROP_AddNodeToList (Node* node, int channelIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - the node • <code>channelIndex</code> - channel index <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
PROP_RemoveNodeFromList	<p>remove node from propChannel nodeList need to make sure that all the interface from that node is not listing on that channel before removing.</p> <p>void PROP_RemoveNodeFromList (Node* node, int channelIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - the node • <code>channelIndex</code> - channel index <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
PROP_GetChannelFrequency	<p>Get channel frequency from profile for PropChannel.</p> <p>double PROP_GetChannelFrequency (Node* node, int channelIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - the node • <code>channelIndex</code> - channel index <p>Returns:</p> <ul style="list-style-type: none"> • <code>double</code> - channel frequency
PROP_SetChannelFrequency	<p>Set channel frequency from profile for PropChannel.</p> <p>void PROP_SetChannelFrequency (Node* node, int channelIndex, double channelFrequency)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - the node • <code>channelIndex</code> - channel index • <code>channelFrequency</code> - new channel frequency <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None

PROP_GetChannelWavelength	<p>double PROP_GetChannelWavelength (Node* node, int channelIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - the node • channelIndex - channel index <p>Returns:</p> <ul style="list-style-type: none"> • double - channel wavelength
PROP_SetChannelWavelength	<p>void PROP_SetChannelWavelength (Node* node, int channelIndex, double channelWavelength)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - the node • channelIndex - channel index • channelWavelength - new channel wavelength <p>Returns:</p> <ul style="list-style-type: none"> • void - None
PROP_GetChannelDopplerFrequency	<p>double PROP_GetChannelDopplerFrequency (Node* node, int channelIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - the node • channelIndex - channel index <p>Returns:</p> <ul style="list-style-type: none"> • double - channel doppler freq
PROP_SetChannelDopplerFrequency	<p>void PROP_SetChannelDopplerFrequency (Node* node, int channelIndex, double channelDopplerFrequency)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - the node • channelIndex - channel index • channelDopplerFrequency - new channel doppler freq <p>Returns:</p> <ul style="list-style-type: none"> • void - None
PROP_FrequencyOverlap	BOOL PROP_FrequencyOverlap (Node* txNode, Node* rxNode, int txChannelIndex, int rxChannelIndex, int txPhyIndex, int rxPhyIndex)

Check if there is frequency overlap between signal and receiver node.

Parameters:

- `txNode` - the Tx node
- `rxNode` - the Rx node
- `txChannelIndex` - the Tx channel index
- `rxChannelIndex` - the Rx channel index
- `txPhyIndex` - the PHY index for the Tx node.
- `rxPhyIndex` - the PHY index for the Rx node.

Returns:

- `BOOL` - if there is frequency overlap



Contact [QualNet Support](#) for questions pertaining to the QualNet API Reference. This document is confidential and proprietary.
It may not be reproduced or distributed without the expressed written consent of [Scalable Network Technologies](#).

QualNet® is a Registered Trademark of [Scalable Network Technologies](#).

Copyright © 2001-2012 [Scalable Network Technologies, Inc.](#). All rights reserved.

QualNet 6.1 API Reference

QUEUES

This file describes the member functions of the queue base class.

Constant / Data Structure Summary

Type	Name
CONSTANT	DEQUEUE_NEXT_PACKET
	Denotes position of packet in the queue for dequeue operation
CONSTANT	ALL_PRIORITIES
	This macro is used to specify that queue and scheduler operations not consider priority value of queue or packet
CONSTANT	QOS_DEFAULT_INTERFACE_OBSERVATION_INTERVAL
	This macro is used to specify the interface observation interval for Qos Routing. Ref.(Qospf.h see QOSPF_DEFAULT_INTERFACE_OBSERVATION_INTERVAL)
CONSTANT	STATISTICS_RESOLUTION
	This macro is used to support overflow issue to account for long delay network such as in space applications, or simply very very long simulations, divide delays by STATISTICS_RESOLUTION during runtime, and multiply by STATISTICS_RESOLUTION at the end of the simulation when IO_PrintStat'ing
CONSTANT	DEFAULT_QUEUE_DELAY_WEIGHT_FACTOR
	This macro is used to define the weight to assign to the most recent delay in calculating an exponential moving average. The value is fairly large because the queue delay is used for QoS routing decisions.
CONSTANT	PACKET_ARRAY_INFO_FIELD_SIZE
	The Queue structure will store a field of data in addition to the Message itself, with a maximum size of this value
ENUMERATION	QueueBehavior
	This enumeration is used by both queues and schedulers to determine the queue behavior.
ENUMERATION	QueueOperation

	This enumeration is used by both queues and schedulers to determine the operation of the retrieve functions.
STRUCT	PacketArrayEntry
	This structure represents an entry in the array of stored messages. The infoField (perhaps this should be renamed to prevent confusion) will store a queue algorithm dependent amount of data about each Message, as well as the simulation time that Message was inserted.

Function / Macro Summary

Return Type	Summary
void	<p>Queue(Message* msg, const void* infoField, BOOL* QueueIsFull, const clocktype currentTime, const double serviceTag)</p> <p>This function prototype determines the arguments that need to be passed to a queue data structure in order to insert a message into it. The infoField parameter has a specified size infoFieldSize, which is set at Initialization, and points to the structure that should be stored along with the Message.</p>
BOOL	<p>Queue(Message** msg, const int index, const QueueOperation operation, const clocktype currentTime, double* serviceTag)</p> <p>This function prototype determines the arguments that need to be passed to a queue data structure in order to dequeue, peek at, or drop a message in its array of stored messages. It now includes the "DropFunction" functionality as well.</p>
BOOL	<p>Queue()</p> <p>This function prototype returns a Boolean value of true if the array of stored messages is empty, false otherwise.</p>
int	<p>Queue()</p> <p>This function prototype returns the number of bytes stored in the array.</p>
int	<p>Queue()</p> <p>This function prototype returns free space in number of bytes in the queue.</p>
int	<p>Queue()</p>

	This function prototype returns the number of Messages stored in the packetArray.
int	Queue()
void	This function prototype returns the size of the Queue Queue(double serviceTag)
int	Set the service tag of the queue Queue(Queue* oldQueue)
void	This function is proposed to replicate the state of the queue, as if it had been the operative queue all along. If there are packets in the existing queue, they are transferred one-by-one into the new queue. This can result in additional drops of packets that had previously been stored. This function returns the number of additional drops. Queue(BOOL suspend)
void	This function is proposed to identify and tag misbehaved queue at the interface, so that they can be punished. Queue(int* qDelayVal, int* totalTransmissionVal, const clocktype currentTime, BOOL isResetTotalTransmissionVal)
int	This function is proposed for qos information update for Qos Routings like Qospf. Queue()
clocktype	This function prototype returns the number of bytes dequeued, not dropped, during a given period. This period starts at the beginning of the simulation, and restarts whenever the Queue resetPeriod function is called. Queue()
clocktype	This function prototype returns the queue utilization, or the amount of time that the queue is nonempty, during a given period. This period starts at the beginning of the simulation, and restarts whenever the queue resetPeriod function is called. Queue()
void	This function prototype returns the average time a packet spends in the queue, during a given period. This period starts at the beginning of the simulation, and restarts whenever the QueueResetPeriodFunctionType function is called. Queue(clocktype currentTime)
clocktype	This function prototype resets the current period statistics variables, and sets the currentPeriodStartTime to the currentTime. Queue()

	This function prototype returns the currentPeriodStartTime. void Queue (Node* node, const char* layer, const int interfaceIndex, const int instanceId, const char* invokingProtocol, const char* splStatStr)
void	This function prototype outputs the final statistics for this queue. The layer, protocol, interfaceAddress, and instanceId parameters are given to IO_PrintStat with each of the queue's statistics. Queue (Node* node, const char queueTypeString[], const int queueSize, const int interfaceIndex, const int queueNumber, const int infoFieldSize, const BOOL enableQueueStat, const BOOL showQueueInGui, const clocktype currentTime, const void* configInfo)
	This function runs queue initialization routine. Any algorithm specific configurable parameters will be kept in a structure and after feeding that structure the structure pointer will be sent to this function via that void pointer configInfo. Some parameters includes default values, to prevent breaking existing models. [Uses: vide Pseudo code]

Constant / Data Structure Detail

Constant	DEQUEUE_NEXT_PACKET 0 Denotes position of packet in the queue for dequeue operation
Constant	ALL_PRIORITIES -1 This macro is used to specify that queue and scheduler operations not consider priority value of queue or packet
Constant	QOS_DEFAULT_INTERFACE_OBSERVATION_INTERVAL 2 * SECOND This macro is used to specify the interface observation interval for Qos Routing. Ref.(Qospf.h see QOSPF_DEFAULT_INTERFACE_OBSERVATION_INTERVAL)
Constant	STATISTICS_RESOLUTION 1 * MICRO_SECOND This macro is used to support overflow issue to account for long delay network such as in space applications, or simply very very long simulations, divide delays by STATISTICS_RESOLUTION during runtime, and multiply by STATISTICS_RESOLUTION at the end of the simulation when IO_PrintStat'ing
Constant	DEFAULT_QUEUE_DELAY_WEIGHT_FACTOR 0.1 This macro is used to define the weight to assign to the most recent delay in calculating an exponential moving average. The value is fairly large because the queue delay is used for QoS routing decisions.
Constant	PACKET_ARRAY_INFO_FIELD_SIZE 32

	The Queue structure will store a field of data in addition to the Message itself, with a maximum size of this value
Enumeration	<p>QueueBehavior</p> <p>This enumeration is used by both queues and schedulers to determine the queue behavior.</p>
Enumeration	<p>QueueOperation</p> <p>This enumeration is used by both queues and schedulers to determine the operation of the retrieve functions.</p>
Structure	<p>PacketArrayEntry</p> <p>This structure represents an entry in the array of stored messages. The infoField (perhaps this should be renamed to prevent confusion) will store a queue algorithm dependent amount of data about each Message, as well as the simulation time that Message was inserted.</p>
Structure	<p>QueueAgeInfo</p> <p>This structure contains information for each packet inserted into the queue to uniquely identify it so that it can be removed from the queue due to age.</p>

Function / Macro Detail

Function / Macro	Format
Queue	<p>void Queue (Message* msg, const void* infoField, BOOL* QueueIsFull, const clocktype currentTime, const double serviceTag)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • msg - Pointer to Message structure • infoField - The infoField parameter • QueueIsFull - returns Queue occupancy status • currentTime - Current Simulation time • serviceTag - ServiceTag <p>Returns:</p> <ul style="list-style-type: none"> • void - Null
Queue	BOOL Queue (Message** msg, const int index, const QueueOperation operation, const clocktype currentTime, double* serviceTag)

<p>This function prototype determines the arguments that need to be passed to a queue data structure in order to dequeue, peek at, or drop a message in its array of stored messages. It now includes the "DropFunction" functionality as well.</p>	<p>Parameters:</p> <ul style="list-style-type: none"> • <code>msg</code> - The retrieved msg • <code>index</code> - The position of the packet in the queue • <code>operation</code> - The retrieval mode • <code>currentTime</code> - Current Simulation time • <code>serviceTag</code> - ServiceTag = NULL <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - TRUE or FALSE
<p>Queue</p> <p>This function prototype returns a Boolean value of true if the array of stored messages is empty, false otherwise.</p>	<p>BOOL Queue ()</p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> • <code>BOOL</code> - TRUE or FALSE
<p>Queue</p> <p>This function prototype returns the number of bytes stored in the array.</p>	<p>int Queue ()</p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> • <code>int</code> - Integer
<p>Queue</p> <p>This function prototype returns free space in number of bytes in the queue.</p>	<p>int Queue ()</p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> • <code>int</code> - number of bytes free.
<p>Queue</p> <p>This function prototype returns the number of Messages stored in the packetArray.</p>	<p>int Queue ()</p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> • <code>int</code> - Integer
<p>Queue</p> <p>This function prototype returns the size of the Queue</p>	<p>int Queue ()</p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> • <code>int</code> - Integer

Queue	<p>Set the service tag of the queue</p> <pre>void Queue (double serviceTag)</pre> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>serviceTag</code> - the value of the service tag <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - NULL
Queue	<p>This function is proposed to replicate the state of the queue, as if it had been the operative queue all along. If there are packets in the existing queue, they are transferred one-by-one into the new queue. This can result in additional drops of packets that had previously been stored. This function returns the number of additional drops.</p> <pre>int Queue (Queue* oldQueue)</pre> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>oldQueue</code> - Old queue pointer <p>Returns:</p> <ul style="list-style-type: none"> • <code>int</code> - Old packetArray
Queue	<p>This function is proposed to identify and tag misbehaved queue at the interface, so that they can be punished.</p> <pre>void Queue (BOOL suspend)</pre> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>suspend</code> - The queue status <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - Null
Queue	<p>This function is proposed for qos information update for Qos Routings like Qospf.</p> <pre>void Queue (int* qDelayVal, int* totalTransmissionVal, const clocktype currentTime, BOOL isResetTotalTransmissionVal)</pre> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>qDelayVal</code> - Returning qDelay value • <code>totalTransmissionVal</code> - Returning totalTransmission value • <code>currentTime</code> - Current simulation time • <code>isResetTotalTransmissionVal</code> - Default false <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - Null
Queue	<p>This function prototype returns the number of</p> <pre>int Queue ()</pre> <p>Parameters:</p> <p>Returns:</p>

<p>bytes dequeued, not dropped, during a given period. This period starts at the beginning of the simulation, and restarts whenever the Queue resetPeriod function is called.</p>	<ul style="list-style-type: none"> • <code>int</code> - Integer
<p>Queue</p> <p>This function prototype returns the queue utilization, or the amount of time that the queue is nonempty, during a given period. This period starts at the beginning of the simulation, and restarts whenever the queue resetPeriod function is called.</p>	<p><code>clocktype Queue ()</code></p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> • <code>clocktype</code> - Utilize Time.
<p>Queue</p> <p>This function prototype returns the average time a packet spends in the queue, during a given period. This period starts at the beginning of the simulation, and restarts whenever the QueueResetPeriodFunctionType function is called.</p>	<p><code>clocktype Queue ()</code></p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> • <code>clocktype</code> - Queue Delays.
<p>Queue</p> <p>This function prototype resets the current period statistics variables, and sets the currentPeriodStartTime to the currentTime.</p>	<p><code>void Queue (clocktype currentTime)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>currentTime</code> - Current simulation time. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - Null
<p>Queue</p> <p>This function prototype returns the currentPeriodStartTime.</p>	<p><code>clocktype Queue ()</code></p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> • <code>clocktype</code> - Current period start time.
<p>Queue</p> <p>This function prototype outputs the final statistics for this queue. The layer, protocol, interfaceAddress, and instanceId parameters are given to IO_PrintStat with each of the queue's statistics.</p>	<p><code>void Queue (Node* node, const char* layer, const int interfaceIndex, const int instanceId, const char* invokingProtocol, const char* splStatStr)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to Node structure • <code>layer</code> - The layer string • <code>interfaceIndex</code> - The interface index

	<ul style="list-style-type: none"> • <code>instanceId</code> - Instance Ids • <code>invokingProtocol</code> - The protocol string • <code>splStatStr</code> - Special string for stat print <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - Null
Queue	<pre>void Queue (Node* node, const char queueTypeString[], const int queueSize, const int interfaceIndex, const int queueNumber, const int infoFieldSize, const BOOL enableQueueStat, const BOOL showQueueInGui, const clocktype currentTime, const void* configInfo)</pre> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Node pointer • <code>queueTypeString[]</code> - Queue type string • <code>queueSize</code> - Queue size in bytes • <code>interfaceIndex</code> - used to set random seed • <code>queueNumber</code> - used to set random seed • <code>infoFieldSize</code> - Default infoFieldSize = 0 • <code>enableQueueStat</code> - Default enableQueueStat = false • <code>showQueueInGui</code> - If want to show this Queue in GUI • <code>currentTime</code> - Current simulation time • <code>configInfo</code> - pointer to a structure that contains <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - Null



Contact [QualNet Support](#) for questions pertaining to the QualNet API Reference. This document is confidential and proprietary.
It may not be reproduced or distributed without the expressed written consent of [Scalable Network Technologies](#).

QualNet® is a Registered Trademark of [Scalable Network Technologies](#).

QualNet 6.1 API Reference

RANDOM NUMBERS

This file describes functions to generate pseudo-random number streams.

Constant / Data Structure Summary

Type	Name
ENUMERATION	RandomDistributionType Random function types
ENUMERATION	RandomDataType Used for parsing input strings.
STRUCT	ValueProbabilityPair Stores one data point in a user defined distribution.
STRUCT	ArbitraryDistribution Stores a user defined distribution.

Function / Macro Summary

Return Type	Summary
void	RANDOM_SetSeed (RandomSeed seed, UInt32 globalSeed, UInt32 nodeId, UInt32 protocolId, UInt32 instanceId) Chooses from a set of pre-defined independent random seeds. The parameter names here are recommend invariants for use in selecting seeds, but other values could be used instead.
double	RANDOM_urand (RandomSeed seed) Returns a uniform distribution in [0.0 .. 1.0]
Int32	RANDOM_irand (RandomSeed seed)

RANDOM NUMBERS

	Returns an integer uniformly distributed between -2^31 and 2^31. RANDOM_nrand (RandomSeed seed)
Int32	Returns an integer uniformly distributed between 0 and 2^31. RANDOM_LoadUserDistributions (NodeInput* nodeInput)
void	Loads all user defined distributions. RandomDistribution.init()
void	Initializes the random distribution RandomDistribution.setDistributionUniform (T min, T max)
void	Sets the distribution to uniform. With this function, even integer types return values between [min, max), meaning to get a boolean distribution, use 0, 2. RandomDistribution.setDistributionUniformInteger (T min, T max)
void	This one gives [min, max] for integers. RandomDistribution.setDistributionExponential (T mean)
void	Sets the distribution to exponential with the given mean. RandomDistribution.setDistributionGaussian (T sigma)
void	Sets the distribution to Gaussian with the given sigma RandomDistribution.setDistributionGaussianInt (T sigma)
void	Sets the distribution to Gaussian with the given sigma RandomDistribution.setDistributionPareto (T val1, T val2, double alpha)
void	Sets the distribution to the truncated Pareto distribution RandomDistribution.setDistributionPareto4 (T val1, T val2, T val3, double alpha)
void	Sets the distribution to the truncated Pareto distribution RandomDistribution.setDistributionGeneralPareto (T val1, T val2, double alpha)

	Sets the distribution to general Pareto distribution <code>RandomDistribution.setDistributionParetoUntruncated(double alpha)</code>
void	Sets the distribution to the truncated Pareto distribution <code>RandomDistribution.setDistributionDeterministic(T val)</code>
void	The distribution will always return val. <code>RandomDistribution.setDistributionNull()</code>
int	The distribution will return 0. This is used for initialization. <code>RandomDistribution.setDistribution(char* inputString, char* printStr, RandomDataType dataType)</code>
T	Sets the distribution by parsing string input. <code>RandomDistribution.getRandomNumber(RandomSeed seed, Node* node)</code>
void	These two functions return the next random number from the defined distribution. <code>RandomDistribution.setSeed(UInt32 globalSeed, UInt32 nodeId, UInt32 protocolId, UInt32 instanceId)</code>
void	Calls RANDOM_SetSeed on the member seed. <code>RandomDistribution.setSeed(RandomSeed seed)</code>
	Copies the parameter seed into the member seed.

Constant / Data Structure Detail

Enumeration	RandomDistributionType Random function types
Enumeration	RandomDataType Used for parsing input strings.
Structure	ValueProbabilityPair

	Stores one data point in a user defined distribution.
Structure	<p>ArbitraryDistribution</p> <p>Stores a user defined distribution.</p>

Function / Macro Detail

Function / Macro	Format
RANDOM_SetSeed	<p>Chooses from a set of pre-defined independent random seeds. The parameter names here are recommended invariants for use in selecting seeds, but other values could be used instead.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>seed</code> - the seed to be set. • <code>globalSeed</code> - the scenario's global seed, i.e. SEED in the • <code>nodeId</code> - the node's ID • <code>protocolId</code> - the protocol number, as defined in the layer • <code>instanceId</code> - the instance of this protocol, often the <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
RANDOM_erand	<p>Returns a uniform distribution in [0.0 .. 1.0]</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>seed</code> - the seed for this random stream. <p>Returns:</p> <ul style="list-style-type: none"> • <code>double</code> - a random number
RANDOM_jrand	<p>Returns an integer uniformly distributed between -2^31 and 2^31.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>seed</code> - the seed for this random stream. <p>Returns:</p> <ul style="list-style-type: none"> • <code>Int32</code> - a random number

RANDOM_nrand	<p>Returns an integer uniformly distributed between 0 and 2^{31}.</p>	<p>Int32 RANDOM_nrand (RandomSeed seed)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>seed</code> - the seed for this random stream. <p>Returns:</p> <ul style="list-style-type: none"> • <code>Int32</code> - a random number
RANDOM_LoadUserDistributions	<p>Loads all user defined distributions.</p>	<p>void RANDOM_LoadUserDistributions (NodeInput* nodeInput)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>nodeInput</code> - the .config file <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
RandomDistribution.init	<p>Initializes the random distribution</p>	<p>void RandomDistribution.init ()</p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
RandomDistribution.setDistributionUniform	<p>Sets the distribution to uniform. With this function, even integer types return values between [min, max), meaning to get a boolean distribution, use 0, 2.</p>	<p>void RandomDistribution.setDistributionUniform (T min, T max)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>min</code> - the low end of the range • <code>max</code> - the high end of the range <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
RandomDistribution.setDistributionUniformInteger	<p>This one gives [min, max] for integers.</p>	<p>void RandomDistribution.setDistributionUniformInteger (T min, T max)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>min</code> - the low end of the range • <code>max</code> - the high end of the range <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
RandomDistribution.setDistributionExponential		<p>void RandomDistribution.setDistributionExponential (T mean)</p> <p>Parameters:</p>

	<p>Sets the distribution to exponential with the given mean.</p> <ul style="list-style-type: none"> • <code>mean</code> - the mean value of the distribution <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
RandomDistribution.setDistributionGaussian Sets the distribution to Gaussian with the given sigma	<p><code>void RandomDistribution.setDistributionGaussian (T sigma)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>sigma</code> - the sigma value <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
RandomDistribution.setDistributionGaussianInt Sets the distribution to Gaussian with the given sigma	<p><code>void RandomDistribution.setDistributionGaussianInt (T sigma)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>sigma</code> - the sigma value <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
RandomDistribution.setDistributionPareto Sets the distribution to the truncated Pareto distribution	<p><code>void RandomDistribution.setDistributionPareto (T val1, T val2, double alpha)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>val1</code> - the low end of the range • <code>val2</code> - the high end of the range • <code>alpha</code> - the alpha value <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
RandomDistribution.setDistributionPareto4 Sets the distribution to the truncated Pareto distribution	<p><code>void RandomDistribution.setDistributionPareto4 (T val1, T val2, T val3, double alpha)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>val1</code> - the minimum value of Pareto distribution • <code>val2</code> - the low end of the range • <code>val3</code> - the high end of the range • <code>alpha</code> - the alpha value <p>Returns:</p>

	<ul style="list-style-type: none"> • void - None
RandomDistribution.setDistributionGeneralPareto	<p>Sets the distribution to general Pareto distribution</p> <p>void RandomDistribution.setDistributionGeneralPareto (T val1, T val2, double alpha)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • val1 - the low end of the range • val2 - the high end of the range • alpha - the alpha value <p>Returns:</p> <ul style="list-style-type: none"> • void - None
RandomDistribution.setDistributionParetoUntruncated	<p>Sets the distribution to the truncated Pareto distribution</p> <p>void RandomDistribution.setDistributionParetoUntruncated (double alpha)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • alpha - the alpha value <p>Returns:</p> <ul style="list-style-type: none"> • void - None
RandomDistribution.setDistributionDeterministic	<p>The distribution will always return val.</p> <p>void RandomDistribution.setDistributionDeterministic (T val)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • val - the value to return <p>Returns:</p> <ul style="list-style-type: none"> • void - None
RandomDistribution.setDistributionNull	<p>The distribution will return 0. This is used for initialization.</p> <p>void RandomDistribution.setDistributionNull ()</p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> • void - None
RandomDistribution.setDistribution	<p>Sets the distribution by parsing string input.</p> <p>int RandomDistribution.setDistribution (char* inputString, char* printStr, RandomDataType dataType)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • inputString - the input string, typically from a line • printStr - usually the name of the calling • dataType - the data type of the template class is <p>Returns:</p>

	<ul style="list-style-type: none"> • <code>int</code> - returns the number of tokens read from the input string
RandomDistribution.getRandomNumber	<p>T RandomDistribution.getRandomNumber (RandomSeed seed, Node* node)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>seed</code> - when the seed parameter is present, it is used in • <code>node</code> - the node parameter is required to look up user <p>Returns:</p> <ul style="list-style-type: none"> • <code>T</code> - the random value
RandomDistribution.setSeed	<p>void RandomDistribution.setSeed (UInt32 globalSeed, UInt32 nodeId, UInt32 protocolId, UInt32 instanceId)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>globalSeed</code> - the scenario's global seed, i.e. SEED in the • <code>nodeId</code> - the node's ID • <code>protocolId</code> - the protocol number, as defined in the layer • <code>instanceId</code> - the instance of this protocol, often the <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
RandomDistribution.setSeed	<p>void RandomDistribution.setSeed (RandomSeed seed)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>seed</code> - an already initialized seed. <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None



Contact [QualNet Support](#) for questions pertaining to the QualNet API Reference. This document is confidential and proprietary.
It may not be reproduced or distributed without the expressed written consent of [Scalable Network Technologies](#).

QualNet® is a Registered Trademark of [Scalable Network Technologies](#).

QualNet 6.1 API Reference

SCHEDULERS

This file describes the member functions of the scheduler base class.

Constant / Data Structure Summary

Type	Name
CONSTANT	DEFAULT_QUEUE_COUNT Default number of queue per interface
STRUCT	QueueData This structure contains pointers to queue structures, default function behaviors, and statistics for the scheduler

Function / Macro Summary

Return Type	Summary
QueueData*	Scheduler (int priority) Returns pointer to QueueData associated with the queue. this is a Private
int	Scheduler () Returns number of queues under this Scheduler
int	Scheduler (int queueIndex) Returns Priority for the queues under this Scheduler
BOOL	Scheduler (const int priority) Returns a Boolean value of TRUE if the array of stored messages in each queue that the scheduler controls are empty, and FALSE otherwise
BOOL	Scheduler (const int priority)

	This function prototype returns the total number of bytes stored in the array of either a specific queue, or all queues that the scheduler controls.
int	<code>Scheduler</code> (const int priority)
	This function prototype returns the number of messages stored in the array of either a specific queue, or all queues that the scheduler controls.
void	<code>Scheduler</code> (int queueIndex, int* qDelayVal, int* totalTransmissionVal, const clocktype currentTime, BOOL isResetTotalTransmissionVal)
	This function enable Qos monitoring for all queues that the scheduler controls.
void	<code>Scheduler</code> (int priority, int packetSize, const clocktype currentTime)
	This function enable data collection for performance study of schedulers.
void	<code>Scheduler</code> (Node* node, const char* layer, const int interfaceIndex, const int instanceId, const char* invokingProtocol, const char* splStatStr)
	This function invokes queue finalization.
void	<code>SCHEDULER_Setup</code> (Scheduler** scheduler, const char schedulerTypeString[], BOOL enableSchedulerStat, const char* graphDataStr)
	This function runs the generic and then algorithm-specific scheduler initialization routine.
int	<code>GenericPacketClassifier</code> (Scheduler* scheduler, int pktPriority)
	Classify a packet for a specific queue

Constant / Data Structure Detail

Constant	DEFAULT_QUEUE_COUNT 3 Default number of queue per interface
Structure	QueueData This structure contains pointers to queue structures, default function behaviors, and statistics for the scheduler

Function / Macro Detail

Function / Macro	Format
Scheduler Returns pointer to QueueData associated with the queue. this is a Private	QueueData* Scheduler (int priority) Parameters: <ul style="list-style-type: none">• priority - Queue priority Returns: <ul style="list-style-type: none">• QueueData* - Pointer of queue
Scheduler Returns number of queues under this Scheduler	int Scheduler () Parameters: Returns: <ul style="list-style-type: none">• int - Number of queue.
Scheduler Returns Priority for the queues under this Scheduler	int Scheduler (int queueIndex) Parameters: <ul style="list-style-type: none">• queueIndex - Queue index Returns: <ul style="list-style-type: none">• int - Return priority of a queue
Scheduler Returns a Boolean value of TRUE if the array of stored messages in each queue that the scheduler controls are empty, and FALSE otherwise	BOOL Scheduler (const int priority) Parameters: <ul style="list-style-type: none">• priority - Priority of a queue Returns: <ul style="list-style-type: none">• BOOL - TRUE or FALSE
Scheduler This function prototype returns the total number of bytes stored in the array of either a specific queue, or all queues that the scheduler controls.	BOOL Scheduler (const int priority) Parameters: <ul style="list-style-type: none">• priority - Priority of a queue Returns: <ul style="list-style-type: none">• BOOL - TRUE or FALSE
Scheduler	int Scheduler (const int priority)

<p>This function prototype returns the number of messages stored in the array of either a specific queue, or all queues that the scheduler controls.</p>	<p>Parameters:</p> <ul style="list-style-type: none"> • <code>priority</code> - Priority of a queue <p>Returns:</p> <ul style="list-style-type: none"> • <code>int</code> - Bytes in queue is used.
<p>Scheduler</p> <p>This function enable Qos monitoring for all queues that the scheduler controls.</p>	<p><code>void Scheduler (int queueIndex, int* qDelayVal, int* totalTransmissionVal, const clocktype currentTime, BOOL isResetTotalTransmissionVal)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>queueIndex</code> - Queue index • <code>qDelayVal</code> - Queue delay • <code>totalTransmissionVal</code> - Transmission value • <code>currentTime</code> - Current simulation time • <code>isResetTotalTransmissionVal</code> - Total Transmission is set or not <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - Null
<p>Scheduler</p> <p>This function enable data collection for performance study of schedulers.</p>	<p><code>void Scheduler (int priority, int packetSize, const clocktype currentTime)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>priority</code> - Priority of the queue • <code>packetSize</code> - Size of packet • <code>currentTime</code> - Current simulation time <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - Null
<p>Scheduler</p> <p>This function invokes queue finalization.</p>	<p><code>void Scheduler (Node* node, const char* layer, const int interfaceIndex, const int instanceId, const char* invokingProtocol, const char* splStatStr)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>node</code> - Pointer to Node structure • <code>layer</code> - The layer string • <code>interfaceIndex</code> - Interface Index • <code>instanceId</code> - Instance Ids • <code>invokingProtocol</code> - The protocol string

	<ul style="list-style-type: none"> • <code>splStatStr</code> - Special string for stat print <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - Null
SCHEDULER_Setup	<p>This function runs the generic and then algorithm-specific scheduler initialization routine.</p> <p><code>void SCHEDULER_Setup (Scheduler** scheduler, const char schedulerTypeString[], BOOL enableSchedulerStat, const char* graphDataStr)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>scheduler</code> - Pointer of pointer to Scheduler class • <code>schedulerTypeString[]</code> - Scheduler Type string • <code>enableSchedulerStat</code> - Scheduler Statistics is set YES or NO • <code>graphDataStr</code> - Scheduler's graph statistics is set or not <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - Null
GenericPacketClassifier	<p>Classify a packet for a specific queue</p> <p><code>int GenericPacketClassifier (Scheduler* scheduler, int pktPriority)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>scheduler</code> - Pointer to a Scheduler class. • <code>pktPriority</code> - Incoming packet's priority <p>Returns:</p> <ul style="list-style-type: none"> • <code>int</code> - Integer.



Contact [QualNet Support](#) for questions pertaining to the QualNet API Reference. This document is confidential and proprietary.
It may not be reproduced or distributed without the expressed written consent of [Scalable Network Technologies](#).

[QualNet](#)® is a Registered Trademark of [Scalable Network Technologies](#).

Copyright © 2001-2012 [Scalable Network Technologies, Inc.](#). All rights reserved.

QualNet 6.1 API Reference

SLIDING-WINDOW

This file describes data structures and functions to implement a sliding window.

Constant / Data Structure Summary

Type	Name
STRUCT	MsTmWin sliding time window averager structure

Function / Macro Summary

Return Type	Summary
void	MsTmWinInit (MsTmWin* pWin, clocktype sSize, int nSlot, double weight, clocktype theTime) initialize time sliding window with the given parameters
void	MsTmWinInit (MsTmWin* pWin, clocktype sSize, int nSlot, double weight, clocktype theTime) resets time sliding window with the given parameters
void	MsTmWinNewData (MsTmWin* pWin, double data, clocktype theTime) updates time sliding window with the given new data
clocktype	MsTmWinWinSize (MsTmWin* pWin, clocktype theTime) returns the window size
double	MsTmWinSum (MsTmWin* pWin, clocktype theTime) computes the data sum of the window
double	MsTmWinAvg (MsTmWin* pWin, clocktype theTime)

	computes the data average of the window MsTmWinTotalSum (MsTmWin* pWin, clocktype theTime)
double	computes the total data sum MsTmWinTotalAvg (MsTmWin* pWin, clocktype theTime) computes the total data average

Constant / Data Structure Detail

Structure	MsTmWin
	sliding time window averager structure

Function / Macro Detail

Function / Macro	Format
MsTmWinInit initialize time sliding window with the given parameters	void MsTmWinInit (MsTmWin* pWin, clocktype sSize, int nSlot, double weight, clocktype theTime) Parameters: <ul style="list-style-type: none">• pWin - pointer to the time sliding window• sSize - sliding window slot size• nSlot - sliding window number of slots• weight - weight for average computation• theTime - the current time Returns: <ul style="list-style-type: none">• void - None
MsTmWinInit resets time sliding window with the given parameters	void MsTmWinInit (MsTmWin* pWin, clocktype sSize, int nSlot, double weight, clocktype theTime) Parameters: <ul style="list-style-type: none">• pWin - pointer to the time sliding window• sSize - sliding window slot size

	<ul style="list-style-type: none"> • <code>nslot</code> - sliding window number of slots • <code>weight</code> - weight for average computation • <code>theTime</code> - the current time <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
MsTmWinNewData	<p>updates time sliding window with the given new data</p> <p><code>void MsTmWinNewData (MsTmWin* pWin, double data, clocktype theTime)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>pWin</code> - pointer to the time sliding window • <code>data</code> - new data • <code>theTime</code> - the current time <p>Returns:</p> <ul style="list-style-type: none"> • <code>void</code> - None
MsTmWinWinSize	<p>returns the window size</p> <p><code>clocktype MsTmWinWinSize (MsTmWin* pWin, clocktype theTime)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>pWin</code> - pointer to the time sliding window • <code>theTime</code> - the current time <p>Returns:</p> <ul style="list-style-type: none"> • <code>clocktype</code> - the window size based on the current time
MsTmWinSum	<p>computes the data sum of the window</p> <p><code>double MsTmWinSum (MsTmWin* pWin, clocktype theTime)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>pWin</code> - pointer to the time sliding window • <code>theTime</code> - the current time <p>Returns:</p> <ul style="list-style-type: none"> • <code>double</code> - the data sum of the window
MsTmWinAvg	<p>computes the data average of the window</p> <p><code>double MsTmWinAvg (MsTmWin* pWin, clocktype theTime)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>pWin</code> - pointer to the time sliding window

	<ul style="list-style-type: none"> • <code>theTime</code> - the current time <p>Returns:</p> <ul style="list-style-type: none"> • <code>double</code> - the data average of the window
MsTmWinTotalSum computes the total data sum	<p><code>double MsTmWinTotalSum (MsTmWin* pWin, clocktype theTime)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>pWin</code> - pointer to the time sliding window • <code>theTime</code> - the current time <p>Returns:</p> <ul style="list-style-type: none"> • <code>double</code> - the total data sum
MsTmWinTotalAvg computes the total data average	<p><code>double MsTmWinTotalAvg (MsTmWin* pWin, clocktype theTime)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> • <code>pWin</code> - pointer to the time sliding window • <code>theTime</code> - the current time <p>Returns:</p> <ul style="list-style-type: none"> • <code>double</code> - the total data average



Contact [QualNet Support](#) for questions pertaining to the QualNet API Reference. This document is confidential and proprietary.
It may not be reproduced or distributed without the expressed written consent of [Scalable Network Technologies](#).

QualNet® is a Registered Trademark of [Scalable Network Technologies](#).

Copyright © 2001-2012 [Scalable Network Technologies, Inc.](#) All rights reserved.

QualNet 6.1 API Reference

TRACE

This file describes data structures and functions used for packet tracing.

Constant / Data Structure Summary

Type	Name
CONSTANT	MAX_TRACE_LENGTH
	Buffer for an XML trace record.
CONSTANT	TRACE_STRING_LENGTH
	Generic maximum length of a string. The maximum length of any line in the input file is 3x this value.
ENUMERATION	TraceDirectionType
	Different direction of packet tracing
ENUMERATION	PacketActionType
	Different types of action on packet
ENUMERATION	PacketDirection
	Direction of packet with respect to the node
ENUMERATION	TraceLayerType
	Keeps track of which layer is being traced.
ENUMERATION	TraceIncludedHeadersType
	Specifies if included headers are output.
ENUMERATION	PacketActionCommentType
	Gives specific comments on the packet action here packet drop.
ENUMERATION	TraceProtocolType

	Enlisting all the possible traces
STRUCT	TraceData Keeps track of which protocol is being traced.
STRUCT	PktQueue Gives details of the packet queue
STRUCT	ActionData Keeps track of protocol action

Function / Macro Summary

Return Type	Summary
void	TRACE_Initialize (Node* node, const NodeInput* nodeInput) Initialize necessary trace information before simulation starts.
BOOL	TRACE_IsTraceAll (Node* node) Determine if TRACE-ALL is enabled from configuration file.
void	TRACE_PrintTrace (Node* node, Message* message, TraceLayerType layerType, PacketDirection pktDirection, ActionData* actionData) Print trace information to file. To be used with Tracer.
void	TRACE_PrintTrace (Node* node, Message* message, TraceLayerType layerType, PacketDirection pktDirection, ActionData* actionData, NetworkType netType) Print trace information to file. To be used with Tracer.
void	TRACE_EnableTraceXML (Node* node, TraceProtocolType protocol, char* protocolName, TracePrintXMLFn xmlPrintFn, BOOL writeMap) Enable XML trace for a particular protocol.
void	TRACE_EnableTraceXML (Node* node, TraceProtocolType protocol, char* protocolName, TracePrintXMLFn xmlPrintFn, BOOL writeMap)

	Enable XML trace for a particular protocol. TRACE_DisableTraceXML (Node* node, TraceProtocolType protocol, char* protocolName, BOOL writeMap)
void	Disable XML trace for a particular protocol. TRACE_WriteToBufferXML (Node* node, char* buf)
void	Write trace information to a buffer, which will then be printed to a file. TRACE_WriteTraceHeader (FILE* fp)
void	Write trace header information to the partition's trace file TRACE_WriteXMLTraceTail (FILE* fp)
	Write trace tail information to the partition's trace file

Constant / Data Structure Detail

Constant	MAX_TRACE_LENGTH (4090) Buffer for an XML trace record.
Constant	TRACE_STRING_LENGTH 400 Generic maximum length of a string. The maximum length of any line in the input file is 3x this value.
Enumeration	TraceDirectionType Different direction of packet tracing
Enumeration	PacketActionType Different types of action on packet
Enumeration	PacketDirection

	Direction of packet with respect to the node
Enumeration	TraceLayerType Keeps track of which layer is being traced.
Enumeration	TraceIncludedHeadersType Specifies if included headers are output.
Enumeration	PacketActionCommentType Gives specific comments on the packet action here packet drop.
Enumeration	TraceProtocolType Enlisting all the possible traces
Structure	TraceData Keeps track of which protocol is being traced.
Structure	PktQueue Gives details of the packet queue
Structure	ActionData Keeps track of protocol action

Function / Macro Detail

Function / Macro	Format
TRACE_Initialize Initialize necessary trace information before simulation starts.	void TRACE_Initialize (Node* node, const NodeInput* nodeInput) Parameters: <ul style="list-style-type: none"> • node - this node • nodeInput - access to configuration file Returns:

	<ul style="list-style-type: none"> • void - NULL
TRACE_IsTraceAll	<p>Determine if TRACE-ALL is enabled from configuration file.</p> <p>BOOL TRACE_IsTraceAll (Node* node)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - this node <p>Returns:</p> <ul style="list-style-type: none"> • BOOL - TRUE if TRACE-ALL is enabled, FALSE otherwise.
TRACE_PrintTrace	<p>Print trace information to file. To be used with Tracer.</p> <p>void TRACE_PrintTrace (Node* node, Message* message, TraceLayerType layerType, PacketDirection pktDirection, ActionData* actionData)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - this node • message - Packet to print trace info from. • layerType - Layer that is calling this function. • pktDirection - If the packet is coming out of • actionData - more details about the packet action <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
TRACE_PrintTrace	<p>Print trace information to file. To be used with Tracer.</p> <p>void TRACE_PrintTrace (Node* node, Message* message, TraceLayerType layerType, PacketDirection pktDirection, ActionData* actionData, NetworkType netType)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - this node • message - Packet to print trace info from. • layerType - Layer that is calling this function. • pktDirection - If the packet is coming out of • actionData - more details about the packet action • netType - The network type. <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
TRACE_EnableTraceXML	<p>void TRACE_EnableTraceXML (Node* node, TraceProtocolType protocol, char* protocolName, TracePrintXMLFn xmlPrintFn, BOOL writeMap)</p>

<p>Enable XML trace for a particular protocol.</p>	<p>Parameters:</p> <ul style="list-style-type: none"> • node - this node • protocol - protocol to enable trace for • protocolName - name of protocol • xmlPrintFn - callback function • writeMap - flag to print protocol ID map <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
<p>TRACE_EnableTraceXML</p> <p>Enable XML trace for a particular protocol.</p>	<p>void <code>TRACE_EnableTraceXML</code> (Node* node, TraceProtocolType protocol, char* protocolName, TracePrintXMLFn xmlPrintFn, BOOL writeMap)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - this node • protocol - protocol to enable trace for • protocolName - name of protocol • xmlPrintFn - callback function • writeMap - flag to print protocol ID map <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
<p>TRACE_DisableTraceXML</p> <p>Disable XML trace for a particular protocol.</p>	<p>void <code>TRACE_DisableTraceXML</code> (Node* node, TraceProtocolType protocol, char* protocolName, BOOL writeMap)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • node - this node • protocol - protocol to enable trace for • protocolName - name of protocol • writeMap - flag to print protocol ID map <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
<p>TRACE_WriteToBufferXML</p>	<p>void <code>TRACE_WriteToBufferXML</code> (Node* node, char* buf)</p> <p>Parameters:</p>

<p>Write trace information to a buffer, which will then be printed to a file.</p>	<ul style="list-style-type: none"> • node - This node. • buf - Content to print to trace file. <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
<p>TRACE_WriteTraceHeader</p> <p>Write trace header information to the partition's trace file</p>	<p>void TRACE_WriteTraceHeader (FILE* fp)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • fp - pointer to the trace file. <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL
<p>TRACE_WriteXMLTraceTail</p> <p>Write trace tail information to the partition's trace file</p>	<p>void TRACE_WriteXMLTraceTail (FILE* fp)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • fp - pointer to the trace file. <p>Returns:</p> <ul style="list-style-type: none"> • void - NULL



Contact [QualNet Support](#) for questions pertaining to the QualNet API Reference. This document is confidential and proprietary. It may not be reproduced or distributed without the expressed written consent of [Scalable Network Technologies](#).

QualNet® is a Registered Trademark of [Scalable Network Technologies](#).

Copyright © 2001-2012 [Scalable Network Technologies, Inc.](#) All rights reserved.

QualNet 6.1 API Reference

TRANSPORT LAYER

This file describes data structures and functions used by the Transport Layer.

Constant / Data Structure Summary

Type	Name
CONSTANT	TRANSPORT_DELAY Delay to process a packet in transport layer
ENUMERATION	TransportProtocol Enlisting different transport layer protocol
STRUCT	TransportData Main data structure of transport layer

Constant / Data Structure Detail

Constant	TRANSPORT_DELAY (1 * MICRO_SECOND) Delay to process a packet in transport layer
Enumeration	TransportProtocol Enlisting different transport layer protocol
Structure	TransportData Main data structure of transport layer

Contact [QualNet Support](#) for questions pertaining to the QualNet API Reference. This document is confidential and proprietary.
It may not be reproduced or distributed without the expressed written consent of [Scalable Network Technologies](#).



[QualNet®](#) is a Registered Trademark of [Scalable Network Technologies](#).

Copyright © 2001-2012 [Scalable Network Technologies, Inc.](#). All rights reserved.

QualNet 6.1 API Reference

USER

This file describes data structures and functions used by the User Layer.

Constant / Data Structure Summary

Type	Name
CONSTANT	USER_PHONE_STARTUP_DELAY
	Delay from a cellphone is powered on until it can start working.
CONSTANT	USER_INCREASE_DISSATISFACTION
	The step value that the user dissatisfaction degree is increased each time.
CONSTANT	USER_CDECREASE_DISSATISFACTION
	The step value that the user dissatisfaction degree is decreased each time.
ENUMERATION	UserApplicationStatus
	Status of an user application session.
STRUCT	UserAppInfo
	Data structure stores information of one user application session.
STRUCT	UserStatus
	Data structure stores statuses of a user
STRUCT	struct_user_str
	Data structure stores information of a user

Function / Macro Summary

Return Type	Summary
void	<p>USER_HandleCallUpdate(Node* node, UserApplicationStatus appStatus)</p> <p>Reaction to the status change of an application session</p>
void	<p>USER_HandleUserLayerEvent(Node* node, Message* msg)</p> <p>Handle messages and events for user layer</p>
void	<p>USER_SetTrafficPattern(Node* node)</p> <p>Set a user's traffic pattern based on its profile.</p>
void	<p>USER_SetApplicationArrival(Node* node)</p> <p>Schedule an application arrival time.</p>

Constant / Data Structure Detail

Constant	<p>USER_PHONE_STARTUP_DELAY 5S</p> <p>Delay from a cellphone is powered on until it can start working.</p>
Constant	<p>USER_INCREASE_DISSATISFACTION 0.1</p> <p>The step value that the user dissatisfaction degree is increased each time.</p>
Constant	<p>USER_CECREASE_DISSATISFACTION -0.1</p> <p>The step value that the user dissatisfaction degree is decreased each time.</p>
Enumeration	<p>UserApplicationStatus</p> <p>Status of an user application session.</p>
Structure	<p>UserAppInfo</p> <p>Data structure stores information of one user application session.</p>
Structure	UserStatus

	Data structure stores statuses of a user
Structure	struct_user_str Data structure stores information of a user

Function / Macro Detail

Function / Macro	Format
USER_HandleCallUpdate Reaction to the status change of an application session	void USER_HandleCallUpdate (Node* node, UserApplicationStatus appStatus) Parameters: <ul style="list-style-type: none">• node - Pointer to node.• appStatus - New status of the app session Returns: <ul style="list-style-type: none">• void - NULL
USER_HandleUserLayerEvent Handle messages and events for user layer	void USER_HandleUserLayerEvent (Node* node, Message* msg) Parameters: <ul style="list-style-type: none">• node - Pointer to node.• msg - The event Returns: <ul style="list-style-type: none">• void - NULL
USER_SetTrafficPattern Set a user's traffic pattern based on its profile.	void USER_SetTrafficPattern (Node* node) Parameters: <ul style="list-style-type: none">• node - Pointer to node. Returns: <ul style="list-style-type: none">• void - NULL
USER_SetApplicationArrival Schedule an application arrival time.	void USER_SetApplicationArrival (Node* node) Parameters: <ul style="list-style-type: none">• node - Pointer to node.

Returns:

- void - NULL



Contact [QualNet Support](#) for questions pertaining to the QualNet API Reference. This document is confidential and proprietary.
It may not be reproduced or distributed without the expressed written consent of [Scalable Network Technologies](#).

QualNet® is a Registered Trademark of [Scalable Network Technologies](#).

Copyright © 2001-2012 [Scalable Network Technologies, Inc.](#). All rights reserved.

QualNet 6.1 API Reference

WALLCLOCK

This file describes methods of the WallClock class whose primary use is to keep track of the amount of real time that has passed during the simulation.

Function / Macro Summary

Return Type	Summary
BOOL	<p><code>WallClock(void)</code></p> <p>This method returns true if the WallClock is currently in the paused state.</p>
double	<p><code>wallclock()</code></p> <p>Return the real time multiple</p>
void	<p><code>WallClock(void)</code></p> <p>Pausing of the WallClock can be disabled by any external interface ambassador. Permission to pause is all or nothing, so if any external interface disables pause, no pausing is allowed. As an example, a simulation using IPNE and HLA is run. If the IPNE code disables pausing, then HLA won't be able to pause the WallClock or in other words the wall clock's value for time just keeps running.</p>
void	<p><code>WallClock(void)</code></p> <p>Allows pausing of the WallClock</p>
double	<p><code>WallClock(void)</code></p> <p>Get the amount of time, in seconds, spent paused.</p>

Function / Macro Detail

Function / Macro	Format
<code>WallClock</code>	<p>BOOL <code>WallClock(void)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <code>void</code>

	<p>This method returns true if the WallClock is currently in the paused state.</p>	<ul style="list-style-type: none"> - None <p>Returns:</p> <ul style="list-style-type: none"> • BOOL - TRUE or FALSE
WallClock		<p>double WallClock ()</p> <p>Parameters:</p>
Return the real time multiple		<p>Returns:</p> <ul style="list-style-type: none"> • double - None
WallClock	<p>Pausing of the WallClock can be disabled by any external interface ambassador. Permission to pause is all or nothing, so if any external interface disables pause, no pausing is allowed. As an example, a simulation using IPNE and HLA is run. If the IPNE code disables pausing, then HLA won't be able to pause the WallClock or in other words the wall clock's value for time just keeps running.</p>	<p>void WallClock (void)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • void - None <p>Returns:</p> <ul style="list-style-type: none"> • void - None
WallClock	Allows pausing of the WallClock	<p>void WallClock (void)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • void - None <p>Returns:</p> <ul style="list-style-type: none"> • void - None
WallClock	Get the amount of time, in seconds, spent paused.	<p>double WallClock (void)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • void - None <p>Returns:</p> <ul style="list-style-type: none"> • double - The amount of time paused, in seconds.



Contact [QualNet Support](#) for questions pertaining to the QualNet API Reference. This document is confidential and proprietary.
It may not be reproduced or distributed without the expressed written consent of [Scalable Network Technologies](#).

QualNet® is a Registered Trademark of [Scalable Network Technologies](#).

Copyright © 2001-2012 [Scalable Network Technologies, Inc.](#) All rights reserved.