



QualNet 7.1 API Reference Guide

August 2013

SCALABLE Network Technologies, Inc.

600 Corporate Pointe, Suite 1200
Culver City, CA 90230

+1.310.338.3318 TEL
+1.310.338.7213 FAX



SCALABLE-NETWORKS.COM

Copyright Information

© 2013 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.

600 Corporate Pointe, Suit 1200

Culver City, CA 90230

+1.310.338.3318 TEL

+1.310.338.7213 FAX

SCALABLE-NETWORKS.COM

QualNet 7.1 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 strucutres 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 Tansport 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 7.1 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
double	MATH_PlaneDistance (Vector3 vector, Vector3 point) 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)
BOOL	MATH_IntersectedPlane (Vector3[] polygon, Vector3[] line, Vector3& normal, double& originDistance) 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	MATH_ConvertXYToLatLong (double x1, double y1, double latitude, double longitude) Converts given cartesian coordinates to Latitide and Longitude

Constant / Data Structure Detail

Structure	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.
Structure	Triangle3 This struture will hold information for one triangle.

Function / Macro Detail

Function / Macro	Format
MATH_CrossProduct Returns a perpendicular vector from 2 given vectors by taking the cross product.	Vector3 MATH_CrossProduct (Vector3 vector1, Vector3 vector2) Parameters: <ul style="list-style-type: none">vector1 - the first vectorvector2 - the second vector Returns: <ul style="list-style-type: none">Vector3 - the cross product
MATH_Vector Returns a vector between 2 points	Vector3 MATH_Vector (Vector3 point1, Vector3 point2) Parameters: <ul style="list-style-type: none">point1 - the first pointpoint2 - the second point Returns: <ul style="list-style-type: none">Vector3 - a vector between the two points

MATH_Magnitude Returns the magnitude of a normal (or any other vector)	double MATH_Magnitude (Vector3 vector) Parameters: <ul style="list-style-type: none">vector - a vector Returns: <ul style="list-style-type: none">double - the magnitude of the vector
MATH_Normalize Returns a normalized vector (of exactly length 1)	Vector3 MATH_Normalize (Vector3 vector) Parameters: <ul style="list-style-type: none">vector - a vector Returns: <ul style="list-style-type: none">Vector3 - a normalized vector
MATH_Normal Returns the direction the polygon is facing	Vector3 MATH_Normal (Vector3[] triangle) Parameters: <ul style="list-style-type: none">triangle - an array of vectors representing a polygon Returns: <ul style="list-style-type: none">Vector3 - the direction vector
MATH_PlaneDistance 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)	double MATH_PlaneDistance (Vector3 vector, Vector3 point) Parameters: <ul style="list-style-type: none">vector - a vectorpoint - a point Returns: <ul style="list-style-type: none">double - the plane's distance from the origin (0,0,0)
MATH_IntersectedPlane Takes a triangle (plane) and line and returns true if they intersected	BOOL MATH_IntersectedPlane (Vector3[] polygon, Vector3[] line, Vector3& normal, double& originDistance) Parameters: <ul style="list-style-type: none">polygon - a polygonline - a linenormal - a normalized vectororiginDistance - the distance

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

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

Returns the whether the given point lies on Line or not.	<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. Returns: <ul style="list-style-type: none">• <code>BOOL</code> - TRUE if the point lies on line
MATH_ConvertXYToLatLong Converts given cartesian coordinates to Latitude and Longitude	<code>void MATH_ConvertXYToLatLong (double x1, double y1, double latitude, double longitude)</code> Parameters: <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 Returns: <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 © 2001-2013 [SCALABLE Network Technologies, Inc.](#) All rights reserved.



QualNet 7.1 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	ANTENNA_Init (Node* node, int phyIndex, const NodeInput* nodeInput) Initialize antennas.
void	ANTENNA_ReadPatterns (Node* node, int phyIndex, const NodeInput* antennaInput, int* numPatterns, int* steerablePatternSetRepeatSectorAngle, float*** pattern_dB, BOOL azimuthPlane) Read in the azimuth pattern file.

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

ANTENNA

float	ANTENNA_GainForThisDirection (Node* node, int phyIndex, Orientation DOA) Return gain for this direction in dB.
float	ANTENNA_GainForThisDirectionWithPatternIndex (Node* node, int phyIndex, int patternIndex, Orientation DOA) Return gain for this direction for the specified pattern in dB.
float	ANTENNA_GainForThisSignal (Node* node, int phyIndex, PropRxInfo* propRxInfo) Return gain in dB.
float	ANTENNA_DefaultGainForThisSignal (Node* node, int phyIndex, PropRxInfo* propRxInfo) Return default gain in dB.
void	ANTENNA_LockAntennaDirection (Node* node, int phyIndex) Lock antenna to current direction.
void	ANTENNA_UnlockAntennaDirection (Node* node, int phyIndex) Unlock antenna.
BOOL	ANTENNA_DirectionIsLocked (Node* node, int phyIndex) Return if direction antenna is locked.
BOOL	ANTENNA_IsLocked (Node* node, int phyIndex) Return if antenna is locked.
void	ANTENNA_SetToDefaultMode (Node* node, int phyIndex) Set default antenna mode (usally omni).
void	ANTENNA_SetToBestGainConfigurationForThisSignal (Node* node, int phyIndex, PropRxInfo* propRxInfo) Set antenna for best gain using the Rx info.
void	ANTENNA_SetBestConfigurationForAzimuth (Node* node, int phyIndex, double azimuth) Set antenna for best gain using the azimuth.
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	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 benodeInput - 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 beantennaInput - structure containing contents ofnumPatterns - contains the number of patternssteerablePatternSetRepeatSectorAngle - containspattern_dB - array used to store the gain valuesazimuthPlane - shows whether the file is azimuth Returns: <ul style="list-style-type: none">void - NULL
ANTENNA_ReadNsmaPatterns	void ANTENNA_ReadNsmaPatterns (Node* node, int phyIndex)

<p>Read in the NSMA pattern file.</p>	<p>Parameters:</p> <ul style="list-style-type: none"><code>node</code> - node being used.<code>phyIndex</code> - interface for which physical <p>Returns:</p> <ul style="list-style-type: none"><code>void</code> - NULL
<p>ANTENNA_ReadRevisedNsmPatterns</p> <p>Read in the Revised NSMA pattern file.</p>	<p>void ANTENNA_ReadRevisedNsmPatterns (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 <p>Returns:</p> <ul style="list-style-type: none"><code>void</code> - NULL
<p>ANTENNA_Read3DAsciiPatterns</p> <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"><code>node</code> - node being used.<code>phyIndex</code> - interface for which physical <p>Returns:</p> <ul style="list-style-type: none"><code>void</code> - NULL
<p>ANTENNA_Read2DAsciiPatterns</p> <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"><code>node</code> - node being used.<code>phyIndex</code> - interface for which physical <p>Returns:</p> <ul style="list-style-type: none"><code>void</code> - NULL
<p>ANTENNA_OmniDirectionalInit</p> <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"><code>node</code> - node being initialized.<code>nodeInput</code> - pointer to node input

	<ul style="list-style-type: none">• <code>phyIndex</code> - interface for which physical to be• <code>antennaModel</code> - pointer to <code>AntennaModelGlobal</code> <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - <code>NULL</code>
<p>ANTENNA_OmniDirectionalInitFromConfigFile</p> <p>Initialize omnidirectional antenna from the default.config file.</p>	<p><code>void ANTENNA_OmniDirectionalInitFromConfigFile (Node* node, int phyIndex, const NodeInput* nodeInput)</code></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> - <code>NULL</code>
<p>ANTENNA_InitFromConfigFile</p> <p>Initialize antenna from the default.config file.</p>	<p><code>void ANTENNA_InitFromConfigFile (Node* node, int phyIndex, const NodeInput* nodeInput)</code></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> - <code>NULL</code>
<p>ANTENNA_IsInOmnidirectionalMode</p> <p>Is antenna in omnidirectional mode.</p>	<p><code>BOOL ANTENNA_IsInOmnidirectionalMode (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 use <p>Returns:</p> <ul style="list-style-type: none">• <code>BOOL</code> - returns <code>TRUE</code> if antenna is in omnidirectional mode
<p>ANTENNA_ReturnPatternIndex</p> <p>Return nodes current pattern index.</p>	<p><code>int ANTENNA_ReturnPatternIndex (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> - interface for which physical to use Returns: <ul style="list-style-type: none">• <code>int</code> - returns pattern index
ANTENNA_ReturnHeight Return nodes antenna height.	float ANTENNA_ReturnHeight (Node* node, int phyIndex) Parameters: <ul style="list-style-type: none">• <code>node</code> - node being used• <code>phyIndex</code> - interface for which physical to be used Returns: <ul style="list-style-type: none">• <code>float</code> - height in meters
ANTENNA_ReturnSystemLossIndB Return system loss in dB.	double ANTENNA_ReturnSystemLossIndB (Node* node, int phyIndex) Parameters: <ul style="list-style-type: none">• <code>node</code> - node being used• <code>phyIndex</code> - interface for which physical to be used Returns: <ul style="list-style-type: none">• <code>double</code> - loss in dB
ANTENNA_GainForThisDirection Return gain for this direction in dB.	float ANTENNA_GainForThisDirection (Node* node, int phyIndex, Orientation DOA) Parameters: <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 Returns: <ul style="list-style-type: none">• <code>float</code> - gain in dB
ANTENNA_GainForThisDirectionWithPatternIndex Return gain for this direction for the specified pattern in dB.	float ANTENNA_GainForThisDirectionWithPatternIndex (Node* node, int phyIndex, int patternIndex, Orientation DOA) Parameters: <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 Returns: <ul style="list-style-type: none">float - gain in dB
ANTENNA_GainForThisSignal Return gain in dB.	float ANTENNA_GainForThisSignal (Node* node, int phyIndex, PropRxInfo* propRxInfo) Parameters: <ul style="list-style-type: none">node - node being usedphyIndex - interface for which physical to be usedpropRxInfo - receiver propagation info Returns: <ul style="list-style-type: none">float - gain in dB
ANTENNA_DefaultGainForThisSignal Return default gain in dB.	float ANTENNA_DefaultGainForThisSignal (Node* node, int phyIndex, PropRxInfo* propRxInfo) Parameters: <ul style="list-style-type: none">node - node being usedphyIndex - interface for which physical to be usedpropRxInfo - receiver propagation info Returns: <ul style="list-style-type: none">float - gain in dB
ANTENNA_LockAntennaDirection Lock antenna to current direction.	void ANTENNA_LockAntennaDirection (Node* node, int phyIndex) Parameters: <ul style="list-style-type: none">node - node being usedphyIndex - interface for which physical to be used Returns: <ul style="list-style-type: none">void - NULL
ANTENNA_UnlockAntennaDirection Unlock antenna.	void ANTENNA_UnlockAntennaDirection (Node* node, int phyIndex) Parameters: <ul style="list-style-type: none">node - node being usedphyIndex - interface for which physical to be used Returns:

	<ul style="list-style-type: none">void - NULL
ANTENNA_DirectionIsLocked Return if direction antenna is locked.	BOOL ANTENNA_DirectionIsLocked (Node* node, int phyIndex) Parameters: <ul style="list-style-type: none">node - node being usedphyIndex - interface for which physical to be used Returns: <ul style="list-style-type: none">BOOL - returns TRUE if the antenna direction is locked
ANTENNA_IsLocked Return if antenna is locked.	BOOL ANTENNA_IsLocked (Node* node, int phyIndex) Parameters: <ul style="list-style-type: none">node - node being usedphyIndex - interface for which physical to be used Returns: <ul style="list-style-type: none">BOOL - Returns TRUE if antenna is locked.
ANTENNA_SetToDefaultMode Set default antenna mode (usally omni).	void ANTENNA_SetToDefaultMode (Node* node, int phyIndex) Parameters: <ul style="list-style-type: none">node - node being usedphyIndex - interface for which physical to be used Returns: <ul style="list-style-type: none">void - NULL
ANTENNA_SetToBestGainConfigurationForThisSignal Set antenna for best gain using the Rx info.	void ANTENNA_SetToBestGainConfigurationForThisSignal (Node* node, int phyIndex, PropRxInfo* propRxInfo) Parameters: <ul style="list-style-type: none">node - node being usedphyIndex - interface for which physical to be usedpropRxInfo - receiver propagation info Returns: <ul style="list-style-type: none">void - NULL
ANTENNA_SetBestConfigurationForAzimuth	void ANTENNA_SetBestConfigurationForAzimuth (Node* node, int phyIndex, double azimuth)

<p>Set antenna for best gain using the azimuth.</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>azimuth</code> - the azimuth <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - NULL
<p>ANTENNA_GetSteeringAngle</p> <p>Get steering angle of the antenna.</p>	<p><code>void ANTENNA_GetSteeringAngle (Node* node, int phyIndex, Orientation* angle)</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>angle</code> - For returning the angle <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - NULL
<p>ANTENNA_SetSteeringAngle</p> <p>Set the steering angle of the antenna</p>	<p><code>void ANTENNA_SetSteeringAngle (Node* node, int phyIndex, Orientation angle)</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>angle</code> - Steering angle to be <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - NULL
<p>ANTENNA_ReturnAsciiPatternFile</p> <p>Read in the ASCII pattern .</p>	<p><code>void ANTENNA_ReturnAsciiPatternFile (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• <code>antennaModelInput</code> - structure containing <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - NULL

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



QualNet 7.1 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	struct antenna Global_model
	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_GlobalAntennaPatternInitFromConfigFile (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)

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

Constant / Data Structure Detail

Constant	MAX_ANTENNA_MODELS 50
	Maximum number of models to allow.
Constant	MAX_ANTENNA_PATTERNS 50
	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.

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 beantennaModelInput - structure containing Returns:

	<ul style="list-style-type: none">• <code>AntennaPattern*</code> - Pointer to the global antenna pattern structure.
ANTENNA_GlobalAntennaModelInit Reads the antenna configuration parameters into the global antenna model structure.	<code>void ANTENNA_GlobalAntennaModelInit</code> (<code>Node*</code> node, <code>int</code> phyIndex, <code>const NodeInput*</code> antennaModelInput) Parameters: <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 Returns: <ul style="list-style-type: none">• <code>void</code> - NULL
ANTENNA_GlobalAntennaPatternInitFromConfigFile Init the antenna pattern structure for pattern name for the Old antenna model.	<code>Void ANTENNA_GlobalAntennaPatternInitFromConfigFile</code> (<code>Node*</code> node, <code>int</code> phyIndex, <code>const char*</code> antennaPatternName, <code>BOOL</code> steer) Parameters: <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 differntiate which Returns: <ul style="list-style-type: none">• <code>Void</code> - NULL
ANTENNA_GlobalAntennaPatternInit Init the antenna pattern structure for pattern name.	<code>Void ANTENNA_GlobalAntennaPatternInit</code> (<code>Node*</code> node, <code>int</code> phyIndex, <code>const NodeInput*</code> antennaModelInput, <code>const char*</code> antennaPatternName) Parameters: <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 Returns: <ul style="list-style-type: none">• <code>Void</code> - NULL
ANTENNA_GlobalAntennaModelAlloc	<code>AntennaModelGlobal*</code> ANTENNA_GlobalAntennaModelAlloc (<code>PartitionData*</code> partitionData) Parameters:

Alloc a new model.	<ul style="list-style-type: none">partitionData - Pointer to partition data. <p>Returns:</p> <ul style="list-style-type: none">AntennaModelGlobal* - Pointer to the global antenna model structure.
ANTENNA_GlobalAntennaModelGet Return the model based on the name.	AntennaModelGlobal* ANTENNA_GlobalAntennaModelGet (PartitionData* partitionData, const char* antennaModelName) Parameters: <ul style="list-style-type: none">partitionData - Pointer to partition data.antennaModelName - contains the name of the <p>Returns:</p> <ul style="list-style-type: none">AntennaModelGlobal* - Pointer to the global antenna model structure.
ANTENNA_GlobalAntennaPatternGet Return the antenna pattern based on the name.	AntennaPattern* ANTENNA_GlobalAntennaPatternGet (PartitionData* partitionData, const char* antennaPatternName) Parameters: <ul style="list-style-type: none">partitionData - Pointer to partition data.antennaPatternName - contains the name of the <p>Returns:</p> <ul style="list-style-type: none">AntennaPattern* - Pointer to the global antenna pattern structure.
ANTENNA_GeneratePatterName Generate the Pattern name base on Pattern type.	void ANTENNA_GeneratePatterName (Node* node, int phyIndex, const NodeInput* antennaModelInput, char* antennaPatternName) Parameters: <ul style="list-style-type: none">node - node being used.phyIndex - interface for which physical to beantennaModelInput - structure containingantennaPatternName - antenna pattern name to be <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 © 2001-2013 [SCALABLE Network Technologies, Inc.](#) All rights reserved.

QualNet 7.1 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
STRUCT	AppToTcpClose
	Application wants to release the connection
STRUCT	AppToTcpConnSetup
	Application sets up connection at the local end Needed for NS TCP to fake connection setup
STRUCT	AppQosToNetworkSend
	Application uses this structure in its info field to perform the initialization of a new QoS connection with its QoS requirements.
STRUCT	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	USER_Finalize (Node * node)
	To collect results of simulation at the end for user layers
void	ATMLAYER2_Finalize (Node * node)
	To collect results at the end of the simulation.
void	ADAPTATION_Finalize (Node * node)
	To collect results of simulation at the end for network layers
void	CHANNEL_ProcessEvent (Node* node, Message* msg)
	Processes the message/event of physical layer received by the node thus simulating the PHYSICAL layer behaviour
void	PHY_ProcessEvent (Node* node, Message* msg)
	Processes the message/event of physical layer received by the node thus simulating the PHYSICAL layer behaviour
void	MAC_ProcessEvent (Node* node, Message* msg)
	Processes the message/event of MAC layer received by the node thus simulating the MAC layer behaviour
void	NETWORK_ProcessEvent (Node* node, Message* msg)
	Processes the message/event received by the node thus simulating the NETWORK layer behaviour
void	TRANSPORT_ProcessEvent (Node* node, Message* msg)
	Processes the message/event received by the node thus simulating the TRANSPORT layer behaviour
void	APP_ProcessEvent (Node* node, Message* msg)
	Processes the message/event received by the node thus simulating the APPLICATION layer behaviour
void	USER_ProcessEvent (Node* node, Message* msg)
	Processes the message/event received by the node thus simulating the USER layer behaviour
void	ATMLAYER2_ProcessEvent (Node* node, Message* msg)

	Processes the message/event of ATM_LAYER2 layer received by the node thus simulating the ATM_LAYER2 layer behaviour
void	ADAPTATION_ProcessEvent (Node* node, Message* msg)
	Processes the message/event received by the node thus simulating the ADAPTATION layer behaviour
void	MAC_RunTimeStat (Node* node)
	To print runtime statistics for the MAC layer
void	NETWORK_RunTimeStat (Node* node)
	To print runtime statistics for the NETWORK layer
void	TRANSPORT_RunTimeStat (Node* node)
	To print runtime statistics for the TRANSPORT layer
void	APP_RunTimeStat (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	PacketNetworkToApp Network to application layer packet structure
Structure	NetworkToTransportInfo Network To Transport layer Information structure
Structure	PacketTransportNetwork Transport to network layer packet structure
Structure	TcpTimerPacket TCP timer packet
Structure	AppToUdpSend Additional information given to UDP from applications. This information is saved in the info field of a message.
Structure	ZigbeeAppInfo Structure used for zigbee GTS implementation
Structure	UdpToAppRecv Additional information given to applications from UDP. This information is saved in the info field of a message.
Structure	AppToRsvpSend send response structure from application layer
Structure	TransportToAppListenResult Report the result of application's listen request.
Structure	TransportToAppOpenResult Report the result of opening a connection.
Structure	TransportToAppDataSent

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

Function / Macro Detail

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

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

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

NETWORK_Finalize To collect results of simulation at the end for network layers	void NETWORK_Finalize (Node * node) Parameters: <ul style="list-style-type: none">node - Node for which finalization function is called Returns: <ul style="list-style-type: none">void - None
TRANSPORT_Finalize To collect results of simulation at the end for transport layers	void TRANSPORT_Finalize (Node * node) Parameters: <ul style="list-style-type: none">node - Node for which finalization function is called Returns: <ul style="list-style-type: none">void - None
APP_Finalize To collect results of simulation at the end for application layers	void APP_Finalize (Node * node) Parameters: <ul style="list-style-type: none">node - Node for which finalization function is called Returns: <ul style="list-style-type: none">void - None
USER_Finalize To collect results of simulation at the end for user layers	void USER_Finalize (Node * node) Parameters: <ul style="list-style-type: none">node - Node for which finalization function is called Returns: <ul style="list-style-type: none">void - None
ATMLAYER2_Finalize To collect results at the end of the simulation.	void ATMLAYER2_Finalize (Node * node) Parameters: <ul style="list-style-type: none">node - Node for which finalization function is called Returns: <ul style="list-style-type: none">void - None
ADAPTATION_Finalize	void ADAPTATION_Finalize (Node * node) Parameters:

<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>void CHANNEL_ProcessEvent (Node* node, Message* msg)</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>void PHY_ProcessEvent (Node* node, Message* msg)</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>void MAC_ProcessEvent (Node* node, Message* msg)</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>void NETWORK_ProcessEvent (Node* node, Message* msg)</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

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

	<p>Returns:</p> <ul style="list-style-type: none"><code>void</code> - None
<p>MAC_RunTimeStat</p> <p>To print runtime statistics for the MAC layer</p>	<p>void MAC_RunTimeStat (Node* node)</p> <p>Parameters:</p> <ul style="list-style-type: none"><code>node</code> - node for which statistics to be printed <p>Returns:</p> <ul style="list-style-type: none"><code>void</code> - None
<p>NETWORK_RunTimeStat</p> <p>To print runtime statistics for the NETWORK layer</p>	<p>void NETWORK_RunTimeStat (Node* node)</p> <p>Parameters:</p> <ul style="list-style-type: none"><code>node</code> - node for which statistics to be printed <p>Returns:</p> <ul style="list-style-type: none"><code>void</code> - None
<p>TRANSPORT_RunTimeStat</p> <p>To print runtime statistics for the TRANSPORT layer</p>	<p>void TRANSPORT_RunTimeStat (Node* node)</p> <p>Parameters:</p> <ul style="list-style-type: none"><code>node</code> - node for which statistics to be printed <p>Returns:</p> <ul style="list-style-type: none"><code>void</code> - None
<p>APP_RunTimeStat</p> <p>To print runtime statistics for the APPLICATION layer</p>	<p>void APP_RunTimeStat (Node* node)</p> <p>Parameters:</p> <ul style="list-style-type: none"><code>node</code> - node for which statistics to be printed <p>Returns:</p> <ul style="list-style-type: none"><code>void</code> - None



[Qualnet®](#) is a Registered Trademark of [SCALABLE Network Technologies](#).

Copyright © 2001-2013 [SCALABLE Network Technologies, Inc.](#) All rights reserved.



QualNet 7.1 API Reference

APP_UTIL

This file describes Application Layer utility functions.

Function / Macro Summary

Return Type	Summary
MACRO	APP_GetTimerType(x) Get the timerType for a received App Layer Timer.
AppInfo*	APP_RegisterNewApp (Node* node, AppType appType, void * dataPtr) Insert a new application into the list of apps on this node.
void	APP_SetTimer (Node* node, AppType appType, int connId, short sourcePort, int timerType, clocktype delay) Set a new App Layer Timer and send to self after delay.
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).
Message*	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_TcpCloseConnection (Node * node, int connId)

	<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	<p>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>Application input parsing API. Parses the source and destination strings.At the same time validates those strings for multicast address.</p>
void	<p>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>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>
void	<p>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>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>
AppInfo*	<p>APP_RegisterNewApp(Node* node, AppType appType, void * dataPtr, short myPort)</p>
	<p>Also inserts the port number being used for this app in the port table.</p>
BOOL	<p>APP_IsFreePort(Node* node, short portNumber)</p>
	<p>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.</p>
short	<p>APP_GetFreePort(Node* node)</p>
void	<p>APP_InserInPortTable(Node* node, AppType appType, short myPort)</p>
unsigned short	<p>APP_GetProtocolType(Node* node, Message* msg)</p>
BOOL	<p>APP_AssignTos(char array tosString, char array tosValString, unsigned * tosVal)</p>
	<p>Application input parsing API. Parses the tos string and tos value strings.At the same time validates those strings for proper ranges.</p>
void	<p>APP_UnregisterApp(Node* node, void * dataPtr, bool freeData)</p>

	Remove an application from list of apps on this node.
void	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 Insert a new application into the list of apps on this node.	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 Set a new App Layer Timer and send to self after delay.	void APP_SetTimer (Node* node, AppType appType, int connId, short sourcePort, int timerType, clocktype delay) Parameters: <ul style="list-style-type: none">• node - node that is issuing the Timer.

	<ul style="list-style-type: none">• <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
<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>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
<p>APP_UdpSendNewHeaderVirtualDataWithPriority</p> <p>(Overloaded for IPv6) Allocate header + virtual data</p>	<p>Message* 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>Parameters:</p>

<p>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>	<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>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>Message*</code> - The created message
<p>APP_TcpServerListen.</p> <p>Listen on a server port.</p>	<p>void APP_TcpServerListen. (Node * node, AppType appType, NodeAddress serverAddr, short serverPort)</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>void APP_TcpServerListen. (Node * node, AppType appType, Address serverAddr, short serverPort)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>node</code> - Node pointer that the protocol is

	<ul style="list-style-type: none">• <code>appType</code> - which application initiates this request• <code>serverAddr</code> - server address• <code>serverPort</code> - server port number Returns: <ul style="list-style-type: none">• <code>void</code> - None
APP_TcpCloseConnection Close the connection.	void APP_TcpCloseConnection (Node * node, int connId) Parameters: <ul style="list-style-type: none">• <code>node</code> - Node pointer that the protocol is• <code>connId</code> - connection id. Returns: <ul style="list-style-type: none">• <code>void</code> - None
APP_InitMulticastGroupMembershipIfAny Start process of joining multicast group if need to do so.	void APP_InitMulticastGroupMembershipIfAny (Node * node, const NodeInput nodeInput) Parameters: <ul style="list-style-type: none">• <code>node</code> - node - node that is joining a group.• <code>nodeInput</code> - used to access configuration file. Returns: <ul style="list-style-type: none">• <code>void</code> - None
APP_CheckMulticastByParsingSourceAndDestString Application input parsing API. Parses the source and destination strings.At the same time validates those strings for multicast address.	void APP_CheckMulticastByParsingSourceAndDestString (Node * node, const char * inputString, const char * sourceString, NodeAddress * sourceNodeId, NodeAddress * sourceAddr, const char * destString, NodeAddress * destNodeId, NodeAddress * destAddr, BOOL * isDestMulticast) Parameters: <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> - The destination string.• <code>destNodeId</code> - A pointer to NodeAddress.• <code>destAddr</code> - A pointer to NodeAddress.

	<ul style="list-style-type: none">• <code>isDestMulticast</code> - Pointer to multicast checking flag. <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
<p>APP_ParsingSourceAndDestString</p> <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">• <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> - The destination string.• <code>destNodeId</code> - A pointer to NodeAddress.• <code>destAddr</code> - A pointer to NodeAddress.• <code>destType</code> - A pointer to Destinationtype. <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
<p>APP_ParsingSourceAndDestString</p> <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, NodeId * sourceNodeId, Address * sourceAddr, const char * destString, NodeId * destNodeId, Address * destAddr, DestinationType * destType)</p> <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> - The destination string.• <code>destNodeId</code> - A pointer to NodeAddress.

	<div>destAddr - A pointer to NodeAddress.</div> <div><ul style="list-style-type: none">destType - A pointer to DestinationType.</div> <div>Returns:</div> <div><ul style="list-style-type: none">void - None</div>
<div>APP_RegisterNewApp</div> <div>Also inserts the port number being used for this app in the port table.</div>	<div>AppInfo* APP_RegisterNewApp (Node* node, AppType appType, void * dataPtr, short myPort)</div> <div>Parameters:</div> <div><ul style="list-style-type: none">node - node that is registering the application.appType - application typedataPtr - pointer to the data space for this appmyPort - port number to be inserted in the port table</div> <div>Returns:</div> <div><ul style="list-style-type: none">AppInfo* - pointer to the new AppInfo data structure</div>
<div>APP_IsFreePort</div> <div>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.</div>	<div>BOOL APP_IsFreePort (Node* node, short portNumber)</div> <div>Parameters:</div> <div><ul style="list-style-type: none">node - node that is checking it's port tableportNumber - port number to check</div> <div>Returns:</div> <div><ul style="list-style-type: none">BOOL - indicates if the port is free</div>
<div>APP_GetFreePort</div>	<div>short APP_GetFreePort (Node* node)</div> <div>Parameters:</div> <div><ul style="list-style-type: none">node - node that is requesting a free port</div> <div>Returns:</div> <div><ul style="list-style-type: none">short - returns a free port</div>
<div>APP_InserInPortTable</div>	<div>void APP_InserInPortTable (Node* node, AppType appType, short myPort)</div> <div>Parameters:</div> <div><ul style="list-style-type: none">node - node that needs to be insert in port tableappType - application running at the portmyPort - port number to check</div>

	<p>Returns:</p> <ul style="list-style-type: none"><code>void</code> - None
<p>APP_GetProtocolType</p>	<p>unsigned short APP_GetProtocolType (Node* node, Message* msg)</p> <p>Parameters:</p> <ul style="list-style-type: none"><code>node</code> - node that received the message<code>msg</code> - pointer to the message received <p>Returns:</p> <ul style="list-style-type: none"><code>unsigned short</code> - protocol which will receive the message
<p>APP_AssignTos</p> <p>Application input parsing API. Parses the tos string and tos value strings.At the same time validates those strings for proper ranges.</p>	<p>BOOL APP_AssignTos (char array tosString, char array tosValString, unsigned * tosVal)</p> <p>Parameters:</p> <ul style="list-style-type: none"><code>tosString</code> - The tos string.<code>tosValString</code> - The tos value string.<code>tosVal</code> - A pointer to equivalent 8-bit TOS value. <p>Returns:</p> <ul style="list-style-type: none"><code>BOOL</code> - None
<p>APP_UnregisterApp</p> <p>Remove an application from list of apps on this node.</p>	<p>void APP_UnregisterApp (Node* node, void * dataPtr, bool freeData)</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 MEM_free) the dataPtr <p>Returns:</p> <ul style="list-style-type: none"><code>void</code> - None
<p>APP_UnregisterApp</p> <p>Also Remove the port number being used for this app in the port table.</p>	<p>void APP_UnregisterApp (Node* node, AppType appType, void * dataPtr, short myPort)</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

	<ul style="list-style-type: none">myPort - port number to be inserted in the port table <p>Returns:</p> <ul style="list-style-type: none">void - None
<p>APP_IsFreePort</p> <p>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.</p>	<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.
<p>APP_RemoveFromPortTable</p>	<p>void APP_RemoveFromPortTable (Node* node, short myPort)</p> <p>Parameters:</p> <ul style="list-style-type: none">node - node that needs to be remove from port tablemyPort - port number to check <p>Returns:</p> <ul style="list-style-type: none">void - None
<p>APP_ReportStatsDbReceiveEvent</p> <p>Report receive event to StatsDB app event table This function will check duplicate and out of order msgs</p>	<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 recieves the msgmsg - The received message or fragmentseqCache - Pointer to the sequence number cacheseqNo - Sequence number of the message or fragmentdelay - Delay of the message/fragmentjitter - Smoothed jitter of the received messagesize - Size of msg/fragment to be report to dbnumRcvd - # of msgs/frags received so farmsgStatus - This is for performance optimization. If <p>Returns:</p>

- `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 © 2001-2013 [SCALABLE Network Technologies, Inc.](#) All rights reserved.



QualNet 7.1 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
STRUCT	PortInfo Store port related information
STRUCT	AppMultimedia Store multimedia signalling related information
STRUCT	AppData Details of application data structure in node structure
STRUCT	AppTimer Timer structure used by applications

Function / Macro Summary

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

Constant / Data Structure Detail

Constant	APP_DEFAULT_TOS Ox00 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 funtion 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 nodevoip - Pointer to the voip application Returns: <ul style="list-style-type: none">void - NULL
TerminateConnectionType Multimedia callback funtion 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 nodevoip - Pointer to the voip application Returns: <ul style="list-style-type: none">void - NULL
IsHostCallingType Multimedia callback funtion 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: <ul style="list-style-type: none">BOOL - TRUE if the node is initiator, FALSE otherwise

IsHostCalledType Multimedia callback funtion to check whether node is in receiver mode	BOOL IsHostCalledType (Node* node) Parameters: <ul style="list-style-type: none">node - Pointer to the node Returns: <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 © 2001-2013 [SCALABLE Network Technologies, Inc.](#) All rights reserved.



QualNet 7.1 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 Returns: <ul style="list-style-type: none">• <code>void</code> - None
BUFFER_RemoveDataFromDataBuffer To remove data from the data buffer	void BUFFER_RemoveDataFromDataBuffer (DataBuffer* buffer, char* startLocation, int size) Parameters: <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 Returns: <ul style="list-style-type: none">• <code>void</code> - None
InitializeReassemblyBuffer Initialize Reassembly buffer	void InitializeReassemblyBuffer (ReassemblyBuffer* buffer, int size) Parameters: <ul style="list-style-type: none">• <code>buffer</code> - ReassemblyBuffer to be initialized• <code>size</code> - maximum allowable size of the buffer Returns: <ul style="list-style-type: none">• <code>void</code> - None
BUFFER_AddDataToAssemblyBuffer Appending data to the reassembly buffer	void BUFFER_AddDataToAssemblyBuffer (ReassemblyBuffer* buffer, char* data, int size, BOOL allowOverflow) Parameters: <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 Returns: <ul style="list-style-type: none">• <code>void</code> - None
BUFFER_ClearAssemblyBuffer clear the buffer	void BUFFER_ClearAssemblyBuffer (ReassemblyBuffer* buffer, int max, BOOL setSize) Parameters: <ul style="list-style-type: none">• <code>buffer</code> - Pointer to ReassemblyBuffer• <code>max</code> - the maximum size you want to set, if setSize 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
<p>BUFFER_SetAnticipatedSizeForAssemblyBuffer</p> <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
<p>BUFFER_AllocatePacketBuffer</p> <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
<p>BUFFER_AllocatePacketBufferWithInitialHeader</p> <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 To add header to buffer	<p>void BUFFER_AddHeaderToPacketBuffer (<code>PacketBuffer*</code> buffer, int headerSize, <code>char**</code> headerPtr)</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 To remove header from packet buffer	<p>void BUFFER_RemoveHeaderFromPacketBuffer (<code>PacketBuffer*</code> buffer, int headerSize, <code>char**</code> dataPtr)</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 To clear data from current buffer	<p>void BUFFER_ClearPacketBufferData (<code>PacketBuffer*</code> buffer)</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 Free the packet buffer passed as argument	<p>void BUFFER_FreePacketBuffer (<code>PacketBuffer*</code> buffer)</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 Add useful contents of source buffer as header to to	<p>void BUFFER_ConcatenatePacketBuffer (const <code>PacketBuffer*</code> source, <code>PacketBuffer*</code> dest)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>source</code> - Pointer to PacketBuffer

the destination buffer	<ul style="list-style-type: none">dest - Pointer to PacketBuffer Returns: <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 © 2001-2013 [SCALABLE Network Technologies, Inc.](#) All rights reserved.



QualNet 7.1 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	CircularBuffer::~CircularBuffer()
	Destructor
bool	CircularBuffer::create (Int32 queueSize)
	Memory allocation for Circular Buffer
void	CircularBuffer::release (void)
	To free the allocated memory
void	CircularBuffer::reset (void)
	reset position and wrap values
bool	CircularBuffer::getCount (Int32& count, Int32 operation)
	gets the number of bytes to read
Int32	CircularBuffer::lengthToEnd (Int32 operation)
	get the circular buffer's allocated size
bool	CircularBuffer::readWithCount (unsigned char* data, Int32& length)
	Read data from Buffer and pass the length of data read
bool	CircularBuffer::readFromBuffer (unsigned char* data, Int32 length, bool noIncrement)
	Reading the required no. of bytes from the circular buffer
bool	CircularBuffer::write (unsigned char* data, Int32 length)
	Write to the circular buffer
bool	CircularBuffer::read (unsigned char* buffer)
	To Read data from Buffer
Int32	CircularBuffer::getIndex (Int32 operation)

	get the circular buffer's allocated size
Int32	CircularBuffer.getCirBufSize (void none)
	get the circular buffer's allocated size
Int32	CircularBuffer.getContentssSize (void none)
	get the size of the queue's contents. This is the maximum amount of data that may be read.
void	CircularBuffer.dumpToStdout (void none)
	Output the contents of the circular buffer to stdout. This function is most useful when the contents of the buffer are human readable strings but it will work for any type of contents.
unsigned short	CircularBuffer.getIndex (void none)
	get the circular buffer's unique index

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
CircularBuffer.incPos	bool CircularBuffer.incPos (Int32 increment, Int32 operation)
increment read/write position based on operation	Parameters: <ul style="list-style-type: none">increment - How much will be incrementedoperation - Type of Operation (Read or Write)

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

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

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

get the circular buffer's allocated size	<p>Parameters:</p> <ul style="list-style-type: none">• none - None <p>Returns:</p> <ul style="list-style-type: none">• Int32 - circular buffer's allocated size
CircularBuffer.getContentsSize get the size of the queue's contents. This is the maximum amount of data that may be read.	<p>Int32 CircularBuffer.getContentsSize (void none)</p> <p>Parameters:</p> <ul style="list-style-type: none">• none - None <p>Returns:</p> <ul style="list-style-type: none">• Int32 - amount of data in buffer
CircularBuffer.dumpToStdout Output the contents of the circular buffer to stdout. This function is most useful when the contents of the buffer are human readable strings but it will work for any type of contents.	<p>void CircularBuffer.dumpToStdout (void none)</p> <p>Parameters:</p> <ul style="list-style-type: none">• none - None <p>Returns:</p> <ul style="list-style-type: none">• void - None
CircularBuffer.getIndex get the circular buffer's unique index	<p>unsigned short CircularBuffer.getIndex (void none)</p> <p>Parameters:</p> <ul style="list-style-type: none">• none - None <p>Returns:</p> <ul style="list-style-type: none">• 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 © 2001-2013 [SCALABLE Network Technologies, Inc.](#) All rights reserved.

QualNet 7.1 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	getSimStartTime(node) To get the simulation start 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_PrintClockInSeconds (clocktype clock, char * stringInSeconds) Print a clocktype value in second.The result is copied to string in Seconds
void	TIME_PrintClockInSeconds (clocktype clock, char * stringInSeconds, Node * node) Print a clocktype value in second.The result is copied to string in Seconds
void	TIME_PrintClockInSeconds (clocktype clock, char * stringInSeconds, PartitionData * partition) Print a clocktype value in second.The result is copied to string in Seconds
clocktype	TIME_ReturnMaxSimClock (Node* node) Return the maximum simulation clock
clocktype	TIME_ReturnStartSimClock (Node* node)

	Return the simulation start clock
--	-----------------------------------

Constant / Data Structure Detail

Constant	<p>CLOCKTYPE_MAX Platform dependent</p> <p>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.</p>
Constant	<p>NANO_SECOND ((clocktype) 1)</p> <p>Defined as basic unit of clocktype</p>
Constant	<p>MICRO_SECOND (1000 * NANO_SECOND)</p> <p>Defined as 1000 times the basic unit of clocktype</p>
Constant	<p>MILLI_SECOND (1000 * MICRO_SECOND)</p> <p>unit of time equal to 1000 times MICRO_SECOND</p>
Constant	<p>SECOND (1000 * MILLI_SECOND)</p> <p>simulation unit of time =1000 times MILLI_SECOND</p>
Constant	<p>MINUTE (60 * SECOND)</p> <p>unit of simulation time = 60 times SECOND</p>
Constant	<p>HOURL (60 * MINUTE)</p> <p>unit of simulation time = 60 times MINUTE</p>
Constant	<p>DAY (24 * HOURL)</p> <p>unit of simulation time = 24 times HOURL</p>
Constant	<p>PROCESS_IMMEDIATELY 0</p>

	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
getSimStartTime(node)	To get the simulation start 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 clocktypestringInSecond - string containing time in seconds Returns: <ul style="list-style-type: none">void - None
TIME_PrintClockInSecond Print a clocktype value in second.The result is copied to string in Seconds	void TIME_PrintClockInSecond (clocktype clock, char * stringInSecond, Node * node) Parameters: <ul style="list-style-type: none">clock - Time in clocktypestringInSecond - string containing time in secondsnode - Input node Returns: <ul style="list-style-type: none">void - None

TIME_PrintClockInSecond Print a clocktype value in second.The result is copied to string in Seconds	void TIME_PrintClockInSecond (clocktype clock, char * stringInSecond, PartitionData * partition) Parameters: <ul style="list-style-type: none">• clock - Time in clocktype• stringInSecond - string containing time in seconds• partition - Input partition Returns: <ul style="list-style-type: none">• void - None
TIME_ReturnMaxSimClock Return the maximum simulation clock	clocktype TIME_ReturnMaxSimClock (Node* node) Parameters: <ul style="list-style-type: none">• node - Input node Returns: <ul style="list-style-type: none">• clocktype - Returns maximum simulation time
TIME_ReturnStartSimClock Return the simulation start clock	clocktype TIME_ReturnStartSimClock (Node* node) Parameters: <ul style="list-style-type: none">• node - Input node Returns: <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 © 2001-2013 [SCALABLE Network Technologies, Inc.](#) All rights reserved.



QualNet 7.1 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
STRUCT	common coordinates
	Defines the three common coordinates
STRUCT	Coordinates
	Defines coordinates
STRUCT	Orientation
	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 eletromegnatic 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)

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

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

Constant / Data Structure Detail

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

	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 eletromegnatic 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 coordinates 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)

To normalize the angleIndex	<p>Parameters:</p> <ul style="list-style-type: none">angleIndex - angleIndexangleResolution - angleResolution <p>Returns:</p> <ul style="list-style-type: none">static int - Return normalized angleIndex
COORD_CalcDistance To calculate the distance between two nodes(points) given the coordinateSystemType and the coordinates of the two points	<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 systemposition1 - coordinates of a pointposition2 - coordinates of a pointdistance - distance between two points <p>Returns:</p> <ul style="list-style-type: none">BOOL - whether the distance is calculated from position1 to position2
COORD_CalcDistanceAndAngle To calculate the Distance and Angle	<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 Typeposition1 - Coordinates*position2 - Coordinates*distance - distanceDOA1 - DOA 1DOA2 - DOA 2 <p>Returns:</p> <ul style="list-style-type: none">static void - None
COORD_ChangeCoordinateSystem Re-calculate coordinate in a new coordinate system	<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">• <code>source</code> - coordinates of point to convert• <code>target_type</code> - coordinate system to• <code>target</code> - coordinate in new coordinate system Returns: <ul style="list-style-type: none">• <code>static void</code> - None
COORD_ChangeCoordinateSystem Re-calculate coordinate in a new coordinate system	static void COORD_ChangeCoordinateSystem (const Coordinates* const source, const CoordinateRepresentationType target_type, Coordinates* const target) Parameters: <ul style="list-style-type: none">• <code>source</code> - coordinates of point to convert• <code>target_type</code> - coordinate systme to• <code>target</code> - coordinate in new coordinate system Returns: <ul style="list-style-type: none">• <code>static void</code> - None
COORD_GeodeticToGeocentricCartesian Convert coordinate from GEODETIC to GEOCENTRIC_CARTESIAN	static void COORD_GeodeticToGeocentricCartesian (const Coordinates* const source, Coordinates* const target) Parameters: <ul style="list-style-type: none">• <code>source</code> - coordinate in GEODETIC• <code>target</code> - new coordinate in GEOCENTRIC_CARTESIAN Returns: <ul style="list-style-type: none">• <code>static void</code> - None
COORD_GeocentricCartesianToGeodetic Convert coordinate from GEOCENTRIC_CARTESIAN to GEODETIC	static void COORD_GeocentricCartesianToGeodetic (const Coordinates* const source, Coordinates* const target) Parameters: <ul style="list-style-type: none">• <code>source</code> - coordinate in GEOCENTRIC_CARTESIAN• <code>target</code> - new coordinate in GEODETIC Returns: <ul style="list-style-type: none">• <code>static void</code> - None
COORD_JGISToGeodetic Convert coordinate from JGIS to GEODETIC	static void COORD_JGISToGeodetic (const Coordinates* const source, Coordinates* const target) Parameters: <ul style="list-style-type: none">• <code>source</code> - coordinate in JGIS

	<ul style="list-style-type: none">target - new coordinate in GEODETIC Returns: <ul style="list-style-type: none">static void - None
COORD_JGISToUnreferencedCartesian Convert coordinate from JGIS to UNREFERENCED_CARTESIAN	static void COORD_JGISToUnreferencedCartesian (const Coordinates* const source, Coordinates* const target) Parameters: <ul style="list-style-type: none">source - coordinate in JGIStarget - new coordinate in UNREFERENCED_CARTESIAN Returns: <ul style="list-style-type: none">static void - None
COORD_ConvertToCoordinates Read the string in "buf" and provide the corresponding coordinates for the string.	static void COORD_ConvertToCoordinates (char* buf, Coordinates* coordinates) Parameters: <ul style="list-style-type: none">buf - input string to be converted to coordinatescoordinates - Pointer to the coordinates Returns: <ul style="list-style-type: none">static void - None
COORD_MapCoordinateSystemToType Set coordinates type field (CoordinateRepresentationType) based on the user-provided coordinate system (coordinate_system_type)	static void COORD_MapCoordinateSystemToType (int coordinateSystem, Coordinates* coordinates) Parameters: <ul style="list-style-type: none">coordinateSystem - enum value indicating coordinate systemcoordinates - Pointer to the coordinates Returns: <ul style="list-style-type: none">static void - None
COORD_NormailzeLongitude Correct the longitude value to between -180 and 180. This function assumes the coordinate system is LLA.	static void COORD_NormailzeLongitude (Coordinates* coordinates) Parameters: <ul style="list-style-type: none">coordinates - Pointer to the coordinates Returns: <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">• <code>coordinateSystemType</code> - Cartesian or Geodetic• <code>sw</code> - Pointer to the SW corner (0,0) if Cartesian• <code>ne</code> - Pointer to the NE corner (dimensions if Cartesian)• <code>point</code> - Pointer to the coordinates <p>Returns:</p> <ul style="list-style-type: none">• <code>bool</code> - True if within range
<p>COORD_RegionsOverlap</p> <p>Determine whether the given regions overlap at all.</p>	<p>bool COORD_RegionsOverlap (int coordinateSystemType, Coordinates* sw1, Coordinates* ne1, Coordinates* sw2, Coordinates* ne2)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>coordinateSystemType</code> - Cartesian or Geodetic• <code>sw1</code> - Pointer to the SW corner of the first region• <code>ne1</code> - Pointer to the NE corner of the first region• <code>sw2</code> - Pointer to the SW corner of the second region• <code>ne2</code> - Pointer to the NE corner of the second region <p>Returns:</p> <ul style="list-style-type: none">• <code>bool</code> - true if the regions overlap at all.
<p>COORD_LongitudeDelta</p> <p>Convenience function for geodetic that, given two longitudes, returns the difference (in degrees) in the shorter direction.</p>	<p>void COORD_LongitudeDelta (CoordinateType long1, CoordinateType long2)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>long1</code> - coordinate 1• <code>long2</code> - coordinate 2 <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
<p>COORD_PrintCoordinates</p> <p>Prints the coordinates in a human readable format.</p>	<p>void COORD_PrintCoordinates (int coordinateSystemType, Coordinates* point)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>coordinateSystemType</code> - Cartesian or Geodetic• <code>point</code> - 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 © 2001-2013 [SCALABLE Network Technologies, Inc.](#) All rights reserved.

QualNet 7.1 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
extern BOOL	ERROR_WriteError (int type, char* condition, char* msg, char* file, int lineno) 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.
void	ERROR_InstallHandler (int type, char* condition, char* msg, char* file, int lineno, QErrorHandler functionPointer) 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.)
void	ERROR_ReportMissingAddon (const char* model, const char* addon) 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.
void	ERROR_ReportMissingInterface (const char* model, const char* iface) 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.
void	ERROR_ReportMissingLibrary (const char* model, const char* library) 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.

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

	Defines the ERROR_WARNING constant
--	------------------------------------

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 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.	<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 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.)	<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

	<div>Returns:</div> <div><ul style="list-style-type: none">void - None</div>
<div><div>ERROR_ReportMissingAddon</div><div>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.</div></div>	<div>void ERROR_ReportMissingAddon (const char* model, const char* addon)</div> <div>Parameters:</div> <div><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</div> <div>Returns:</div> <div><ul style="list-style-type: none">void - None</div>
<div><div>ERROR_ReportMissingInterface</div><div>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.</div></div>	<div>void ERROR_ReportMissingInterface (const char* model, const char* iface)</div> <div>Parameters:</div> <div><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</div> <div>Returns:</div> <div><ul style="list-style-type: none">void - None</div>
<div><div>ERROR_ReportMissingLibrary</div><div>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.</div></div>	<div>void ERROR_ReportMissingLibrary (const char* model, const char* library)</div> <div>Parameters:</div> <div><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</div> <div>Returns:</div> <div><ul style="list-style-type: none">void - None</div>

QualNet 7.1 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_ThreadedMessage A struct containing data needed to send a message from an external thread to the main thread.
STRUCT	EXTERNAL_ThreadedForwarded

	<p>A struct containing data needed to send a forwarded packet from the main thread to an external forward function</p>
STRUCT	<p>EXTERNAL_Mapping</p> <p>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.</p>
STRUCT	<p>EXTERNAL_MobilityEvent</p> <p>A linked list of mobility events</p>
STRUCT	<p>EXTERNAL_MobilityEventBuffer</p> <p>A buffer containing all mobility events yet to be added to the simulation.</p>
STRUCT	<p>EXTERNAL_InterfaceList</p> <p>A list containing all of the registered external entities</p>
STRUCT	<p>EXTERNAL_Interface</p> <p>The information pertaining to one external interface</p>

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	<p>EXTERNAL_SetTimeManagementRealTime(EXTERNAL_Interface* iface, clocktype lookahead)</p>

	<p>Turns time management on and specifies the lookahead value. The lookahead value may be changed later by calling <code>EXTERNAL_ChangeRealTimeLookahead()</code>.</p>
void	<p>EXTERNAL_ChangeRealTimeLookahead(EXTERNAL_Interface* iface, clocktype lookahead)</p>
	<p>Modifies the lookahead value. Must be called after <code>EXTERNAL_SetTimeManagementRealTime()</code>. May be called during the simulation.</p>
void	<p>EXTERNAL_InitializeWarmupParams(EXTERNAL_Interface* iface, NodeInput* nodeInput)</p>
void	<p>EXTERNAL_RealtimeIndicator(EXTERNAL_Interface* iface, NodeInput* nodeInput)</p> <p>for realtime indicator initialization</p>
void	<p>EXTERNAL_SetWarmupTime(EXTERNAL_Interface* iface, clocktype warmup)</p> <p>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	<p>EXTERNAL_BeginWarmup(EXTERNAL_Interface* iface)</p> <p>Each interface that calls <code>EXTERNAL_SetWarmupTime</code> must call <code>EXTERNAL_BeginWarmup</code> when it is ready to enter warmup time.</p>
clocktype	<p>EXTERNAL_QueryWarmupTime(EXTERNAL_Interface* iface)</p> <p>Get the warmup time for the entire simulation. Interfaces should use this function to test when warmup time is over.</p>
BOOL	<p>EXTERNAL_IsInWarmup(EXTERNAL_Interface* iface)</p> <p>Check if QualNet is in the warmup phase</p>
BOOL	<p>EXTERNAL_IsInWarmup(PartitionData* partitionData)</p> <p>Check if QualNet is in the warmup phase</p>
void	<p>EXTERNAL_Pause(EXTERNAL_Interface* iface)</p> <p>Pause every interface. Only usable when running in real-time.</p>
void	<p>EXTERNAL_Resume(EXTERNAL_Interface* iface)</p> <p>Resume every interface. Only usable when running in real-time, and after calling pause.</p>
clocktype	<p>EXTERNAL_QueryExternalTime(EXTERNAL_Interface* iface)</p>

	<p>This function will return the External Time of an external interface</p>
clocktype	<p>EXTERNAL_QuerySimulationTime(EXTERNAL_Interface* iface)</p>
	<p>This function will return the Simulation Time</p>
void	<p>EXTERNAL_Sleep(clocktype amount)</p>
	<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>
void	<p>EXTERNAL_SetReceiveDelay(EXTERNAL_Interface* iface, clocktype delay)</p>
	<p>This function will set the minimum delay between two consecutive calls to the receive function. The time used is the simulation time.</p>
void	<p>EXTERNAL_SendMessage(EXTERNAL_Interface* iface, Node* node, Message* msg, clocktype timestamp)</p>
	<p>This function will send a message from the external interface. This function is thread-safe.</p>
void	<p>EXTERNAL_ForwardData(EXTERNAL_Interface* iface, Node* node, void* forwardData, int forwardSize, EXTERNAL_ForwardData_ReceiverOpt FwdReceiverOpt)</p>
	<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>
void	<p>EXTERNAL_RemoteForwardData(EXTERNAL_Interface* iface, Node* node, void* forwardData, int forwardSize, int partitionId)</p>
	<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 and external interface on a different partition.</p>
void	<p>EXTERNAL_ForwardDataTimeStamped(EXTERNAL_Interface* iface, Node* node, Message* message, clocktype timestamp)</p>
	<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>
void	<p>EXTERNAL_UserFunctionRegistration(EXTERNAL_InterfaceList * list, NodeInput* nodeInput)</p>
	<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>
void	<p>EXTERNAL_InitializeInterface(EXTERNAL_Interface* iface)</p>
	<p>This function will initialize an external interface</p>
void	<p>EXTERNAL_FinalizeExternalInterface(EXTERNAL_Interface* iface)</p>

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

	<p>This function will call all receive function that were not started in a thread</p>
void	<p>EXTERNAL_CallFinalizeFunctions(EXTERNAL_InterfaceList* list)</p> <p>This function will call all finalize functions</p>
void	<p>EXTERNAL_InitializeExternalInterfaces(partitionData* partitionData)</p> <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>
void	<p>EXTERNAL_PostInitialize(partitionData* partitionData)</p> <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>
void	<p>EXTERNAL_GetExternalMessages(partitionData* partitionData, clocktype nextInternalEventTime)</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>
void	<p>EXTERNAL_Finalize(partitionData* partitionData)</p> <p>Shuts down interfaces to external simulators</p>
void	<p>EXTERNAL_SetActive(partitionData* partitionData)</p> <p>Sets isActive parameter based on interface registration</p>
void	<p>EXTERNAL_DeactivateInterface(EXTERNAL_Interface* ifaceToDeactivate)</p> <p>Remove the indicated interface for the list of currently activateed interfaces.</p>
void	<p>EXTERNAL_ProcessEvent(Node* node, Message* msg)</p> <p>Process events meant for external code.</p>
clocktype	<p>GetNextInternalEventTime(PartitionData* partitionData)</p> <p>Get the next internal event on the given partition. This includes both regular events and mobility events.</p>
void	<p>EXTERNAL_SendRtssNotification(Node* node)</p> <p>To send Rtss notification over all active external interfaces that support Rtss</p>

Constant / Data Structure Detail

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

	<p>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.</p>
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
<p>EXTERNAL_RegisterExternalInterface</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>	<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 interfacesname - 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
<p>EXTERNAL_RegisterExternalInterface</p> <p>This function is an overloaded variation. for registering a new external interface with QualNet</p>	<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

	<ul style="list-style-type: none">• <code>name</code> - The name of the external interface.• <code>params</code> - The performance parameters• <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
<p>EXTERNAL_RegisterFunction</p> <p>Register a new function for an interface.</p>	<p>void EXTERNAL_RegisterFunction (EXTERNAL_Interface* iface, EXTERNAL_FunctionType type, EXTERNAL function)</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
<p>EXTERNAL_SetTimeManagementRealTime</p> <p>Turns time management on and specifies the lookahead value. The lookahead value may be changed later by calling EXTERNAL_ChangeRealTimeLookahead().</p>	<p>void EXTERNAL_SetTimeManagementRealTime (EXTERNAL_Interface* iface, clocktype lookahead)</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
<p>EXTERNAL_ChangeRealTimeLookahead</p> <p>Modifies the lookahead value. Must be called after EXTERNAL_SetTimeManagementRealTime(). May be called during the simulation.</p>	<p>void EXTERNAL_ChangeRealTimeLookahead (EXTERNAL_Interface* iface, clocktype lookahead)</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
<p>EXTERNAL_InitializeWarmupParams</p>	<p>void EXTERNAL_InitializeWarmupParams (EXTERNAL_Interface* iface, NodeInput* nodeInput)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>iface</code> - The external interface

	<ul style="list-style-type: none">nodeInput - The configuration file. Returns: <ul style="list-style-type: none">void - None
EXTERNAL_RealtimeIndicator for realtime indicator initialization	void EXTERNAL_RealtimeIndicator (EXTERNAL_Interface* iface, NodeInput* nodeInput) Parameters: <ul style="list-style-type: none">iface - The external interfacenodeInput - The configuration file. Returns: <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.	void EXTERNAL_SetWarmupTime (EXTERNAL_Interface* iface, clocktype warmup) Parameters: <ul style="list-style-type: none">iface - The external interfacewarmup - The warmup time for this interface Returns: <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.	void EXTERNAL_BeginWarmup (EXTERNAL_Interface* iface) Parameters: <ul style="list-style-type: none">iface - The external interface Returns: <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.	clocktype EXTERNAL_QueryWarmupTime (EXTERNAL_Interface* iface) Parameters: <ul style="list-style-type: none">iface - The external interface Returns: <ul style="list-style-type: none">clocktype - The inclusive end of warmup time. -1 if no warmup time.
EXTERNAL_IsInWarmup	BOOL EXTERNAL_IsInWarmup (EXTERNAL_Interface* iface) Parameters:

Check if QualNet is in the warmup phase	<div>iface - The external interface</div> <div>Returns:<ul style="list-style-type: none">• <code>BOOL</code> - 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</div>
<div>EXTERNAL_IsInWarmup</div> <div>Check if QualNet is in the warmup phase</div>	<div>BOOL EXTERNAL_IsInWarmup (PartitionData* partitionData)</div> <div>Parameters:<ul style="list-style-type: none">• <code>partitionData</code> - pointer to partition's data structure</div> <div>Returns:<ul style="list-style-type: none">• <code>BOOL</code> - TRUE if in warmup, FALSE if not</div>
<div>EXTERNAL_Pause</div> <div>Pause every interface. Only usable when running in real-time.</div>	<div>void EXTERNAL_Pause (EXTERNAL_Interface* iface)</div> <div>Parameters:<ul style="list-style-type: none">• <code>iface</code> - The external interface</div> <div>Returns:<ul style="list-style-type: none">• <code>void</code> - None</div>
<div>EXTERNAL_Resume</div> <div>Resume every interface. Only usable when running in real-time, and after calling pause.</div>	<div>void EXTERNAL_Resume (EXTERNAL_Interface* iface)</div> <div>Parameters:<ul style="list-style-type: none">• <code>iface</code> - The external interface</div> <div>Returns:<ul style="list-style-type: none">• <code>void</code> - None</div>
<div>EXTERNAL_QueryExternalTime</div> <div>This function will return the External Time of an external interface</div>	<div>clocktype EXTERNAL_QueryExternalTime (EXTERNAL_Interface* iface)</div> <div>Parameters:<ul style="list-style-type: none">• <code>iface</code> - The external interface</div> <div>Returns:<ul style="list-style-type: none">• <code>clocktype</code> - The External Time. Returns <code>EXTERNAL_MAX_TIME</code> if no time function is defined.</div>
<div>EXTERNAL_QuerySimulationTime</div> <div>This function will return the Simulation Time</div>	<div>clocktype EXTERNAL_QuerySimulationTime (EXTERNAL_Interface* iface)</div> <div>Parameters:<ul style="list-style-type: none">• <code>iface</code> - The external interface</div>

	<div>Returns:</div> <ul style="list-style-type: none"><code>clocktype</code> - The Simulation Time
<div>EXTERNAL_Sleep</div> <div>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.</div>	<div>void EXTERNAL_Sleep (clocktype amount)</div> <div>Parameters:</div> <ul style="list-style-type: none"><code>amount</code> - The amount of time to sleep <div>Returns:</div> <ul style="list-style-type: none"><code>void</code> - None
<div>EXTERNAL_SetReceiveDelay</div> <div>This function will set the minimum delay between two consecutive calls to the receive function. The time used is the simulation time.</div>	<div>void EXTERNAL_SetReceiveDelay (EXTERNAL_Interface* iface, clocktype delay)</div> <div>Parameters:</div> <ul style="list-style-type: none"><code>iface</code> - The external interface<code>delay</code> - The minimum delay <div>Returns:</div> <ul style="list-style-type: none"><code>void</code> - None
<div>EXTERNAL_SendMessage</div> <div>This function will send a message from the external interface. This function is thread-safe.</div>	<div>void EXTERNAL_SendMessage (EXTERNAL_Interface* iface, Node* node, Message* msg, clocktype timestamp)</div> <div>Parameters:</div> <ul style="list-style-type: none"><code>iface</code> - The external interface<code>node</code> - Node sending the message<code>msg</code> - The message to send<code>timestamp</code> - The timestamp for this message. Since this message is <div>Returns:</div> <ul style="list-style-type: none"><code>void</code> - None
<div>EXTERNAL_ForwardData</div> <div>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.</div>	<div>void EXTERNAL_ForwardData (EXTERNAL_Interface* iface, Node* node, void* forwardData, int forwardSize, EXTERNAL_ForwardData_ReceiverOpt FwdReceiverOpt)</div> <div>Parameters:</div> <ul style="list-style-type: none"><code>iface</code> - The external interface<code>node</code> - The node that is forwarding the data<code>forwardData</code> - The data to forward<code>forwardSize</code> - The size of the data to forward

	<ul style="list-style-type: none">FwdReceiverOpt - Whether to store the <p>Returns:</p> <ul style="list-style-type: none">void - None
<p>EXTERNAL_RemoteForwardData</p> <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 and 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 interfacenode - The node that is forwarding the dataforwardData - The data to forwardforwardSize - The size of the data to forwardpartitionId - The partition Id to forward the message to <p>Returns:</p> <ul style="list-style-type: none">void - None
<p>EXTERNAL_ForwardDataTimeStamped</p> <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 interfacenode - The node that is forwarding the datamessage - The messagetimestamp - The time stamp. This value is in external <p>Returns:</p> <ul style="list-style-type: none">void - None
<p>EXTERNAL_UserFunctionRegistration</p> <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 interfacesnodeInput - The configuration file <p>Returns:</p> <ul style="list-style-type: none">void - None
<p>EXTERNAL_InitializeInterface</p>	<p>void EXTERNAL_InitializeInterface (EXTERNAL_Interface* iface)</p>

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

and make sures that if simProps needs to change it is changed and then a broadcast message is sent to other partitions.	
EXTERNAL_FinalizeInterfaceList This function will finalize all ExternalInterfaces in the list, as well as the list itself	void EXTERNAL_FinalizeInterfaceList (EXTERNAL_InterfaceList* list) Parameters: <ul style="list-style-type: none">list - The external interface list Returns: <ul style="list-style-type: none">void - None
EXTERNAL_GetInterfaceByName This function will search an interface list for an interface with the given name	EXTERNAL_Interface* EXTERNAL_GetInterfaceByName (EXTERNAL_InterfaceList* list, char* name) Parameters: <ul style="list-style-type: none">list - The external interface listname - The interface's name Returns: <ul style="list-style-type: none">EXTERNAL_Interface* - The interface, NULL if not found
EXTERNAL_CallInitializeFunctions This function will call all initialize functions	void EXTERNAL_CallInitializeFunctions (EXTERNAL_InterfaceList* list, NodeInput* nodeInput) Parameters: <ul style="list-style-type: none">list - The list of external interfacesnodeInput - The input configuration file Returns: <ul style="list-style-type: none">void - None
EXTERNAL_CallInitializeNodesFunctions This function will call all intialize nodes functions	void EXTERNAL_CallInitializeNodesFunctions (EXTERNAL_InterfaceList* list, NodeInput* nodeInput) Parameters: <ul style="list-style-type: none">list - The list of external interfacesnodeInput - The input configuration file Returns: <ul style="list-style-type: none">void - None
EXTERNAL_StartThreads This function will start the receive/forward	void EXTERNAL_StartThreads (EXTERNAL_InterfaceList* list) Parameters: <ul style="list-style-type: none">list - The list of external interfaces

<p>threads for all threaded interfaces. Called after <code>EXTERNAL_CallInitializeNodesFunctions</code>.</p>	<p>Returns:</p> <ul style="list-style-type: none"><code>void</code> - None
<p>EXTERNAL_CalculateMinSimulationHorizon</p> <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 >= 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"><code>list</code> - The list of external interfaces<code>now</code> - The current time <p>Returns:</p> <ul style="list-style-type: none"><code>clocktype</code> - The minimum Simulation Horizon, or <code>EXTERNAL_MAX_TIME</code> if no horizon.
<p>EXTERNAL_CallReceiveFunctions</p> <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"><code>list</code> - The list of external interfaces <p>Returns:</p> <ul style="list-style-type: none"><code>void</code> - None
<p>EXTERNAL_CallFinalizeFunctions</p> <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"><code>list</code> - The list of external interfaces <p>Returns:</p> <ul style="list-style-type: none"><code>void</code> - None
<p>EXTERNAL_InitializeExternalInterfaces</p> <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"><code>partitionData</code> - pointer to data for this partition <p>Returns:</p> <ul style="list-style-type: none"><code>void</code> - None
<p>EXTERNAL_PostInitialize</p> <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</p>	<p>void EXTERNAL_PostInitialize (partitionData* partitionData)</p> <p>Parameters:</p> <ul style="list-style-type: none"><code>partitionData</code> - pointer to data for this partition <p>Returns:</p>

developer can use either this function, the preceding one or both.	<ul style="list-style-type: none">void - None
EXTERNAL_GetExternalMessages 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 EXTERNAL_GetExternalMessages (partitionData* partitionData, clocktype nextInternalEventTime) Parameters: <ul style="list-style-type: none">partitionData - pointer to data for this partitionnextInternalEventTime - the time of the next event, Returns: <ul style="list-style-type: none">void - None
EXTERNAL_Finalize Shuts down interfaces to external simulators	void EXTERNAL_Finalize (partitionData* partitionData) Parameters: <ul style="list-style-type: none">partitionData - pointer to data for this partition Returns: <ul style="list-style-type: none">void - None
EXTERNAL_SetActive Sets isActive parameter based on interface registration	void EXTERNAL_SetActive (partitionData* partitionData) Parameters: <ul style="list-style-type: none">partitionData - pointer to data for this partition Returns: <ul style="list-style-type: none">void - None
EXTERNAL_DeactivateInterface Remove the indicated interface for the list of currently activateed interfaces.	void EXTERNAL_DeactivateInterface (EXTERNAL_Interface* ifaceToDeactivate) Parameters: <ul style="list-style-type: none">ifaceToDeactivate - Pointer to the interface Returns: <ul style="list-style-type: none">void - None
EXTERNAL_ProcessEvent Process events meant for external code.	void EXTERNAL_ProcessEvent (Node* node, Message* msg) Parameters: <ul style="list-style-type: none">node - Pointer to node data structure.msg - Message to be processed. Returns:

	<ul style="list-style-type: none">void - None
GetNextInternalEventTime Get the next internal event on the given partition. This includes both regular events and mobility events.	clocktype GetNextInternalEventTime (PartitionData* partitionData) Parameters: <ul style="list-style-type: none">partitionData - Pointer to the partition Returns: <ul style="list-style-type: none">clocktype - The next internal event
EXTERNAL_SendRtssNotification To send Rtss notification over all active external interfaces that support Rtss	void EXTERNAL_SendRtssNotification (Node* node) Parameters: <ul style="list-style-type: none">node - Node pointer. Returns: <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 © 2001-2013 [SCALABLE Network Technologies, Inc.](#) All rights reserved.



QualNet 7.1 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)

	<p>This function will increase the maximum size of the VarArray so that it can contain at least "size" bytes.</p>
void	<p>EXTERNAL_VarArrayAppendData(EXTERNAL_VarArray* array, char* data, unsigned int size)</p>
	<p>This function will add data to the end of the VarArray. The size of the VarArray will be increased if necessary.</p>
void	<p>EXTERNAL_VarArrayConcatString(EXTERNAL_VarArray* array, char* string)</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 EXTERNAL_VarArrayAppendData should be used instead.</p>
void	<p>EXTERNAL_VarArrayFree(EXTERNAL_VarArray* array)</p>
	<p>This function will free all memory allocated to the VarArray</p>
void	<p>EXTERNAL_hton(void* ptr, unsigned size)</p>
	<p>Convert data from host byte order to network byte order</p>
void	<p>EXTERNAL_ntoh(void* ptr, unsigned size)</p>
	<p>Convert data from network byte order to host byte order</p>
void	<p>EXTERNAL_swapBitfield(void* ptr, unsigned size)</p>
void	<p>EXTERNAL_SocketInit(EXTERNAL_Socket* socket)</p>
	<p>Initialize a socket. Must be called before all other socket API calls on the individual socket.</p>
EXTERNAL_SocketErrorType	<p>EXTERNAL_SocketInitUDP(EXTERNAL_Socket* socket)</p>
	<p>Initialize a UDP socket. Must be called before all other socket API calls on the individual socket.</p>
bool	<p>EXTERNAL_SocketValid(EXTERNAL_Socket* socket)</p>
	<p>Check if a socket connection is valid and no errors have occurred.</p>
EXTERNAL_SocketErrorType	<p>EXTERNAL_SocketListen(EXTERNAL_Socket* listenSocket, int port, EXTERNAL_Socket* connectSocket)</p>
	<p>Listen and accept a connections on a socket. This function is a wrapper for EXTERNAL_SocketInitListen and EXTERNAL_SocketAccept.</p>
EXTERNAL_SocketErrorType	<p>EXTERNAL_SocketInitListen(EXTERNAL_Socket* listenSocket, int port)</p>

	<p>Initialize an input socket and have it listen on the given port. Call <code>EXTERNAL_SocketAccept</code> to accept connections on the socket. Call <code>EXTERNAL_SocketDataAvailable</code> 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 <code>EXTERNAL_SocketDataAvailable</code> 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 <code>EXTERNAL_DataNotSent</code> 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	<div>EXTERNAL_DEFAULT_VAR_ARRAY_SIZE 512</div> <div>The default size of a VarArray</div>
Constant	<div>THREADED_BUFFER_SIZE 2000000</div> <div>The thread buffer size</div>
Enumeration	<div>EXTERNAL_SocketErrorType</div> <div>A listing of error types that could occur.</div>
Structure	<div>EXTERNAL_VarArray</div> <div>A variable sized array. This structure is primarily used to assemble data to be sent on a socket connection.</div>
Structure	<div>EXTERNAL_Socket</div> <div>The socket data structure</div>

Function / Macro Detail

Function / Macro	Format
<div>EXTERNAL_VarArrayInit</div> <div>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.</div>	<div>void EXTERNAL_VarArrayInit (EXTERNAL_VarArray* array, unsigned int size)</div> <div>Parameters:</div> <div><ul style="list-style-type: none">array - Pointer to the uninitialized VarArraysize - The initial size of the array in bytes . Defaults</div> <div>Returns:</div> <div><ul style="list-style-type: none">void - None</div>
<div>EXTERNAL_VarArrayAccomodateSize</div>	<div>void EXTERNAL_VarArrayAccomodateSize (EXTERNAL_VarArray* array, unsigned int size)</div> <div>Parameters:</div>

<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>void EXTERNAL_VarArrayAppendData (EXTERNAL_VarArray* array, char* data, unsigned int size)</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 EXTERNAL_VarArrayAppendData should be used instead.</p>	<p>void EXTERNAL_VarArrayConcatString (EXTERNAL_VarArray* array, char* string)</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>void EXTERNAL_VarArrayFree (EXTERNAL_VarArray* array)</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>void EXTERNAL_hton (void* ptr, unsigned size)</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 Convert data from network byte order to host byte order	void EXTERNAL_ntoh (void* ptr, unsigned size) Parameters: <ul style="list-style-type: none">ptr - Pointer to the datasize - Size of the data Returns: <ul style="list-style-type: none">void - None
EXTERNAL_swapBitfield	void EXTERNAL_swapBitfield (void* ptr, unsigned size) Parameters: <ul style="list-style-type: none">ptr - Pointer to the datasize - Size of the data (in bytes) Returns: <ul style="list-style-type: none">void - None
EXTERNAL_SocketInit Initialize a socket. Must be called before all other socket API calls on the individual socket.	void EXTERNAL_SocketInit (EXTERNAL_Socket* socket) Parameters: <ul style="list-style-type: none">socket - Pointer to the socket Returns: <ul style="list-style-type: none">void - None
EXTERNAL_SocketInitUDP Initialize a UDP socket. Must be called before all other socket API calls on the individual socket.	EXTERNAL_SocketErrorType EXTERNAL_SocketInitUDP (EXTERNAL_Socket* socket) Parameters: <ul style="list-style-type: none">socket - Pointer to the socket Returns: <ul style="list-style-type: none">EXTERNAL_SocketErrorType - Error type
EXTERNAL_SocketValid Check if a socket connection is valid and no errors have occurred.	bool EXTERNAL_SocketValid (EXTERNAL_Socket* socket) Parameters: <ul style="list-style-type: none">socket - Pointer to the socket Returns:

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

<div><div>EXTERNAL_SocketRecv</div><div>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.</div></div>	<div><div>EXTERNAL_SocketErrorType EXTERNAL_SocketRecv (EXTERNAL_Socket* socket, char* data, unsigned int size, unsigned int* size, bool block)</div><div>Parameters:<ul style="list-style-type: none">socket - Pointer to the socketdata - Pointer to the destinationsize - The amount of data to receive in bytessize - The number of bytes received. This could be lessblock - TRUE if the call can block, FALSE if non-blocking.</div><div><div>Returns:</div><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.</div></div>
<div><div>EXTERNAL_SocketRecv</div><div>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.</div></div>	<div><div>EXTERNAL_SocketErrorType EXTERNAL_SocketRecv (EXTERNAL_Socket* socket, char* data, unsigned int size, unsigned int* size, int* ip, int* port, bool block)</div><div>Parameters:<ul style="list-style-type: none">socket - Pointer to the socketdata - Pointer to the destinationsize - The amount of data to receive in bytessize - The number of bytes received. This could be lessip - The IP address it was received fromport - The port it was received fromblock - TRUE if the call can block, FALSE if non-blocking.</div><div><div>Returns:</div><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.</div></div>
<div><div>EXTERNAL_SocketRecv</div><div>Receive data on a connected socket. Continues reading until a '\n' character is found. This function always blocks.</div></div>	<div><div>EXTERNAL_SocketErrorType EXTERNAL_SocketRecv (EXTERNAL_Socket* socket, std data)</div><div>Parameters:<ul style="list-style-type: none">socket - Pointer to the socketdata - string*</div><div><div>Returns:</div><ul style="list-style-type: none">EXTERNAL_SocketErrorType - EXTERNAL_NoSocketError if successful, different error if not successful which could be</div></div>

	due to a number of reasons.
EXTERNAL_SocketClose Close a socket. Must be called for each socket that is listening or connected.	EXTERNAL_SocketErrorType EXTERNAL_SocketClose (EXTERNAL_Socket* socket) Parameters: <ul style="list-style-type: none">• socket - Pointer to the socket Returns: <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 © 2001-2013 [SCALABLE Network Technologies, Inc.](#) All rights reserved.



QualNet 7.1 API Reference

EXTERNAL_UTILITIES

This file describes utilities for external interfaces.

Constant / Data Structure Summary

Type	Name
ENUMERATION	ExternalsScheduleType 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 UDP 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 A table. Generally used for storing external packet data, but can be used for anything.
STRUCT	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)

	<p>This function will retrieve the earliest record in the table or NULL if the table is empty.</p>
BOOL	<p>EXTERNAL_GetEarliestRecord(EXTERNAL_Table* table, char* data)</p>
	<p>This function will check if a data pointer is still in the table.</p>
EXTERNAL_TableRecord*	<p>EXTERNAL_FreeRecord(EXTERNAL_Table* table)</p>
	<p>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.</p>
void	<p>EXTERNAL_SendDataAppLayerUDP(EXTERNAL_Interface* iface, NodeAddress from, NodeAddress to, char* data, int dataSize, clocktype timestamp, AppType app, TraceProtocolType trace, TosType priority)</p>
	<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>
void	<p>EXTERNAL_SendDataAppLayerUDP(EXTERNAL_Interface* iface, NodeAddress from, NodeAddress to, char* header, int headerSize, int virtualDataSize, clocktype timestamp, AppType app, TraceProtocolType trace, TosType priority)</p>
	<p>Sends virtual data originating from the app layer using UDP. When the packet reaches its destination it will call the forward forward function of the external interface, if it exists.</p>
void	<p>EXTERNAL_SendDataAppLayerTCP(EXTERNAL_Interface* iface, NodeAddress from, NodeAddress to, char* data, int dataSize, clocktype timestamp)</p>
	<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>
void	<p>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)</p>
	<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>
void	<p>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>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>
void	<p>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>
	<p>Sends data originating from network layer. No provisions are made for handling this data once it enters the QualNet network. This is the</p>

	responsibility of the external interface or protocols the data is sent to.
void	<div><div>EXTERNAL_SendIpv6DataNetworkLayer(EXTERNAL_Interface* iface, Address from, Address srcAddr, Address destAddr, TosType tos, unsigned char protocol, unsigned int hlim, char* payload, int payloadSize, clocktype timestamp)</div><div></div><div>Sends ipv6 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.</div></div>
void	<div><div>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)</div><div></div><div>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.</div></div>
void	<div><div>EXTERNAL_SendNetworkLayerPacket(Node* node, Message* msg)</div><div></div><div>Sends the packet from EXTERNAL_SendDataNetworkLayer after some delay. This function should never be called directly.</div></div>
void	<div><div>EXTERNAL_CreateMapping(EXTERNAL_Interface* iface, char* key, int keySize, char* value, int valueSize)</div><div></div><div>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.</div></div>
int	<div><div>EXTERNAL_ResolveMapping(EXTERNAL_Interface* iface, char* key, int keySize, char** value, int* valueSize)</div><div></div><div>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.</div></div>
int	<div><div>EXTERNAL_DeleteMapping(EXTERNAL_Interface* iface, char* key, int keySize)</div><div></div><div>Deletes a mapping created by EXTERNAL_CreateMapping.</div></div>
void	<div><div>EXTERNAL_ActivateNode(EXTERNAL_Interface* iface, Node* node)</div><div></div><div>Activate a node so that it can begin processing events.</div></div>
void	<div><div>EXTERNAL_DectivateNode(EXTERNAL_Interface* iface, Node* node)</div><div></div><div>Dectivate a node so that it stops processing events.</div></div>
void	<div><div>EXTERNAL_PHY_SetTxPower(Node* node, int phyIndex, double newTxPower)</div><div></div><div>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.</div></div>
void	<div><div>(Node* node, int phyIndex, double * txPowerPtr)</div><div></div><div></div></div>

	<p>EXTERNAL_PHY_GetTxPower</p> <p>Just like PHY_GetTxPower (), but able to handle getting transmission power when node is owned by a remote partition.</p>
void	<p>EXTERNAL_ChangeNodePosition(EXTERNAL_Interface* iface, Node* node, double c1, double c2, double c3)</p> <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	<p>EXTERNAL_ChangeNodeOrientation(EXTERNAL_Interface* iface, Node* node, short azimuth, short elevation)</p> <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	<p>EXTERNAL_ChangeNodePositionAndOrientation(EXTERNAL_Interface* iface, Node* node, double c1, double c2, double c3, short azimuth, short elevation)</p> <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	<p>EXTERNAL_ChangeNodePositionOrientationAndSpeedAtTime(EXTERNAL_Interface* iface, Node* node, clocktype mobilityEventTime, double c1, double c2, double c3, short azimuth, short elevation, double speed)</p> <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	<p>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)</p> <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	<p>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)</p> <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	<p>EXTERNAL_ChangeNodeVelocityAtTime(EXTERNAL_Interface* iface, Node* node, clocktype mobilityEventTime, double speed, double c1Speed, double c2Speed, double c3Speed)</p>

	<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	<p>EXTERNAL_ChangeNodeVelocityAtTime(EXTERNAL_Interface* iface, Node* node, clocktype mobilityEventTime, double c1Speed, double c2Speed, double c3Speed)</p> <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>
BOOL	<p>EXTERNAL_ConfigStringPresent(NodeInput* nodeInput, char* string)</p> <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>
BOOL	<p>EXTERNAL_ConfigStringIsYes(NodeInput* nodeInput, char* string)</p> <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. Checks that the string is YES.</p>
void	<p>EXTERNAL_MESSAGE_RemoteSend(EXTERNAL_Interface* iface, int destinationPartitionId, Message * msg, clocktype delay, ExternalScheduleType scheduling)</p> <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 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 your 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.</p>
void	<p>EXTERNAL_SetSimulationEndTime(partitionData* partitionData, clocktype endTime)</p> <p>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.</p>
clocktype	<p>EXTERNAL_QueryRealTime()</p> <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>
clocktype	<p>EXTERNAL_QueryRealTime()</p> <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>

clocktype	<div><div>EXTERNAL_QueryCPUTime(EXTERNAL_Interface* iface)</div><div>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.</div></div>
void	<div><div>EXTERNAL_Sleep(clocktype amount)</div><div>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.</div></div>
void	<div><div>EXTERNAL_AddHdr(Node* node, Message* sendMessage, int payloadSize, UInt8* src, TosType tos, unsigned char protocol, unsigned ttl)</div><div>This function will create qualnet in6_addr and add ipv6 header to message</div></div>

Constant / Data Structure Detail

Enumeration	<div><div>ExternalScheduleType</div><div>Enumeration of allowed scheduling operations - e.g. EXTERNAL_ActivateNode</div></div>
Structure	<div><div>EXTERNAL_TreeNode</div><div>Structure of each node of a Splaytree</div></div>
Structure	<div><div>EXTERNAL_Tree</div><div>Structure of a Splaytree</div></div>
Structure	<div><div>EXTERNAL_ForwardInstantiate</div><div>Info field used for instantiating a forward app</div></div>
Structure	<div><div>EXTERNAL_ForwardSendUdpData</div><div>Info field used for sending a UDP forward app</div></div>
Structure	<div><div>EXTERNAL_ForwardSendTcpData</div></div>

	Info field used for sending a UDP 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 A table. Generally used for storing external packet data, but can be used for anything.
Structure	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	void EXTERNAL_TreeInitialize (EXTERNAL_Tree* tree, BOOL useStore, int maxStore) Parameters: <ul style="list-style-type: none">tree - Pointer to the splaytreeuseStore - Use StoremaxStore - Max Store Returns: <ul style="list-style-type: none">void - None
SCHED_SplayTreeInsert To insert a node into the Splaytree	void SCHED_SplayTreeInsert (EXTERNAL_Tree* tree, EXTERNAL_TreeNode* treeNode) Parameters: <ul style="list-style-type: none">tree - Pointer to the splaytree

	<ul style="list-style-type: none"><code>treeNode</code> - Pointer to the splayNode <p>Returns:</p> <ul style="list-style-type: none"><code>void</code> - None
<p>EXTERNAL_TreePeekMin</p> <p>To look up a node in the Splaytree</p>	<p>void EXTERNAL_TreePeekMin (EXTERNAL_Tree* tree)</p> <p>Parameters:</p> <ul style="list-style-type: none"><code>tree</code> - Pointer to the splaytree <p>Returns:</p> <ul style="list-style-type: none"><code>void</code> - None
<p>EXTERNAL_InitializeTable</p> <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"><code>table</code> - The table<code>size</code> - The size of the table <p>Returns:</p> <ul style="list-style-type: none"><code>void</code> - None
<p>EXTERNAL_FinalizeTable</p> <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"><code>table</code> - The table <p>Returns:</p> <ul style="list-style-type: none"><code>void</code> - None
<p>EXTERNAL_GetUnusedRecord</p> <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"><code>table</code> - The table <p>Returns:</p> <ul style="list-style-type: none"><code>EXTERNAL_TableRecord*</code> - The retrieved record
<p>EXTERNAL_GetEarliestRecord</p> <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"><code>table</code> - The table

	<div>Returns:</div> <div><ul style="list-style-type: none">EXTERNAL_TableRecord* - The retrieved record</div>
<div>EXTERNAL_GetEarliestRecord</div> <div>This function will check if a data pointer is still in the table.</div>	<div>BOOL EXTERNAL_GetEarliestRecord (EXTERNAL_Table* table, char* data)</div> <div>Parameters:</div> <div><ul style="list-style-type: none">table - The tabledata - The data to check for</div> <div>Returns:</div> <div><ul style="list-style-type: none">BOOL - TRUE if it is in the table, FALSE if not</div>
<div>EXTERNAL_FreeRecord</div> <div>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.</div>	<div>EXTERNAL_TableRecord* EXTERNAL_FreeRecord (EXTERNAL_Table* table)</div> <div>Parameters:</div> <div><ul style="list-style-type: none">table - The table</div> <div>Returns:</div> <div><ul style="list-style-type: none">EXTERNAL_TableRecord* - The retrieved record</div>
<div>EXTERNAL_SendDataAppLayerUDP</div> <div>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.</div>	<div>void EXTERNAL_SendDataAppLayerUDP (EXTERNAL_Interface* iface, NodeAddress from, NodeAddress to, char* data, int dataSize, clocktype timestamp, AppType app, TraceProtocolType trace, TosType priority)</div> <div>Parameters:</div> <div><ul style="list-style-type: none">iface - The external interfacefrom - The address of the sending nodeto - The address of the receiving nodedata - The data that is to be sent. This may be NULL if theredataSize - The size of the datatimestamp - The time to send this message. Pass 0 to sendapp - The application to send to, defaults to APP_FORWARDtrace - The trace protocol, defaults to TRACE_FORWARDpriority - The priority to send this message at</div> <div>Returns:</div> <div><ul style="list-style-type: none">void - None</div>
<div>EXTERNAL_SendDataAppLayerUDP</div>	<div>void EXTERNAL_SendDataAppLayerUDP (EXTERNAL_Interface* iface, NodeAddress from,</div>

<p>Sends virtual data originating from the app layer using UDP. When the packet reaches its destination it will call the forward forward function of the external interface, if it exists.</p>	<p>NodeAddress to, char* header, int headerSize, int virtualDataSize, clocktype timestamp, AppType app, TraceProtocolType trace, TosType priority)</p> <p>Parameters:</p> <ul style="list-style-type: none">iface - The external interfacefrom - The address of the sending nodeto - The address of the receiving nodeheader - The header that is to be sent.headerSize - The size of the headervirtualDataSize - The size of the virtual datatimestamp - The time to send this message. Pass 0 to sendapp - The application to send to, defaults to APP_FORWARDtrace - The trace protocol, defaults to TRACE_FORWARDpriority - The priority to send this message at. defaults to IPTOS_PREC_ROUTINE <p>Returns:</p> <ul style="list-style-type: none">void - None
<p>EXTERNAL_SendDataAppLayerTCP</p> <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>void EXTERNAL_SendDataAppLayerTCP (EXTERNAL_Interface* iface, NodeAddress from, NodeAddress to, char* data, int dataSize, clocktype timestamp)</p> <p>Parameters:</p> <ul style="list-style-type: none">iface - The external interfacefrom - The address of the sending nodeto - The address of the receiving nodedata - The data that is to be sent. This may be NULL if theredataSize - The size of the datatimestamp - The time to send this message. Pass 0 to send <p>Returns:</p> <ul style="list-style-type: none">void - None
<p>EXTERNAL_SendDataNetworkLayer</p> <p>Sends data originating from network layer. No provisions are made</p>	<p>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)</p> <p>Parameters:</p>

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.

- `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
- `payloadSize` - The size of the data
- `timestamp` - The time to send this packet. Pass 0 to send

Returns:

- `void` - None

EXTERNAL_SendDataNetworkLayerOnInterface

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_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`)

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
- `ttl` - The Time to Live field in the IP header
- `payload` - The data that is to be sent. This should include

	<ul style="list-style-type: none">• <code>payloadSize</code> - The size of the data• <code>interfaceIndex</code> - The interface index• <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
<p>EXTERNAL_SendVirtualDataNetworkLayer</p> <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>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> <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• <code>payload</code> - The data that is to be sent. This should include• <code>dataSize</code> - The size of the data• <code>virtualSize</code> - The size of the virtual data• <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
<p>EXTERNAL_SendIpv6DataNetworkLayer</p> <p>Sends ipv6 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>void EXTERNAL_SendIpv6DataNetworkLayer (EXTERNAL_Interface* iface, Address from, Address srcAddr, Address destAddr, TosType tos, unsigned char protocol, unsigned int hlim, char* payload, int payloadSize, clocktype timestamp)</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

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.

- `tos` - The Type of Service field in the IPv6 header
- `protocol` - The protocol field in the IPv6 header
- `hlim` - The hop limit field in the IPv6 header
- `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

Returns:

- `void` - None

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
- `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
- `ipHeaderLength` - length of the IP Header including options if any
- `ipOptions` - pointer to the IP Option.
- `timestamp` - The time to send this packet. Pass 0 to send

	<div>Returns:</div> <div><ul style="list-style-type: none"><code>void</code> - None</div>
<div>EXTERNAL_SendNetworkLayerPacket</div> <div>Sends the packet from EXTERNAL_SendDataNetworkLayer after some delay. This function should never be called directly.</div>	<div>void EXTERNAL_SendNetworkLayerPacket (Node* node, Message* msg)</div> <div>Parameters:</div> <div><ul style="list-style-type: none"><code>node</code> - The node sending the packet<code>msg</code> - The message</div> <div>Returns:</div> <div><ul style="list-style-type: none"><code>void</code> - None</div>
<div>EXTERNAL_CreateMapping</div> <div>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.</div>	<div>void EXTERNAL_CreateMapping (EXTERNAL_Interface* iface, char* key, int keySize, char* value, int valueSize)</div> <div>Parameters:</div> <div><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</div> <div>Returns:</div> <div><ul style="list-style-type: none"><code>void</code> - None</div>
<div>EXTERNAL_ResolveMapping</div> <div>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.</div>	<div>int EXTERNAL_ResolveMapping (EXTERNAL_Interface* iface, char* key, int keySize, char** value, int* valueSize)</div> <div>Parameters:</div> <div><ul style="list-style-type: none"><code>iface</code> - The external interface<code>key</code> - Pointer to the key<code>keySize</code> - The size of the key in bytes<code>value</code> - Pointer to the value (output)<code>valueSize</code> - The size of the key in bytes (output)</div> <div>Returns:</div> <div><ul style="list-style-type: none"><code>int</code> - 0 if the mapping resolved, non-zero if it did not</div>
<div>EXTERNAL_DeleteMapping</div>	<div>int EXTERNAL_DeleteMapping (EXTERNAL_Interface* iface, char* key, int keySize)</div>

<p>Deletes a mapping created by EXTERNAL_CreateMapping.</p>	<p>Parameters:</p> <ul style="list-style-type: none">iface - The external interfacekey - Pointer to the keykeySize - The size of the key in bytes <p>Returns:</p> <ul style="list-style-type: none">int - 0 if the mapping resolved, non-zero if it did not
<p>EXTERNAL_ActivateNode</p> <p>Activate a node so that it can begin processing events.</p>	<p>void EXTERNAL_ActivateNode (EXTERNAL_Interface* iface, Node* node)</p> <p>Parameters:</p> <ul style="list-style-type: none">iface - The external interfacenode - The node <p>Returns:</p> <ul style="list-style-type: none">void - None
<p>EXTERNAL_DectivateNode</p> <p>Dectivate a node so that it stops processing events.</p>	<p>void EXTERNAL_DectivateNode (EXTERNAL_Interface* iface, Node* node)</p> <p>Parameters:</p> <ul style="list-style-type: none">iface - The external interfacenode - The node <p>Returns:</p> <ul style="list-style-type: none">void - None
<p>EXTERNAL_PHY_SetTxPower</p> <p>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.</p>	<p>void EXTERNAL_PHY_SetTxPower (Node* node, int phyIndex, double newTxPower)</p> <p>Parameters:</p> <ul style="list-style-type: none">node - The node (can be either a local node or remote)phyIndex - The physical indexnewTxPower - The new transmission power. <p>Returns:</p> <ul style="list-style-type: none">void - None
<p>EXTERNAL_PHY_GetTxPower</p>	<p>void EXTERNAL_PHY_GetTxPower (Node* node, int phyIndex, double * txPowerPtr)</p> <p>Parameters:</p>

<p>Just like PHY_GetTxPower (), but able to handle getting transmission power when node is owned by a remote partition.</p>	<ul style="list-style-type: none">node - The node (can be either a local node or remote)phyIndex - The physical indextxPowerPtr - (OUT) value of transmission power will be <p>Returns:</p> <ul style="list-style-type: none">void - None
<p>EXTERNAL_ChangeNodePosition</p> <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>void EXTERNAL_ChangeNodePosition (EXTERNAL_Interface* iface, Node* node, double c1, double c2, double c3)</p> <p>Parameters:</p> <ul style="list-style-type: none">iface - The external interfacenode - The nodec1 - The first coordinatec2 - The second coordinatec3 - The third coordinate <p>Returns:</p> <ul style="list-style-type: none">void - None
<p>EXTERNAL_ChangeNodeOrientation</p> <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>void EXTERNAL_ChangeNodeOrientation (EXTERNAL_Interface* iface, Node* node, short azimuth, short elevation)</p> <p>Parameters:</p> <ul style="list-style-type: none">iface - The external interfacenode - The nodeazimuth - The azimuth, 0 <= azimuth <= 359elevation - The elevation, -180 <= elevation <= 180 <p>Returns:</p> <ul style="list-style-type: none">void - None
<p>EXTERNAL_ChangeNodePositionAndOrientation</p> <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">iface - The external interfacenode - The node

	<ul style="list-style-type: none">• <code>c1</code> - The first coordinate• <code>c2</code> - The second coordinate• <code>c3</code> - The third coordinate• <code>azimuth</code> - The azimuth, $0 \leq \text{azimuth} \leq 359$• <code>elevation</code> - The elevation, $-180 \leq \text{elevation} \leq 180$ <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
<p>EXTERNAL_ChangeNodePositionOrientationAndSpeedAtTime</p> <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, $0 \leq \text{azimuth} \leq 359$• <code>elevation</code> - The elevation, $-180 \leq \text{elevation} \leq 180$• <code>speed</code> - The speed in m/s <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
<p>EXTERNAL_ChangeNodePositionOrientationAndVelocityAtTime</p> <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>	<p>void 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)</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

	<ul style="list-style-type: none">• <code>c2</code> - The second coordinate• <code>c3</code> - The third coordinate• <code>azimuth</code> - The azimuth, $0 \leq \text{azimuth} \leq 359$• <code>elevation</code> - The elevation, $-180 \leq \text{elevation} \leq 180$• <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
<p>EXTERNAL_ChangeNodePositionOrientationAndVelocityAtTime</p> <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>	<p>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)</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, $0 \leq \text{azimuth} \leq 359$• <code>elevation</code> - The elevation, $-180 \leq \text{elevation} \leq 180$• <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
<p>EXTERNAL_ChangeNodeVelocityAtTime</p>	<p>void EXTERNAL_ChangeNodeVelocityAtTime (EXTERNAL_Interface* iface, Node* node, clocktype mobilityEventTime, double speed, double c1Speed, double c2Speed, double c3Speed)</p>

<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>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
<p>EXTERNAL_ChangeNodeVelocityAtTime</p> <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</code> EXTERNAL_ChangeNodeVelocityAtTime (<code>EXTERNAL_Interface*</code> <code>iface</code>, <code>Node*</code> <code>node</code>, <code>clocktype</code> <code>mobilityEventTime</code>, <code>double</code> <code>c1Speed</code>, <code>double</code> <code>c2Speed</code>, <code>double</code> <code>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
<p>EXTERNAL_ConfigStringPresent</p> <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>	<p><code>BOOL</code> EXTERNAL_ConfigStringPresent (<code>NodeInput*</code> <code>nodeInput</code>, <code>char*</code> <code>string</code>)</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
<p>EXTERNAL_ConfigStringIsYes</p>	<p><code>BOOL</code> EXTERNAL_ConfigStringIsYes (<code>NodeInput*</code> <code>nodeInput</code>, <code>char*</code> <code>string</code>)</p>

<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. Checks that the string is YES.</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 YES, FALSE otherwise
<p>EXTERNAL_MESSAGE_RemoteSend</p> <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 your 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>	<p>void EXTERNAL_MESSAGE_RemoteSend (EXTERNAL_Interface* iface, int destinationPartitionId, Message * msg, clocktype delay, ExternalScheduleType scheduling)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>iface</code> - Your external interface• <code>destinationPartitionId</code> - The partitionId that you want to send to• <code>msg</code> - The external message to send• <code>delay</code> - When the message should be scheduled on the remote partion.• <code>scheduling</code> - Whether this event can be executed lossely <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
<p>EXTERNAL_SetSimulationEndTime</p> <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
<p>EXTERNAL_QueryRealTime</p> <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>	<p>clocktype EXTERNAL_QueryRealTime ()</p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none">• <code>clocktype</code> - The real time, not adjusted for simulation pauses.
<p>EXTERNAL_QueryRealTime</p>	<p>clocktype EXTERNAL_QueryRealTime ()</p> <p>Parameters:</p>

<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>	<p>Returns:</p> <ul style="list-style-type: none"><code>clocktype</code> - The real time, adjusted for simulation pauses.
<p>EXTERNAL_QueryCPUTime</p> <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>	<p><code>clocktype</code> EXTERNAL_QueryCPUTime (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 CPU time
<p>EXTERNAL_Sleep</p> <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><code>void</code> EXTERNAL_Sleep (clocktype amount)</p> <p>Parameters:</p> <ul style="list-style-type: none"><code>amount</code> - The amount of time to sleep <p>Returns:</p> <ul style="list-style-type: none"><code>void</code> - None
<p>EXTERNAL_AddHdr</p> <p>This function will create qualnet in6_addr and add ipv6 header to message</p>	<p><code>void</code> EXTERNAL_AddHdr (Node* node, Message* sendMessage, int payloadSize, UInt8* src, TosType tos, unsigned char protocol, unsigned ttl)</p> <p>Parameters:</p> <ul style="list-style-type: none"><code>node</code> - Node pointer<code>sendMessage</code> - Message on which ipv6 header needs to be added<code>payloadSize</code> - Size of payload in packet<code>src</code> - IPv6 Source address<code>tos</code> - Packet Priority<code>protocol</code> - Protcol after ipv6 header<code>ttl</code> - Hop limit <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 © 2001-2013 [SCALABLE Network Technologies, Inc.](#) All rights reserved.

QualNet 7.1 API Reference

FILEIO

This file describes data strucutres 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_GLOBAL Defines the matching at global level
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
	Defines input allocation unit.
STRUCT	NodeInput
	Definition of node input structure. typedef to NodeInput in include/main.h.

Function / Macro Summary

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

void	<div><div>IO_TrimNsbpsSpaces(char* s)</div><div>Changes nsbp charecters for UTF-8 encoding to spaces.</div></div>
void	<div><div>IO_TrimLeft(char* s)</div><div>Strips leading white space from a string (by memmove()ing string contents left).</div></div>
void	<div><div>IO_TrimRight(char* s)</div><div>Strips trailing white space from a string (by inserting early NULL).</div></div>
void	<div><div>IO_CompressWhiteSpace(char* s)</div><div>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.</div></div>
BOOL	<div><div>IO_IsStringNonNegativeInteger(const char* s)</div><div>Returns TRUE if every character in string is a digit. (Even white space will cause return of FALSE)</div></div>
void	<div><div>IO_ConvertStringToLowerCase(char s[])</div><div>Runs tolower() on each character in string and converts the same to lowercase.</div></div>
void	<div><div>IO_ConvertStringToUpperCase(char s[])</div><div>Runs toupper() on each character in string and converts the same to uppercase.</div></div>
BOOL	<div><div>IO_CaseInsensitiveStringsAreEqual(const char[] s1, const char[] s2, char lengthToCompare)</div><div>Checks two strings are equal or not ignoring case.</div></div>
BOOL	<div><div>IO_BlankLine(char s[])</div><div>Checks the blank line/string.</div></div>
BOOL	<div><div>IO_CommentLine(char s[])</div><div>Checks whether the line is a comment(i.e. starts with '#').</div></div>
int	<div><div>IO_FindCaseInsensitiveStringPos(const char s[], const char subString[])</div><div>Finds the case insensitive sub string position in a string.</div></div>

int	IO_FindCaseInsensitiveStringPos (const char s[], const char subString[]) Finds the case insensitive sub string position in a string.
	IO_SkipToken. (char* token, char* tokenSep, char* skip) skip the first n tokens.
NodeInput *	IO_CreateNodeInput (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	IO_InitializeNodeInput (NodeInput* nodeInput) Initializes a NodeInput structure
void	IO_ReadNodeInput (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	IO_ReadNodeInputEx (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	IO_ConvertFile (NodeInput* nodeInput, NodeInput* nodeOutput, char* version) Converts the contents of an old configuration file to the latest version.
void	IO_ReadLine (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	IO_ReadString (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	IO_ReadBool (const NodeAddress nodeId, const NodeAddress interfaceAddress, const NodeInput* nodeInput, const char* index, BOOL* wasFound, BOOL* readVal)

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

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

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

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

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

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

	<p>Parses IPv6 address into a formatted string. Overloaded API for Ipv6 compatibility.</p>
void	<p>IO_ConvertIpAddressToString(Address* ipAddress, char* interfaceAddr)</p>
	<p>Parses IPv6 address into a formatted string. Overloaded API for Ipv6 compatibility.</p>
void	<p>IO_ConvertStringToNodeAddress(char* addressString, NodeAddress* outputNodeAddress)</p>
	<p>This API is used to covert a string parameter to NodeAddress.</p>
BOOL	<p>IO_CheckIsSameAddress(Address addr1, Address addr2)</p>
	<p>Compares IPv4 IPv6 address. API for Ipv6 compatibility.</p>
void	<p>IO_ReadString(Node* node node, const NodeInput* nodeInput, const char* index, BOOL* wasFound, char* readVal)</p>
	<p>This API is used to retrieve a string parameter value from input files.</p>
void	<p>IO_CacheFile(const NodeInput* nodeInput, const char* filename)</p>
	<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>
int	<p>IO_GetMaxLen(fileName char*)</p>
	<p>This API is used to get the maximun length of a line in the file.</p>
int	<p>IO_GetMaxLen(fp FILE*)</p>
	<p>This API is used to get the maximum length of a line in the file.</p>
int	<p>NI_GetMaxLen(nodeInput NodeInput*)</p>
	<p>This API is used to get the maximum length of a line in nodeInput.</p>
int	<p>NI_GetMaxLen(nodeInput const NodeInput*)</p>
	<p>This API is used to get the maximum length of a line in nodeInput.</p>
void	<p>IO_ParseNetworkAddress(const char s[], unsigned int* u_atmVal)</p>
	<p>Parses a string for a network address. Overloaded API for ATM compatibility.</p>
void	<p>IO_ConvertAddrToString(Address* address, char* addrStr)</p>

	Convert generic address to appropriate network type address string format.
void	IO_ConvertAtmAddressToString (AtmAddress addr, char* addrStr) Convert Atm address to address string format.
void	IO_InsertIntValue (const char s[], const unsigned int val, unsigned int u_atmVal) Insert integer value for specific string in case of ATM
int	IO_ReadCachedFileIndex (NodeAddress nodeId, NodeAddress interfaceAddress, unsigned int nodeInput) Return Cached file index for the given parameter name
void	IO_ReadString (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	IO_ReadInt64 (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	IO_ReadTime (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	IO_ReadInt (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	IO_ReadDouble (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	IO_ReadCachedFile (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	IO_ReadLine (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	<div><div>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)</div><div>This API is used to retrieve string parameter values from input files for a specific instance.</div></div>
void	<div><div>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)</div><div>This API is used to retrieve double parameter values from input files for a specific instance.</div></div>
void	<div><div>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)</div><div>This API is used to retrieve int parameter values from input files for a specific instance.</div></div>
void	<div><div>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)</div><div>This API is used to retrieve clocktype parameter values from input files for a specific instance.</div></div>
void	<div><div>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)</div><div>This API is used to retrieve cached file parameter values from input files for a specific instance.</div></div>
void	<div><div>IO_ReadStringUsingIpAddress(Node* node, int interfaceIndex, const NodeInput* nodeInput, const char* parameterName, BOOL* wasFound, char* parameterValue)</div><div>This API is used to retrieve a string parameter value from input files using the ip-address.</div></div>
void	<div><div>IO_ReadString(const NodeAddress nodeId, NodeAddress ipv4Address, in6_addr* ipv6Address, const NodeInput* nodeInput, const char* parameterName, BOOL* wasFound, char* parameterValue)</div><div>This API is used to retrieve a string parameter value from input files.</div></div>
void	<div><div>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)</div><div>This API is used to retrieve a string parameter value from input files.</div></div>
void	<div><div>IO_ReadChannelMask(Node* node, const NodeAddress nodeId, const Address* address, int interfaceIndex, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, BOOL* parseChannelList, char* parameterValue)</div><div>This API is used to retrieve the value channel mask for a specific instance.</div></div>

void	<div><div><div>IO_ReadBool(Node* node, const NodeId nodeId, int interfaceIndex, const NodeInput* nodeInput, const char* parameterName, BOOL* wasFound, BOOL* parameterValue)</div></div></div> <div>This API is used to retrieve a boolean parameter value from input files.</div>
void	<div><div><div>IO_ReadFloat(Node* node, const NodeAddress nodeId, int interfaceIndex, const NodeInput* nodeInput, const char* parameterName, BOOL* wasFound, float* parameterValue)</div></div></div> <div>This API is used to retrieve a float parameter value from input files.</div>

Constant / Data Structure Detail

Constant	<div><div><div>ANY_NODEID 0xffffffff</div></div></div> <div>Optional macro values to use when calling IO_Read...() APIs. Defines any node id.</div>
Constant	<div><div><div>ANY_ADDRESS 0xffffffff</div></div></div> <div>Optional macro values to use when calling IO_Read...() APIs. Defines any node address.</div>
Constant	<div><div><div>ANY_INSTANCE 0xffffffff</div></div></div> <div>Optional macro values to use when calling IO_Read...() APIs. Defines any instance.</div>
Constant	<div><div><div>MAX_INPUT_FILE_LINE_LENGTH 6 * MAX_STRING_LENGTH</div></div></div> <div>Maximum input file line length. Evaluates (6 * MAX_STRING_LENGTH)</div>
Constant	<div><div><div>MAX_ADDRESS_STRING_LENGTH 80</div></div></div> <div>Maximum length of address string.</div>
Constant	<div><div><div>MAX_NUM_CACHED_FILES 128</div></div></div> <div>Max number of -FILE references in an input file. (Restriction is only for immediate children)</div>
Constant	<div><div><div>MATCH_GLOBAL 2</div></div></div> <div>Defines the matching at global level</div>

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.	void IO_ConvertIpAddressToString (NodeAddress ipAddress, char* addressString) Parameters: <ul style="list-style-type: none">ipAddress - IPv4 address to be converted intoaddressString - Storage for string. Returns: <ul style="list-style-type: none">void - None
IO_FindStringPos Returns the index of the first subString found in s.	int IO_FindStringPos (const char s[], const char subString[]) Parameters: <ul style="list-style-type: none">s[] - Source string.subString[] - Substring to earch for. Returns:

	<ul style="list-style-type: none">• <code>int</code> - Index of the first subString found in <code>s</code>. -1, if not found.
IO_GetToken Searches source buffer for the first %s-style token encountered, and copies it to <code>dst</code> .	<p><code>char* IO_GetToken (char* <code>dst</code>, const char* <code>src</code>, char ** <code>next</code>)</code></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 <code>dst</code> .	<p><code>char* IO_GetDelimitedToken (char* <code>dst</code>, const char* <code>src</code>, const char* <code>delim</code>, char** <code>next</code>)</code></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.• <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_Right Returns a pointer to the right side of the string of length "count" characters.	<p><code>const char* IO_Right (const char * <code>s</code>, unsigned <code>count</code>)</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 count is 0, then a pointer to the string's terminating NULL is returned. If count 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 char type that's not NULL. So, '\n' counts as a character.
IO_Chop Removes the last character of string.	<p><code>char* IO_Chop (const char* <code>s</code>)</code></p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>s</code> - String. <p>Returns:</p>

	<ul style="list-style-type: none">char* - s. If the string has a strlen() of zero, then the string is returned unmodified. A "character" is just a byte of char type that's not NULL. So, '\n' counts as a character.
IO_TrimNsbpSpaces Changes nsbp charecters for UTF-8 encoding to spaces.	void IO_TrimNsbpSpaces (char* s) Parameters: <ul style="list-style-type: none">s - String. Returns: <ul style="list-style-type: none">void - None
IO_TrimLeft Strips leading white space from a string (by memmove()ing string contents left).	void IO_TrimLeft (char* s) Parameters: <ul style="list-style-type: none">s - String. Returns: <ul style="list-style-type: none">void - None
IO_TrimRight Strips trailing white space from a string (by inserting early NULL).	void IO_TrimRight (char* s) Parameters: <ul style="list-style-type: none">s - String. Returns: <ul style="list-style-type: none">void - None
IO_CompressWhiteSpace 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.	void IO_CompressWhiteSpace (char* s) Parameters: <ul style="list-style-type: none">s - String. Returns: <ul style="list-style-type: none">void - None
IO_IsStringNonNegativeInteger Returns TRUE if every character in string is a digit. (Even white space will cause return of FALSE)	BOOL IO_IsStringNonNegativeInteger (const char* s) Parameters: <ul style="list-style-type: none">s - String. Returns: <ul style="list-style-type: none">BOOL - TRUE if every character is a digit. FALSE, otherwise.

IO_ConvertStringToLowerCase Runs tolower() on each character in string and converts the same to lowercase.	void IO_ConvertStringToLowerCase (char s[]) Parameters: <ul style="list-style-type: none">s[] - String. Returns: <ul style="list-style-type: none">void - None
IO_ConvertStringToUpperCase Runs toupper() on each character in string and converts the same to uppercase.	void IO_ConvertStringToUpperCase (char s[]) Parameters: <ul style="list-style-type: none">s[] - String. Returns: <ul style="list-style-type: none">void - None
IO_CaseInsensitiveStringsAreEqual Checks two strings are equal or not ignoring case.	BOOL IO_CaseInsensitiveStringsAreEqual (const char[] s1, const char[] s2, char lengthToCompare) Parameters: <ul style="list-style-type: none">s1 - First string.s2 - Second string.lengthToCompare - Length to compare. Returns: <ul style="list-style-type: none">BOOL - Returns TRUE if strings are equal, FALSE otherwise.
IO_BlankLine Checks the blank line/string.	BOOL IO_BlankLine (char s[]) Parameters: <ul style="list-style-type: none">s[] - String. Returns: <ul style="list-style-type: none">BOOL - Returns TRUE if the string is blank. FALSE, otherwise.
IO_CommentLine Checks whether the line is a comment(i.e. starts with '#').	BOOL IO_CommentLine (char s[]) Parameters: <ul style="list-style-type: none">s[] - String. Returns: <ul style="list-style-type: none">BOOL - Returns TRUE if the line is a comment. FALSE, otherwise.
IO_FindCaseInsensitiveStringPos	int IO_FindCaseInsensitiveStringPos (const char s[], const char subString[])

<p>Finds the case insensitive sub string position in a string.</p>	<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.
<p>IO_FindCaseInsensitiveStringPos</p> <p>Finds the case insensitive sub string position in a string.</p>	<p><code>int IO_FindCaseInsensitiveStringPos</code> (const char s[], const char subString[])</p> <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.
<p>IO_SkipToken.</p> <p>skip the first n tokens.</p>	<p>IO_SkipToken. (char* token, char* tokenSep, char* skip)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>token</code> - pointer to the input string,• <code>tokenSep</code> - pointer to the token separators,• <code>skip</code> - 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</code> (NodeInput* nodeInput, const char* filename)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>nodeInput</code> - Pointer to node input.• <code>filename</code> - Path to input file. <p>Returns:</p> <ul style="list-style-type: none">• <code>NodeInput *</code> - None
<p>IO_InitializeNodeInput</p> <p>Initializes a NodeInput structure</p>	<p><code>void IO_InitializeNodeInput</code> (NodeInput* nodeInput)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>nodeInput</code> - A pointer to NodeInput structure.

	<div>Returns:</div> <div><ul style="list-style-type: none">void - None</div>
<div><div>IO_ReadNodeInput</div><div>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.</div></div>	<div><div>void IO_ReadNodeInput (NodeInput* nodeInput, const char* filename)</div><div>Parameters:</div><div><ul style="list-style-type: none">nodeInput - Pointer to node input.filename - Path to input file.</div><div>Returns:</div><div><ul style="list-style-type: none">void - None</div></div>
<div><div>IO_ReadNodeInputEx</div><div>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.</div></div>	<div><div>void IO_ReadNodeInputEx (NodeInput* nodeInput, const char* filename, const char* includeComment)</div><div>Parameters:</div><div><ul style="list-style-type: none">nodeInput - Pointer to node input.filename - Path to input file.includeComment - When this flag is true it</div><div>Returns:</div><div><ul style="list-style-type: none">void - None</div></div>
<div><div>IO_ConvertFile</div><div>Converts the contents of an old configuration file to the latest version.</div></div>	<div><div>BOOL IO_ConvertFile (NodeInput* nodeInput, NodeInput* nodeOutput, char* version)</div><div>Parameters:</div><div><ul style="list-style-type: none">nodeInput - A pointer to node input.nodeOutput - A pointer to node input. Goes throughversion - Not used.</div><div>Returns:</div><div><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.</div></div>
<div><div>IO_ReadLine</div><div>This API is used to retrieve a whole line from input files.</div></div>	<div><div>void IO_ReadLine (const NodeAddress nodeId, const NodeAddress interfaceAddress, const NodeInput* nodeInput, const char* index, BOOL* wasFound, char* readVal)</div><div>Parameters:</div><div><ul style="list-style-type: none">nodeId - nodeId. Can be ANY_NODEID.</div></div>

	<ul style="list-style-type: none">• <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
<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">• <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
<p>IO_ReadBool</p> <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">• <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> <p><code>void</code></p>

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

	<ul style="list-style-type: none">• <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_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">• <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
<p>IO_ReadCachedFile</p> <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">• <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>parameterValue</code> - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
<p>IO_ReadStringInstance</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>This API is used to retrieve string parameter values from input files for a specific instance.</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 seach.• <code>parameterValue</code> - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
<p>IO_ReadBoolInstance</p> <p>This API is used to retrieve boolean parameter values from input files for a specific instance.</p>	<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>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 seach.• <code>parameterValue</code> - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - 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>

	<div>nodeId - nodeId. Can be ANY_NODEID.</div> <div><ul style="list-style-type: none">interfaceAddress - IP address of interface.nodeInput - Pointer to node input.parameterName - Parameter name.parameterInstanceNumber - Instance number.fallbackIfNoInstanceMatch - Selects parameter withoutwasFound - Storage for success of seach.parameterValue - Storage for parameter value.</div> <div>Returns:<div><ul style="list-style-type: none">void - None</div></div>
<div>IO_ReadDoubleInstance</div> <div>This API is used to retrieve double parameter values from input files for a specific instance.</div>	<div>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)</div> <div>Parameters:<div><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 withoutwasFound - Storage for success of seach.parameterValue - Storage for parameter value.</div></div> <div>Returns:<div><ul style="list-style-type: none">void - None</div></div>
<div>IO_ReadFloatInstance</div> <div>This API is used to retrieve float parameter values from input files for a specific instance.</div>	<div>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)</div> <div>Parameters:<div><ul style="list-style-type: none">nodeId - nodeId. Can be ANY_NODEID.interfaceAddress - IP address of interface.</div></div>

	<ul style="list-style-type: none">• <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
<p>IO_ReadTimeInstance</p> <p>This API is used to retrieve time parameter values from input files for a specific instance.</p>	<p>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)</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 seach.• <code>parameterValue</code> - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
<p>IO_ReadCachedFileInstance</p> <p>This API is used to retrieve file parameter values from input files for a specific instance.</p>	<p>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)</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. <p><code>parameterName</code></p>

	<div>- Parameter name.</div> <div><ul style="list-style-type: none">parameterInstanceNumber - Instance number.fallbackIfNoInstanceMatch - Selects parameter withoutwasFound - Storage for success of seach.parameterValue - Storage for parameter value.</div> <div>Returns:</div> <div><ul style="list-style-type: none">void - None</div>
<div>IO_ParseNodeIdHostOrNetworkAddress</div> <div>Parses a string for a nodeId, host address, or network address.</div>	<div>void IO_ParseNodeIdHostOrNetworkAddress (const char s[], NodeAddress* outputNodeAddress, int* numHostBits, BOOL* isNodeId)</div> <div>Parameters:</div> <div><ul style="list-style-type: none">s[] - String to parse.outputNodeAddress - Storage for nodeId or IP address.numHostBits - Storage for number of host bitsisNodeId - Storage for whether the string is</div> <div>Returns:</div> <div><ul style="list-style-type: none">void - None</div>
<div>IO_ParseNodeIdOrHostAddress</div> <div>Parses a string for a nodeId or host address.</div>	<div>void IO_ParseNodeIdOrHostAddress (const char s[], NodeAddress* outputNodeAddress, BOOL* isNodeId)</div> <div>Parameters:</div> <div><ul style="list-style-type: none">s[] - String to parse.outputNodeAddress - Storage for nodeId or IP address.isNodeId - Storage for whether the string is</div> <div>Returns:</div> <div><ul style="list-style-type: none">void - None</div>
<div>IO_ParseNetworkAddress</div> <div>Parses a string for a network address.</div>	<div>void IO_ParseNetworkAddress (const char s[], NodeAddress* outputNodeAddress, int* numHostBits)</div> <div>Parameters:</div> <div><ul style="list-style-type: none">s[] - String to parse.outputNodeAddress - Storage for network address.numHostBits - Storage for number of host bits</div>

	<div>Returns:</div> <div><ul style="list-style-type: none">void - None</div>
<div>IO_FreeNodeInput</div> <div>Frees a NodeInput struct. (Currently unused.)</div>	<div>void IO_FreeNodeInput (NodeInput* nodeInput)</div> <div>Parameters:</div> <div><ul style="list-style-type: none">nodeInput - Pointer to node input.</div> <div>Returns:</div> <div><ul style="list-style-type: none">void - None</div>
<div>IO_PrintStat</div> <div>Print out the relevant stat in "buf", along with the node id and the layer type generating this stat.</div>	<div>void IO_PrintStat (Node* node, const char* layer, const char* protocol, NodeAddress interfaceAddress, int instanceId, const char* buf)</div> <div>Parameters:</div> <div><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.instanceId - Instance id.buf - String which has the statistic to</div> <div>Returns:</div> <div><ul style="list-style-type: none">void - None</div>
<div>IO_AppParseSourceAndDestStrings</div> <div>Application input parsing API. Parses the source and destination strings.</div>	<div>void IO_AppParseSourceAndDestStrings (Node* node, const char* inputString, const char* sourceString, NodeAddress* sourceNodeId, NodeAddress* sourceAddr, const char* destString, NodeAddress* destNodeId, NodeAddress* destAddr)</div> <div>Parameters:</div> <div><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 - Const char pointer.destNodeId - A pointer to NodeAddress.</div>

	<ul style="list-style-type: none">destAddr - A pointer to NodeAddress. <p>Returns:</p> <ul style="list-style-type: none">void - None
<p>IO_AppParseSourceString</p> <p>Application input parsing API. Parses the source string.</p>	<p>void IO_AppParseSourceString (Node* node, const char* inputString, const char* sourceString, NodeAddress* sourceNodeId, NodeAddress* sourceAddr)</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. <p>Returns:</p> <ul style="list-style-type: none">void - None
<p>IO_AppParseDestString</p> <p>Application input parsing API. Parses the destination string.</p>	<p>void IO_AppParseDestString (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_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.

	<ul style="list-style-type: none">• <code>destNodeId</code> - A pointer to <code>NodeAddress</code>.• <code>destAddr</code> - A pointer to <code>NodeAddress</code>. <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
<p>IO_AppForbidSameSourceAndDest</p> <p>Application input checking API. Checks for the same source and destination node id. Calls <code>abort()</code> for same source and destination.</p>	<p>void IO_AppForbidSameSourceAndDest (<code>const char*</code> <code>inputString</code>, <code>NodeAddress</code> <code>sourceNodeId</code>, <code>NodeAddress</code> <code>destNodeId</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>inputString</code> - The input string.• <code>sourceNodeId</code> - Source node id, read from the• <code>destNodeId</code> - Destination node id, read from the <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
<p>QualifierMatches</p> <p>This is an auxiliary API used by the <code>IO_Read...</code>() set of APIs.</p>	<p>BOOL QualifierMatches (<code>const NodeAddress</code> <code>nodeId</code>, <code>const NodeAddress</code> <code>interfaceAddress</code>, <code>const char*</code> <code>qualifier</code>, <code>int*</code> <code>matchType</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>nodeId</code> - <code>nodeId</code> 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 <code>TRUE</code> if match found. <code>FALSE</code>, 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 (<code>const NodeAddress</code> <code>nodeId</code>, <code>const in6_addr*</code> <code>interfaceAddress</code>, <code>const NodeInput*</code> <code>nodeInput</code>, <code>const char*</code> <code>parameterName</code>, <code>BOOL*</code> <code>wasFound</code>, <code>BOOL*</code> <code>parameterValue</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>nodeId</code> - <code>nodeId</code>. Can be <code>ANY_NODEID</code>.• <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.

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

<div><div>IO_ReadString</div><div>Reads string value for specified ATM address.</div></div>	<div><div>void IO_ReadString (const NodeId nodeId, const AtmAddress* interfaceAddress, const NodeInput * nodeInput, const char * parameterName, BOOL * wasFound, char * parameterValue)</div><div>Parameters:<ul style="list-style-type: none">nodeId - NodeId for which parameter hasinterfaceAddress - ATM Interface addressnodeInput - pointer to configuration inputsparameterName - Parameter to be readwasFound - Parameter found or notparameterValue - Parameter's value if found.</div><div>Returns:<ul style="list-style-type: none">void - None</div></div>
<div><div>IO_ReadString</div><div>This API is used to retrieve a string parameter value from input files. Overloaded API for Ipv6 compatibility.</div></div>	<div><div>void IO_ReadString (const NodeId nodeId, const Address* address, const NodeInput* nodeInput, const char* index, BOOL* wasFound, char* readVal)</div><div>Parameters:<ul style="list-style-type: none">nodeId - nodeId. Can be ANY_NODEID.address - IP address of interface.nodeInput - Pointer to node input.index - Parameter name.wasFound - Storage for success of seach.readVal - Storage for parameter value.</div><div>Returns:<ul style="list-style-type: none">void - None</div></div>
<div><div>IO_ReadInt</div><div>This API is used to retrieve an integer parameter value from input files. Overloaded API for Ipv6 compatibility.</div></div>	<div><div>void IO_ReadInt (const NodeId nodeId, const in6_addr* interfaceAddress, const NodeInput* nodeInput, const char* index, BOOL* wasFound, int* readVal)</div><div>Parameters:<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.</div></div>

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

	<ul style="list-style-type: none">void - None
IO_ReadDouble Reads double value for specified ATM address.	None IO_ReadDouble () Parameters: Returns: <ul style="list-style-type: none">None - None
IO_ReadDouble This API is used to retrieve a double parameter value from input files. Overloaded API for Ipv6 compatibility.	void IO_ReadDouble (const NodeId nodeId, const Address* address, const NodeInput* nodeInput, const char* index, BOOL* wasFound, double* readVal) Parameters: <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. Returns: <ul style="list-style-type: none">void - None
IO_ReadFloat This API is used to retrieve a float 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) Parameters: <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. Returns: <ul style="list-style-type: none">void - None
IO_ReadFloat	void IO_ReadFloat ()

Reads float value for specified ATM address.	<p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none">void - None NOTE: Overloaded API IO_ReadFloat()
<p>IO_ReadFloat</p> <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
<p>IO_ReadTime</p> <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
<p>IO_ReadTime</p> <p>Reads time value for specified ATM address.</p>	<p>void IO_ReadTime ()</p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none">void - None

<div><div><div>IO_ReadTime</div></div><div><div>This API is used to retrieve time parameter value from input files. Overloaded API for Ipv6 compatibility.</div></div></div>	<div><div><div>void IO_ReadTime (const NodeId nodeId, const Address* address, const NodeInput* nodeInput, const char* index, BOOL* wasFound, clocktype* readVal)</div></div><div><div>Parameters:</div><div><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.</div><div><div>Returns:</div><div><ul style="list-style-type: none">void - None</div></div></div></div>
<div><div><div>IO_ReadBoolInstance</div></div><div><div>Reads BOOL value for specified ATM address.</div></div></div>	<div><div><div>None IO_ReadBoolInstance ()</div></div><div><div>Parameters:</div></div><div><div>Returns:</div><div><ul style="list-style-type: none">None - None</div></div></div>
<div><div><div>IO_ReadStringInstance</div></div><div><div>This API is used to retrieve string parameter values from input files for a specific instance. Overloaded API for Ipv6 compatibility.</div></div></div>	<div><div><div>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)</div></div><div><div>Parameters:</div><div><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 withoutwasFound - Storage for success of seach.parameterValue - Storage for parameter value.</div><div><div>Returns:</div><div><ul style="list-style-type: none">void - None</div></div></div></div>

IO_ReadStringInstance	<div>void IO_ReadStringInstance ()</div> <div>Parameters:</div> <div>Returns:</div> <div><ul style="list-style-type: none">void - None NOTE: Overloaded API IO_ReadStringInstance()</div>
IO_ReadStringInstance This API is used to retrieve string parameter values from input files for a specific instance. Overloaded API for Ipv6 compatibility.	<div>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)</div> <div>Parameters:</div> <div><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 withoutwasFound - Storage for success of seach.parameterValue - Storage for parameter value.</div> <div>Returns:</div> <div><ul style="list-style-type: none">void - None</div>
IO_ReadIntInstance This API is used to retrieve integer parameter values from input files for a specific instance. Overloaded API for Ipv6 compatibility.	<div>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)</div> <div>Parameters:</div> <div><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 withoutwasFound - Storage for success of seach.</div>

	<div>parameterValue - Storage for parameter value.</div> <div>Returns:</div> <div><ul style="list-style-type: none">void - None</div>
<div>IO_ReadIntInstance</div> <div>This API is used to retrieve integer parameter values from input files for a specific instance. Overloaded API for Ipv6 compatibility.</div>	<div>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)</div> <div>Parameters:</div> <div><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 withoutwasFound - Storage for success of seach.parameterValue - Storage for parameter value.</div> <div>Returns:</div> <div><ul style="list-style-type: none">void - None</div>
<div>IO_ReadDoubleInstance</div> <div>This API is used to retrieve double parameter values from input files for a specific instance. Overloaded API for Ipv6 compatibility.</div>	<div>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)</div> <div>Parameters:</div> <div><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 withoutwasFound - Storage for success of seach.parameterValue - Storage for parameter value.</div> <div>Returns:</div>

	<ul style="list-style-type: none">void - None
IO_ReadDoubleInstance Reads double value for specified ATM address.	<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 This API is used to retrieve double parameter values from input files for a specific instance. Overloaded API for Ipv6 compatibility.	<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.fallbackIfNoInstanceMatch - Selects parameter withoutwasFound - Storage for success of seach.parameterValue - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none">void - None
IO_ReadFloatInstance This API is used to retrieve float parameter values from input files for a specific instance. Overloaded API for Ipv6 compatibility.	<p>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)</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

	<div>wasFound - Storage for success of seach.</div> <div><ul style="list-style-type: none">parameterValue - Storage for parameter value.</div> <div>Returns:</div> <div><ul style="list-style-type: none">void - None</div>
<div>IO_ReadFloatInstance</div> <div>This API is used to retrieve float parameter values from input files for a specific instance. Overloaded API for Ipv6 compatibility.</div>	<div>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)</div> <div>Parameters:</div> <div><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 withoutwasFound - Storage for success of seach.parameterValue - Storage for parameter value.</div> <div>Returns:</div> <div><ul style="list-style-type: none">void - None</div>
<div>IO_ReadTimeInstance</div> <div>This API is used to retrieve time parameter values from input files for a specific instance. Overloaded API for Ipv6 compatibility.</div>	<div>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)</div> <div>Parameters:</div> <div><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 withoutwasFound - Storage for success of seach.parameterValue - Storage for parameter value.</div>

	<div>Returns:</div> <div><ul style="list-style-type: none"><code>void</code> - None</div>
<div>IO_ReadTimeInstance</div> <div>Reads clocktype value for specified ATM address.</div>	<div>void IO_ReadTimeInstance ()</div> <div>Parameters:</div> <div>Returns:</div> <div><ul style="list-style-type: none"><code>void</code> - None NOTE: Overloaded API IO_ReadTimeInstance()</div>
<div>IO_ReadTimeInstance</div> <div>This API is used to retrieve time parameter values from input files for a specific instance. Overloaded API for Ipv6 compatibility.</div>	<div>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)</div> <div>Parameters:</div> <div><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.</div> <div>Returns:</div> <div><ul style="list-style-type: none"><code>void</code> - None</div>
<div>IO_ReadCachedFile</div> <div>This API is used to retrieve cached file parameter value from input files. Overloaded API for Ipv6 compatibility.</div>	<div>void IO_ReadCachedFile (const NodeAddress nodeId, const Address* address, const NodeInput* nodeInput, const char* index, BOOL* wasFound, NodeInput* parameterValue)</div> <div>Parameters:</div> <div><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>parameterValue</code> - Storage for parameter value.</div>

	<div>Returns:</div> <div><ul style="list-style-type: none"><code>void</code> - None</div>
<div><div>IO_ReadCachedFileInstance</div><div>This API is used to retrieve file parameter values from input files for a specific instance. Overloaded API for Ipv6 compatibility.</div></div>	<div><div>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)</div><div>Parameters:</div><div><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.</div><div>Returns:</div><div><ul style="list-style-type: none"><code>void</code> - None</div></div>
<div><div>IO_PrintStat</div><div>Print out the relevant stat in "buf", along with the node id and the layer type generating this stat. Overloaded API for Ipv6 compatibility.</div></div>	<div><div>void IO_PrintStat (Node* node, const char* layer, const char* protocol, const char* interfaceAddress, int instanceId, const char* buf)</div><div>Parameters:</div><div><ul style="list-style-type: none"><code>node</code> - The node generating the stat.<code>layer</code> - The layer generating the stat.<code>protocol</code> - The protocol generating the stat.<code>interfaceAddress</code> - The Interface address the stat.<code>instanceId</code> - Instance id.<code>buf</code> - String which has the statistic to</div><div>Returns:</div><div><ul style="list-style-type: none"><code>void</code> - None</div></div>
<div><div>IO_ParseNodeIdHostOrNetworkAddress</div></div>	<div><div>void IO_ParseNodeIdHostOrNetworkAddress (const char s[], in6_addr* ipAddress, BOOL* isIpAddr, NodeId* nodeId)</div><div>Parameters:</div></div>

<p>Parses a string for a nodeId, host address, or network address. Overloaded API for Ipv6 compatibility.</p>	<ul style="list-style-type: none">• <code>s[]</code> - String to parse.• <code>ipAddress</code> - Storage for ipv6address.• <code>isIpAddr</code> - Storage for whether the string is• <code>nodeId</code> - Storage for nodeId. <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
<p>IO_ParseNodeIdHostOrNetworkAddress</p> <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">• <code>s[]</code> - String to parse.• <code>atmAddress</code> - Storage for ATMAddress.• <code>isAtmAddr</code> - Storage for whether the string is• <code>nodeId</code> - Storage for nodeId. <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
<p>IO_ParseNodeIdOrHostAddress</p> <p>Parses a string for a nodeId or host address.</p>	<p>void IO_ParseNodeIdOrHostAddress (const char s[], in6_addr* outputNodeAddress, BOOL* isNodeId)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>s[]</code> - String to parse.• <code>outputNodeAddress</code> - Storage for ipv6address.• <code>isNodeId</code> - Storage for whether the string is <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
<p>IO_ParseNetworkAddress</p> <p>Parses a string for a network address. Overloaded API for Ipv6 compatibility.</p>	<p>void IO_ParseNetworkAddress (const char s[], unsigned int* tla, unsigned int* nla, unsigned int* sla)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>s[]</code> - String to parse.• <code>tla</code> - Storage for tla• <code>nla</code> - Storage for nla.• <code>sla</code> - Storage for sla. <p>Returns:</p>

	<ul style="list-style-type: none">void - None
<div><div>IO_AppParseSourceAndDestStrings</div><div>Application input parsing API. Parses the source and destination strings. Overloaded for Ipv6 compatibility.</div></div>	<div><div>void IO_AppParseSourceAndDestStrings (Node* node, const char* inputString, const char* sourceString, NodeId* sourceNodeId, Address* sourceAddr, const char* destString, NodeId* destNodeId, Address* destAddr)</div><div>Parameters:<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.</div><div>Returns:<ul style="list-style-type: none">void - None</div></div>
<div><div>IO_AppParseSourceString</div><div>Application input parsing API. Parses the source string. Overloaded for Ipv6 compatibility.</div></div>	<div><div>void IO_AppParseSourceString (Node* node, const char* inputString, const char* sourceString, NodeAddress* sourceNodeId, Address* sourceAddr, NetworkType networkType)</div><div>Parameters:<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</div><div>Returns:<ul style="list-style-type: none">void - None</div></div>
<div><div>IO_AppParseDestString</div><div>Application input parsing API. Parses the destination string. Overloaded for Ipv6</div></div>	<div><div>void IO_AppParseDestString (Node* node, const char* inputString, const char* destString, NodeAddress* destNodeId, Address* destAddr, NetworkType networkType)</div><div>Parameters:<ul style="list-style-type: none">node - A pointer to Node.</div></div>

compatibility.	<ul style="list-style-type: none">• <code>inputString</code> - The input string.• <code>destString</code> - Const char pointer.• <code>destNodeId</code> - A pointer to NodeAddress.• <code>destAddr</code> - A pointer to Address.• <code>networkType</code> - used when sourceString <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
<p>QualifierMatches</p> <p>This is an auxiliary API used by the <code>IO_Read...()</code> set of APIs. Overloaded for Ipv6 compatibility</p>	<p>BOOL <code>QualifierMatches</code> (const NodeId nodeId, const in6_addr interfaceAddress, const char* qualifier, int* matchType)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>nodeId</code> - nodeId to select for.• <code>interfaceAddress</code> - IPv6 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>QualifierMatches</p> <p>This is an auxiliary API used by the <code>IO_Read...()</code> set of APIs. Overloaded for Ipv6 compatibility</p>	<p>BOOL <code>QualifierMatches</code> (const NodeId nodeId, const AtmAddress* interfaceAddress, const char* qualifier, int* matchType)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>nodeId</code> - nodeId to select for.• <code>interfaceAddress</code> - ATM 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_ConvertIpv6StringToAddress()</p> <p>Convert IPv6 address string to in6_addr structure. API for Ipv6 compatibility.</p>	<p>void <code>IO_ConvertIpv6StringToAddress()</code> (char* interfaceAddr, in6_addr* ipAddress)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>interfaceAddr</code> - Storage for ipv6address string• <code>ipAddress</code> - Storage for ipv6address.

	<div>Returns:</div> <div><ul style="list-style-type: none"><code>void</code> - None</div>
<div><div>IO_ConvertIpAddressToString</div><div>Parses IPv6 address into a formatted string. Overloaded API for Ipv6 compatibility.</div></div>	<div><div>void IO_ConvertIpAddressToString (in6_addr* ipAddress, char* interfaceAddr)</div><div>Parameters:</div><div><ul style="list-style-type: none"><code>ipAddress</code> - Storage for ipv6address.<code>interfaceAddr</code> - Storage for ipv6address string</div><div>Returns:</div><div><ul style="list-style-type: none"><code>void</code> - None</div></div>
<div><div>IO_ConvertIpAddressToString</div><div>Parses IPv6 address into a formatted string. Overloaded API for Ipv6 compatibility.</div></div>	<div><div>void IO_ConvertIpAddressToString (Address* ipAddress, char* interfaceAddr)</div><div>Parameters:</div><div><ul style="list-style-type: none"><code>ipAddress</code> - IP address info<code>interfaceAddr</code> - Storage for ipv6address string</div><div>Returns:</div><div><ul style="list-style-type: none"><code>void</code> - None</div></div>
<div><div>IO_ConvertStringToNodeAddress</div><div>This API is used to covert a string parameter to NodeAddress.</div></div>	<div><div>void IO_ConvertStringToNodeAddress (char* addressString, NodeAddress* outputNodeAddress)</div><div>Parameters:</div><div><ul style="list-style-type: none"><code>addressString</code> - IP address string info<code>outputNodeAddress</code> - Storage for IP address</div><div>Returns:</div><div><ul style="list-style-type: none"><code>void</code> - None</div></div>
<div><div>IO_CheckIsSameAddress</div><div>Compares IPv4 IPv6 address. API for Ipv6 compatibility.</div></div>	<div><div>BOOL IO_CheckIsSameAddress (Address addr1, Address addr2)</div><div>Parameters:</div><div><ul style="list-style-type: none"><code>addr1</code> - Storage for IPv4 IPv6 address<code>addr2</code> - Storage for IPv4 IPv6 address</div><div>Returns:</div><div><ul style="list-style-type: none"><code>BOOL</code> - None</div></div>
<div><div>IO_ReadString</div></div>	<div><div>void IO_ReadString (Node* node node, const NodeInput* nodeInput, const char* index, BOOL* wasFound, char* readVal)</div><div>Parameters:</div></div>

<p>This API is used to retrieve a string parameter value from input files.</p>	<ul style="list-style-type: none">• <code>node</code> - Node pointer for which string is• <code>nodeInput</code> - Pointer to node input.• <code>index</code> - Parameter name.• <code>wasFound</code> - Storage for success of search.• <code>readVal</code> - Storage for parameter value. <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
<p>IO_CacheFile</p> <p>This API is used to read an auxiliary input file into a <code>NodeInput</code> struct Called for each file variable in the config file.</p>	<p>void IO_CacheFile (const <code>NodeInput*</code> nodeInput, const <code>char*</code> filename)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>nodeInput</code> - Pointer to node input.• <code>filename</code> - Path to input file. <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - NULL
<p>IO_GetMaxLen</p> <p>This API is used to get the maximun length of a line in the file.</p>	<p>int IO_GetMaxLen (fileName <code>char*</code>)</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> - Interger with the largest line length.
<p>IO_GetMaxLen</p> <p>This API is used to get the maximun length of a line in the file.</p>	<p>int IO_GetMaxLen (fp <code>FILE*</code>)</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> - Interger with the largest line length.
<p>NI_GetMaxLen</p> <p>This API is used to get the maximun length of a line in <code>nodeInput</code>.</p>	<p>int NI_GetMaxLen (nodeInput <code>NodeInput*</code>)</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> - Interger with the largest line length.
NI_GetMaxLen This API is used to get the maximun length of a line in <code>nodeInput</code> .	<code>int NI_GetMaxLen (nodeInput const NodeInput*)</code> Parameters: <ul style="list-style-type: none">• <code>const NodeInput*</code> - Pointer to a node input. Returns: <ul style="list-style-type: none">• <code>int</code> - Interger with the largest line length.
IO_ParseNetworkAddress Parses a string for a network address. Overloaded API for ATM compatibility.	<code>void IO_ParseNetworkAddress (const char s[], unsigned int* u_atmVal)</code> Parameters: <ul style="list-style-type: none">• <code>s[]</code> - String to parse.• <code>u_atmVal</code> - Storage for icd, aid, ptp Returns: <ul style="list-style-type: none">• <code>void</code> - None
IO_ConvertAddrToString Convert generic address to appropriate network type address string format.	<code>void IO_ConvertAddrToString (Address* address, char* addrStr)</code> Parameters: <ul style="list-style-type: none">• <code>address</code> - generic address• <code>addrStr</code> - address string Returns: <ul style="list-style-type: none">• <code>void</code> - NULL
IO_ConvertAtmAddressToString Convert Atm address to address string format.	<code>void IO_ConvertAtmAddressToString (AtmAddress addr, char* addrStr)</code> Parameters: <ul style="list-style-type: none">• <code>addr</code> - Atm address• <code>addrStr</code> - address string Returns: <ul style="list-style-type: none">• <code>void</code> - NULL
IO_InsertIntValue Insert integer value for specific string in case of ATM	<code>void IO_InsertIntValue (const char s[], const unsigned int val, unsigned int u_atmVal)</code> Parameters: <ul style="list-style-type: none">• <code>s[]</code> - character array• <code>val</code> - value to be inserted

	<ul style="list-style-type: none">• <code>u_atmVal</code> - <code>atm_value</code> need to be checked <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - <code>NULL</code>
<p>IO_ReadCachedFileIndex</p> <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">• <code>nodeId</code> - node Id• <code>interfaceAddress</code> - Interface Address for the given node• <code>nodeInput</code> - <code>atm_value</code> need to be checked <p>Returns:</p> <ul style="list-style-type: none">• <code>int</code> - <code>None</code>
<p>IO_ReadString</p> <p>This API is used to retrieve a string parameter value from input files.</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> <ul style="list-style-type: none">• <code>node</code> - node structure pointer.• <code>nodeId</code> - <code>nodeId</code>.• <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> - <code>None</code>
<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">• <code>node</code> - node structure pointer.• <code>nodeId</code> - <code>nodeId</code>.• <code>interfaceIndex</code> - interface Index.

	<ul style="list-style-type: none">• <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_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">• <code>node</code> - node structure pointer.• <code>nodeId</code> - <code>nodeId</code>.• <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
<p>IO_ReadInt</p> <p>This API is used to retrieve a Int parameter value from input files.</p>	<p>void IO_ReadInt (Node* node, const NodeId nodeId, int interfaceIndex, const NodeInput* nodeInput, const char* parameterName, BOOL* wasFound, int* parameterValue)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>node</code> - node structure pointer.• <code>nodeId</code> - <code>nodeId</code>.• <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">void - None
<p>IO_ReadDouble</p> <p>This API is used to retrieve a double parameter value from input files.</p>	<p>void IO_ReadDouble (Node* node, const NodeId nodeId, int interfaceIndex, const NodeInput* nodeInput, const char* parameterName, BOOL* wasFound, double* 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_ReadCachedFile</p> <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
<p>IO_ReadLine</p> <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.

	<ul style="list-style-type: none">• <code>nodeId</code> - <code>nodeId</code>.• <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
<p>IO_ReadStringInstance</p> <p>This API is used to retrieve string parameter values from input files for a specific instance.</p>	<p><code>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)</code></p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>node</code> - node structure pointer.• <code>nodeId</code> - <code>nodeId</code>.• <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
<p>IO_ReadDoubleInstance</p> <p>This API is used to retrieve double parameter values from input files for a specific instance.</p>	<p><code>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)</code></p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>node</code> - node structure pointer.• <code>nodeId</code> - <code>nodeId</code>.• <code>interfaceIndex</code> - interface Index.

	<ul style="list-style-type: none">• <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
<p>IO_ReadIntInstance</p> <p>This API is used to retrieve int parameter values from input files for a specific instance.</p>	<p>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)</p> <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
<p>IO_ReadTimeInstance</p> <p>This API is used to retrieve clocktype parameter values from input files for a specific instance.</p>	<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">• <code>node</code> - node structure pointer.• <code>nodeId</code> - nodeId.

	<div><div>interfaceIndex - interface Index.</div><div><ul style="list-style-type: none">nodeInput - Pointer to node input.parameterName - Parameter name.parameterInstanceNumber - Instance number.fallbackIfNoInstanceMatch - Selects parameter withoutwasFound - Storage for success of seach.parameterValue - Storage for parameter value.</div></div> <div>Returns:<div><ul style="list-style-type: none">void - None</div></div>
<div><div>IO_ReadCachedFileInstance</div><div>This API is used to retrieve cached file parameter values from input files for a specific instance.</div></div>	<div><div>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)</div><div>Parameters:<div><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 withoutwasFound - Storage for success of seach.parameterValue - Storage for parameter value.</div></div><div>Returns:<div><ul style="list-style-type: none">void - None</div></div></div>
<div><div>IO_ReadStringUsingIpAddress</div><div>This API is used to retrieve a string parameter value from input files using the ip-address.</div></div>	<div><div>void IO_ReadStringUsingIpAddress (Node* node, int interfaceIndex, const NodeInput* nodeInput, const char* parameterName, BOOL* wasFound, char* parameterValue)</div><div>Parameters:<div><ul style="list-style-type: none">node - node structure pointer.interfaceIndex - interface Index.</div></div></div>

	<ul style="list-style-type: none">• <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_ReadString</p> <p>This API is used to retrieve a string parameter value from input files.</p>	<p>void IO_ReadString (const NodeAddress nodeId, NodeAddress ipv4Address, in6_addr* ipv6Address, const NodeInput* nodeInput, const char* parameterName, BOOL* wasFound, char* parameterValue)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>nodeId</code> - nodeId.• <code>ipv4Address</code> - IP address of an interface• <code>ipv6Address</code> - IPv6 address of an 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_ReadString</p> <p>This API is used to retrieve a string parameter value from input files.</p>	<p>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)</p> <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.

	<ul style="list-style-type: none">wasFound - Storage for success of search.matchType - Storage for matchType. Returns: <ul style="list-style-type: none">void - None
IO_ReadChannelMask This API is used to retrieve the value channel mask for a specific instance.	<p>void IO_ReadChannelMask (Node* node, const NodeAddress nodeId, const Address* address, int interfaceIndex, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, BOOL* parseChannelList, char* parameterValue)</p> Parameters: <ul style="list-style-type: none">node - node structure pointer.nodeId - nodeId.address - Pointer to addressinterfaceIndex - interface Index.nodeInput - Pointer to node input.parameterName - Parameter name.parameterInstanceNumber - Instance number.fallbackIfNoInstanceMatch - Selects parameter withoutwasFound - Storage for success of search.parseChannelList - Storage for identifying ifparameterValue - Storage for parameter value. Returns: <ul style="list-style-type: none">void - None
IO_ReadBool This API is used to retrieve a boolean parameter value from input files.	<p>void IO_ReadBool (Node* node, const NodeId nodeId, int interfaceIndex, const NodeInput* nodeInput, const char* parameterName, BOOL* wasFound, BOOL* parameterValue)</p> Parameters: <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">wasFound - Storage for success of seach.parameterValue - Storage for parameter value. Returns: <ul style="list-style-type: none">void - None
IO_ReadFloat This API is used to retrieve a float parameter value from input files.	<p>void IO_ReadFloat (Node* node, const NodeAddress nodeId, int interfaceIndex, const NodeInput* nodeInput, const char* parameterName, BOOL* wasFound, float* parameterValue)</p> <p>Parameters:</p> <ul style="list-style-type: none">node - node structure pointer.nodeId - nodeId. Can be ANY_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



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 © 2001-2013 [SCALABLE Network Technologies, Inc.](#) All rights reserved.

QualNet 7.1 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.
ENUMERATION	GuiVisObjCommands Commands for displaying visualization objects
ENUMERATION	GuiVisShapes Shape selections for GUI_DRAW_SHAPE command
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.
STRUCT	MetricData Class to identify a specific dynamic statistic.
STRUCT	MetricLayerData Contains a list of the metrics collected at a layer of the protocol stack.
STRUCT	GuiCommand Structure containing command received from Animator.

Function / Macro Summary

Return Type	Summary
void	GUI_HandleHITLInput (const char * args, PartitionData * partition) 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 HITL

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

	<p>Changes the icon associated with a node.</p>
void	<p>GUI_SetNodeLabel(NodeId nodeId, char* label, clocktype time)</p>
	<p>Changes the label (the node name) of a node.</p>
void	<p>GUI_SetNodeRange(NodeId nodeId, int interfaceIndex, double range, clocktype time)</p>
	<p>Changes the transmission range of a node</p>
void	<p>GUI_SetNodeType(NodeId nodeId, int type, clocktype time)</p>
	<p>Changes the (symbolic) type of a node</p>
void	<p>GUI_SetPatternIndex(Node* node, int interfaceIndex, int index, clocktype time)</p>
	<p>Sets the antenna pattern to one of a previously specified antenna pattern file.</p>
void	<p>GUI_SetPatternAndAngle(node node*, int interfaceIndex, int index, int angleInDegrees, clocktype time)</p>
	<p>For steerable antennas, it sets the pattern to use, and also an angle relative to the node's current orientation.</p>
void	<p>GUI_AddLink(NodeId sourceID, NodeId destID, GuiLayers layer, int type, NodeAddress subnetAddress, int numHostBits, clocktype time)</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>
void	<p>GUI_AddLink(NodeId sourceID, NodeId destID, GuiLayers layer, int type, int tla, int nla, int sla, clocktype time)</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>
void	<p>GUI_AddLink(NodeId sourceID, NodeId destID, GuiLayers layer, int type, IPv6subnetAddress ip6_addr, IPv6subnetPrefixLen unsigned int, clocktype time)</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>
void	<p>GUI_DeleteLink(NodeId sourceID, NodeId destID, GuiLayers layer, int type, clocktype time)</p>
	<p>Removes link of a specific type.</p>
void	<p>GUI_DeleteLink(NodeId sourceID, NodeId destID, GuiLayers layer, clocktype time)</p>
	<p>Removes the aforementioned link, no matter the "type."</p>

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_Unicast (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)

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

	<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>
void	<p>GUI_SendIntegerData(NodeId nodeId, int metricID, int value, clocktype time)</p> <p>Sends data for an integer metric.</p>
void	<p>GUI_SendUnsignedData(NodeId nodeId, int metricID, unsigned value, clocktype time)</p> <p>Sends data for an unsigned metric.</p>
void	<p>GUI_SendRealData(NodeId nodeId, int metricID, double value, clocktype time)</p> <p>Sends data for a floating point metric.</p>
bool	<p>GUI_isAnimateOrInteractive()</p> <p>Returns true if the GUI was activated on the command line.</p>
void	<p>GUI_EXTERNAL_Bootstrap(int argc, char** argv, NodeInput* nodeInput, int thisPartitionId)</p> <p>Creates a connection to the GUI</p>
void	<p>GUI_EXTERNAL_Registration(PartitionData* partitionData, EXTERNAL_InterfaceList* list)</p> <p>Registers the GUI as an external interface</p>
void	<p>GUI_CreateReply(GuiReplies replyType, std msg)</p> <p>Function used to replace newline characters in a string being sent to the GUI.</p>

Constant / Data Structure Detail

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

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

	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.
Enumeration	GuiVisObjCommands
	Commands for displaying visualization objects
Enumeration	GuiVisShapes
	Shape selections for GUI_DRAW_SHAPE command
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 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 HITL commands	void GUI_HandleHITLInput (const char * args, PartitionData * partition) Parameters: <ul style="list-style-type: none">args - the command itselfpartition - the partition pointer Returns: <ul style="list-style-type: none">void - NULL
GUI_Initialize Initializes the GUI in order to start the animation. The terrain map should give the path (either absolute, or relative to QUALNET_HOME) of an file to represent	void GUI_Initialize (NodeInput* nodeInput, int numNodes, int coordinateSystem, Coordinates origin, Coordinates dimensions, clocktype maxClock) Parameters: <ul style="list-style-type: none">nodeInput - configuration filenumNodes - the number of nodes in the simulation

the terrain.	<ul style="list-style-type: none">• <code>coordinateSystem</code> - LATLONALT or CARTESIAN• <code>origin</code> - Southwest corner• <code>dimensions</code> - Northeast corner, or size• <code>maxClock</code> - length of the simulation <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - 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">• <code>event</code> - the type of event for the new effect• <code>layer</code> - the protocol layer• <code>type</code> - special key to distinguish similar events• <code>effect</code> - the effect to use• <code>color</code> - optional color for the effect <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - 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">• <code>node</code> - the node• <code>nodeInput</code> - configuration file• <code>time</code> - the current simulation time <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - 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">• <code>node</code> - the node• <code>interfaceIndex</code> - the interface to initialize <p>Returns:</p>

	<ul style="list-style-type: none">void - NULL
GUI_InitializeInterfaces Sets the IP address associated with one of the node's interfaces.	<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 Sets the IP address associated with one of the node's interfaces.	<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 IDinterfaceAddress - new IP addressinterfaceIndex - interface Address to changetime - the current simulation time <p>Returns:</p> <ul style="list-style-type: none">void - NULL
GUI_SetSubnetMask Sets the Subnet mask associated with one of the node's interfaces.	<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 IDinterfaceAddress - new Subnet maskinterfaceIndex - Subnet mask to changetime - the current simulation time <p>Returns:</p> <ul style="list-style-type: none">void - NULL
GUI_SetInterfaceName Sets the Interface name associated with one of the node's interfaces.	<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 IDinterfaceAddress - new Interface nameinterfaceIndex - interface Name to changetime - the current simulation time

	<p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - <code>NULL</code>
<p>GUI_MoveNode</p> <p>Moves the node to a new position.</p>	<p>void GUI_MoveNode (NodeId nodeId, Coordinates position, clocktype time)</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</code> - <code>NULL</code>
<p>GUI_SetNodeOrientation</p> <p>Changes the orientation of a node.</p>	<p>void GUI_SetNodeOrientation (NodeId nodeId, Orientation orientation, clocktype time)</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</code> - <code>NULL</code>
<p>GUI_SetNodeIcon</p> <p>Changes the icon associated with a node.</p>	<p>void GUI_SetNodeIcon (NodeId nodeId, char* iconFile, clocktype time)</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</code> - <code>NULL</code>
<p>GUI_SetNodeLabel</p> <p>Changes the label (the node name) of a node.</p>	<p>void GUI_SetNodeLabel (NodeId nodeId, char* label, clocktype time)</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
<p>GUI_SetNodeRange</p> <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
<p>GUI_SetNodeType</p> <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 GUI_SetEffect• <code>time</code> - the current simulation time <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - NULL
<p>GUI_SetPatternIndex</p> <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
<p>GUI_SetPatternAndAngle</p>	<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><code>void GUI_AddLink (NodeId sourceID, NodeId destID, GuiLayers layer, int type, NodeAddress subnetAddress, int numHostBits, 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>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><code>void GUI_AddLink (NodeId sourceID, NodeId destID, GuiLayers layer, int type, int tla, int nla, int sla, 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>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">time - the current simulation time <p>Returns:</p> <ul style="list-style-type: none">void - 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, IPv6subnetAddress ip6_addr, IPv6subnetPrefixLen unsigned int, clocktype time)</p> <p>Parameters:</p> <ul style="list-style-type: none">sourceID - the source node for the linkdestID - the destination nodelayer - the protocol layer associated w/ the linktype - a user-defined type for the linkip6_addr - IPv6 addressunsigned int - IPv6 address prefix lengthtime - the current simulation time <p>Returns:</p> <ul style="list-style-type: none">void - NULL
<p>GUI_DeleteLink</p> <p>Removes link of a specific type.</p>	<p>void GUI_DeleteLink (NodeId sourceID, NodeId destID, GuiLayers layer, int type, clocktype time)</p> <p>Parameters:</p> <ul style="list-style-type: none">sourceID - the source node for the linkdestID - the destination nodelayer - the protocol layer associated w/ the linktype - type of link being deletedtime - the current simulation time <p>Returns:</p> <ul style="list-style-type: none">void - NULL
<p>GUI_DeleteLink</p> <p>Removes the aforementioned link, no matter the "type."</p>	<p>void GUI_DeleteLink (NodeId sourceID, NodeId destID, GuiLayers layer, clocktype time)</p> <p>Parameters:</p> <ul style="list-style-type: none">sourceID - the source node for the linkdestID - 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
<p>GUI_Broadcast</p> <p>Indicates a broadcast.</p>	<p>void GUI_Broadcast (NodeId nodeID, GuiLayers layer, int type, int interfaceIndex, clocktype time)</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
<p>GUI_EndBroadcast</p> <p>Indicates the end of a broadcast.</p>	<p>void GUI_EndBroadcast (NodeId nodeID, GuiLayers layer, int type, int interfaceIndex, clocktype time)</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
<p>GUI_Multicast</p> <p>Indicates a multicast. (Probably need to add a destination address.)</p>	<p>void GUI_Multicast (NodeId nodeID, GuiLayers layer, int type, int interfaceIndex, clocktype time)</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">time - the current simulation time <p>Returns:</p> <ul style="list-style-type: none">void - NULL
<p>GUI_Unicast</p> <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>void GUI_Unicast (NodeId sourceID, NodeId destID, GuiLayers layer, int type, int sendingInterfaceIndex, int receivingInterfaceIndex, clocktype time)</p> <p>Parameters:</p> <ul style="list-style-type: none">sourceID - the source nodedestID - the destination nodelayer - protocol layer associated w/ the eventtype - a user-defined typesendingInterfaceIndex - sender's interface to usereceivingInterfaceIndex - receiver's interface to usetime - the current simulation time <p>Returns:</p> <ul style="list-style-type: none">void - NULL
<p>GUI_Receive</p> <p>Shows a successful receipt at a destination.</p>	<p>void GUI_Receive (NodeId sourceID, NodeId destID, GuiLayers layer, int type, int sendingInterfaceIndex, int receivingInterfaceIndex, clocktype time)</p> <p>Parameters:</p> <ul style="list-style-type: none">sourceID - the source nodedestID - the destination nodelayer - protocol layer associated w/ the eventtype - a user-defined typesendingInterfaceIndex - sender's interface to usereceivingInterfaceIndex - receiver's interface to usetime - the current simulation time <p>Returns:</p> <ul style="list-style-type: none">void - NULL
<p>GUI_Drop</p>	<p>void GUI_Drop (NodeId sourceID, NodeId destID, GuiLayers layer, int type, int sendingInterfaceIndex, int receivingInterfaceIndex, clocktype time)</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
<p>GUI_Collision</p> <p>Shows a node detecting a collision.</p>	<p><code>void GUI_Collision</code> (NodeId nodeID, GuiLayers layer, clocktype time)</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
<p>GUI_CreateSubnet</p> <p>Creates a subnet. Normally done at startup.</p>	<p><code>void GUI_CreateSubnet</code> (GuiSubnetTypes type, NodeAddress subnetAddress, int numHostBits, const char* nodeList, clocktype time)</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
<p>GUI_CreateSubnet</p>	<p><code>void GUI_CreateSubnet</code> (GuiSubnetTypes type, in6_addr IPv6subnetAddress, unsigned int IPv6subnetPrefixLen, const</p>

<p>Creates a IPv6 subnet. Normally done at startup.</p>	<p>char* nodeList, clocktype time)</p> <p>Parameters:</p> <ul style="list-style-type: none">• type - GUI_WIRED/WIRELESS/SATELLITE_NETWORK• IPv6subnetAddress - base address for the subnet• IPv6subnetPrefixLen - number of network bits present• nodeList - the rest of the .config file SUBNET line• time - the current simulation time <p>Returns:</p> <ul style="list-style-type: none">• void - NULL
<p>GUI_CreateSubnet</p> <p>Creates a IPv6 subnet. Normally done at startup.</p>	<p>void GUI_CreateSubnet (GuiSubnetTypes type, IPv6subnetAddress ip6_addr, IPv6subnetPrefixLen unsigned int, const char* nodeList, clocktype time)</p> <p>Parameters:</p> <ul style="list-style-type: none">• type - GUI_WIRED/WIRELESS/SATELLITE_NETWORK• ip6_addr - IPv6 address• unsigned int - IPv6 address prefix length• nodeList - the rest of the .config file SUBNET line• time - the current simulation time <p>Returns:</p> <ul style="list-style-type: none">• void - NULL
<p>GUI_CreateHierarchy</p> <p>Since the GUI supports hierarchical design, this function informs the GUI of the contents of a hierarchical component.</p>	<p>void GUI_CreateHierarchy (int componentID, char* nodeList)</p> <p>Parameters:</p> <ul style="list-style-type: none">• componentID - an identifier for the hierarchy• nodeList - the rest of the .config file COMPONENT line <p>Returns:</p> <ul style="list-style-type: none">• void - NULL
<p>GUI_MoveHierarchy</p> <p>Moves the center point of a hierarchy to a new position.</p>	<p>void GUI_MoveHierarchy (int hierarchyId, Coordinates centerCoordinates, Orientation orientation, clocktype time)</p> <p>Parameters:</p> <ul style="list-style-type: none">• hierarchyId - 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
<p>GUI_CreateWeatherPattern</p> <p>Sends the input line describing a weather pattern to the GUI.</p>	<p>void GUI_CreateWeatherPattern (int patternID, char* inputLine)</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
<p>GUI_MoveWeatherPattern</p> <p>Moves the first point of a weather pattern to a new position.</p>	<p>void GUI_MoveWeatherPattern (int patternID, Coordinates coordinates, clocktype time)</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
<p>GUI_AddApplication</p> <p>Shows label beside the client and the server as app link is setup.</p>	<p>void GUI_AddApplication (NodeId sourceID, NodeId destID, char* appName, int uniqueId, clocktype time)</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
<p>GUI_DeleteApplication</p>	<p>void GUI_DeleteApplication (NodeId sourceID, NodeId destID, char* appName, int uniqueId, clocktype time)</p>

Deletes the labels shown by AddApplication.	<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
<p>GUI_AddInterfaceQueue</p> <p>Creates a queue for a node, interface and priority.</p>	<p><code>void GUI_AddInterfaceQueue (NodeId nodeId, GuiLayers layer, int interfaceIndex, unsigned priority, int queueSize, clocktype time)</code></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
<p>GUI_QueueInsertPacket</p> <p>Inserting one packet to a queue for a node, interface and priority</p>	<p><code>void GUI_QueueInsertPacket (NodeId nodeId, GuiLayers layer, int interfaceIndex, unsigned priority, int packetSize, clocktype time)</code></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"><code>void</code> - NULL
<p>GUI_QueueDropPacket</p> <p>Dropping one packet from a queue for a node, interface and priority.</p>	<p><code>void GUI_QueueDropPacket (NodeId nodeID, GuiLayers layer, int interfaceIndex, unsigned priority, clocktype time)</code></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>time</code> - the current simulation time <p>Returns:</p> <ul style="list-style-type: none"><code>void</code> - NULL
<p>GUI_QueueDequeuePacket</p> <p>Dequeuing one packet from a queue for a node, interface and priority</p>	<p><code>void GUI_QueueDequeuePacket (NodeId nodeID, GuiLayers layer, int interfaceIndex, unsigned priority, int packetSize, clocktype time)</code></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"><code>void</code> - NULL
<p>GUI_DefineMetric</p> <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><code>int GUI_DefineMetric (char* name, NodeId nodeID, GuiLayers layer, int linkID, GuiDataTypes datatype, GuiMetrics metrictype)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"><code>name</code> - the name of the metric<code>nodeID</code> - the node's ID<code>layer</code> - protocol layer associated w/ the event<code>linkID</code> - e.g., an application session ID

	<ul style="list-style-type: none">• datatype - real/unsigned/integer• metrictype - cumulative/average, etc. <p>Returns:</p> <ul style="list-style-type: none">• int - an identifier associated the the metric name and layer
<p>GUI_SendIntegerData</p> <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">• nodeID - the node's ID• metricID - the value returned by DefineMetric• value - the current value of the metric• time - the current simulation time <p>Returns:</p> <ul style="list-style-type: none">• void - NULL
<p>GUI_SendUnsignedData</p> <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">• nodeID - the node's ID• metricID - the value returned by DefineMetric• value - the current value of the metric• time - the current simulation time <p>Returns:</p> <ul style="list-style-type: none">• void - NULL
<p>GUI_SendRealData</p> <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">• nodeID - the node's ID• metricID - the value returned by DefineMetric• value - the current value of the metric• time - the current simulation time <p>Returns:</p>

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



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 © 2001-2013 [SCALABLE Network Technologies, Inc.](#) All rights reserved.

QualNet 7.1 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>IP_TOS_PREC_EF</u> IP precedence 'EF' class internet control
CONSTANT	<u>IP_TOS_PREC_NET</u> IP precedence 'net control'
CONSTANT	<u>IP_TOS_PREC_INTERNETCONTROL</u> IP precedence 'internet control'
CONSTANT	<u>IP_TOS_PREC_CRITIC_ECP</u> IP precedence 'critic ecp'
CONSTANT	<u>IP_TOS_PREC_FLASHOVERRIDE</u> IP precedence 'flash override'
CONSTANT	<u>IP_TOS_PREC_FLASH</u> IP precedence 'flash'
CONSTANT	<u>IP_TOS_PREC_IMMEDIATE</u> IP precedence 'immediate'
CONSTANT	<u>IP_TOS_PREC_PRIORITY</u> IP precedence 'priority'
CONSTANT	<u>IP_TOS_PREC_ROUTINE</u> IP precedence 'routing'
CONSTANT	<u>IP_TOS_PREC_INTERNETCONTROL_MIN_DELAY_SET</u> IP precedence 'internet control' with the 'minimize delay' bit set
CONSTANT	<u>IP_TOS_PREC_CRITIC_ECP_MIN_DELAY_SET</u>

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

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

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

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

	IP protocol numbers for IPSEC.
CONSTANT	<u>IPPROTO_ISAKMP</u>
	IP protocol numbers for IPSEC.
CONSTANT	<u>IPPROTO_CES_ISAKMP</u>
	IP protocol numbers for IPSEC.
CONSTANT	<u>IPPROTO_IKEP</u>
	IP protocol numbers.
CONSTANT	<u>IPPROTO_OSPF</u>
	IP protocol numbers for OSPF .
CONSTANT	<u>IPPROTO_PIM</u>
	IP protocol numbers for PIM .
CONSTANT	<u>IPPROTO_RPIM</u>
	IP protocol numbers for PIM .
CONSTANT	<u>IPPROTO_IGRP</u>
	IP protocol numbers for IGRP .
CONSTANT	<u>IPPROTO_EIGRP</u>
	IP protocol numbers for EIGRP .
CONSTANT	<u>IPPROTO_BELLMANFORD</u>
	IP protocol numbers for BELLMANFORD.
CONSTANT	<u>IPPROTO_IPIP_RED</u>
	IP protocol numbers for IP_RED.
CONSTANT	<u>IPPROTO_FISHEYE</u>

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

	IP protocol numbers for ROUTING_CES_SRW IP encapsulation.
CONSTANT	<u>IPPROTO_IPIP_SDR</u>
	IP protocol numbers for SDR IP encapsulation.
CONSTANT	<u>IPPROTO_IPIP_SDR</u>
	IP protocol numbers for SDR IP encapsulation.
CONSTANT	<u>IPPROTO_MULTICAST_CES_SRW_MOSPF</u>
	IP protocol numbers for MULTICAST_CES_SRW_MOSPF IP encapsulation.
CONSTANT	<u>IPPROTO_CES_HSLs</u>
	IP protocol numbers for HSLs protocol.
CONSTANT	<u>IPPROTO_AODV</u>
	IP protocol numbers for AODV .
CONSTANT	<u>IPPROTO_DYMO</u>
	IP protocol numbers for DYMO .
CONSTANT	<u>IPPROTO_MAODV</u>
	IP protocol numbers for MAODV.
CONSTANT	<u>IPPROTO_DSR</u>
	IP protocol numbers for DSR .
CONSTANT	<u>IPPROTO_ODMRP</u>
	IP protocol numbers for ODMRP .
CONSTANT	<u>IPPROTO_LAR1</u>
	IP protocol numbers for LAR1.
CONSTANT	<u>IPPROTO_STAR</u>

	IP protocol numbers for STAR.
CONSTANT	<u>IPPROTO_DAWN</u>
	IP protocol numbers for DAWN.
CONSTANT	<u>IPPROTO_EPLRS</u>
	IP protocol numbers for EPLRS protocol.
CONSTANT	<u>IPPROTO_CES_EPLRS</u>
	IP protocol numbers for EPLRS protocol for CES.
CONSTANT	<u>IPPROTO_CES_EPLRS_MPR</u>
	IP protocol numbers for EPLRS MPR protocol.
CONSTANT	<u>IPPROTO_DVMRP</u>
	IP protocol numbers for DVMRP.
CONSTANT	<u>IPPROTO_GSM</u>
	IP protocol numbers for GSM.
CONSTANT	<u>IPPROTO_EXTERNAL</u>
	IP protocol for external interface.
CONSTANT	<u>IPPROTO_INTERNET_GATEWAY</u>
	IP protocol numbers for Internet gateway for emulated nodes
CONSTANT	<u>IPPROTO_EXATA_VIRTUAL_LAN</u>
	IP protocol numbers for Internet gateway for emulated nodes
CONSTANT	<u>IPPROTO_NDP</u>
	IP protocol numbers for NDP.
CONSTANT	<u>IPPROTO_BRP</u>
	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
STRUCT	NetworkIpMulticastGroupEntry
	Structure for Multicast Group Entry
STRUCT	IpPerHopBehaviorInfoType
	Structure to maintain DS priority queue mapping
STRUCT	IpMfqcParameter
	Variables of the structure define a unique condition class
STRUCT	IpMultiFieldTrafficConditionerInfo
	Structure used to store traffic condition.
STRUCT	IpOptionsHeaderType
	Structure of optional header for IP source route
STRUCT	NetworkIpStatsType
	Structure used to keep track of all stats of network layer.
STRUCT	NetworkForwardingTableRow
	Structure of an entity of forwarding table.
STRUCT	NetworkForwardingTable
	Structure of forwarding table.
STRUCT	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.
STRUCT	ip_frag_data
	QualNet typedefs struct ip_frag_data to IpFragData. 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	NetworkDataIp;
	Main structure of network layer.
STRUCT	IpReassemblyBufferType
	Structure of reassembly buffer
STRUCT	IpReassemblyBufferListCellType
	Structure of reassembly buffer cell listing
STRUCT	IpReassemblyBufferListType
	Structure of reassembly buffer list
STRUCT	AddressChangeType
	enumeration to define address change events by DHCP

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
MACRO	FragmentOffset(ipHeader)
	Starting position of this fragment in actual packet.
MACRO	SetFragmentOffset(ipHeader, offset)
	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
unsigned int	IpHeaderGetHLen() (UInt32 ip_v_hl_tos_len)
	Returns the value of header length for IpHeaderType
unsigned int	IpHeaderGetTOS() (UInt32 ip_v_hl_tos_len)
	Returns the value of Type of Service for IpHeaderType
unsigned int	IpHeaderGetIpLength() (UInt32 ip_v_hl_tos_len)
	Returns the value of ip length for IpHeaderType
UInt16	IpHeaderGetIpFragOffset() (UInt16 ipFragment)
	Returns the value of ip_fragment_offset for IpHeaderType
BOOL	IpHeaderGetIpDontFrag() (UInt16 ipFragment)
	Returns the value of ip_dont_fragment for IpHeaderType
BOOL	IpHeaderGetIpMoreFrag() (UInt16 ipFragment)
	Returns the value of ip_more_fragment for IpHeaderType
BOOL	IpHeaderGetIpReserved() (UInt16 ipFragment)
	Returns the value of ipReserved for IpHeaderType
void	Ip_timestampSetFlag() (unsigned char flgOflw, unsigned char flag)
	Set the value of flag for ip_timestamp_str
void	Ip_timestampSetOvflw() (unsigned char flgOflw, unsigned char ovflw)
	Set the value of ovflw for ip_timestamp_str
unsigned char	Ip_timestampGetFlag() (unsigned char flgOflw)

	Returns the value of flag for ip_timestamp_str
unsigned char	Ip_timestampGetOvflw() (unsigned char flgOflw)
	Returns the value of overflow counter for ip_timestamp_str
NodeAddress	ConvertNumHostBitsToSubnetMask (int numHostBits)
	To generate subnetmask using number of host bit
int	ConvertSubnetMaskToNumHostBits (NodeAddress subnetMask)
	To generate number of host bit using subnetmask.
NodeAddress	MaskIpAddress (NodeAddress address, NodeAddress mask)
	To mask a ip address.
NodeAddress	MaskIpAddressWithNumHostBits (NodeAddress address, int numHostBits)
	To mask a ip address.
NodeAddress	CalcBroadcastIpAddress (NodeAddress address, int numHostBits)
	To generate broadcast address.
BOOL	IsIpAddressInSubnet (NodeAddress address, NodeAddress subnetAddress, int numHostbits)
	To check if a ip address belongs to a subnet.
void	NetworkIpAddHeader (Node* node, Message* msg, NodeAddress sourceAddress, NodeAddress destinationAddress, TosType priority, unsigned char protocol, unsigned ttl)
	Add an IP packet header to a message. Just calls AddIpHeader.
IpOptionsHeaderType*	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	NetworkIpFinalize (Node* node)
	Finalize function for the IP model. Finalize functions for all network-layer IP protocols are called here.
void	NetworkIpReceivePacketFromTransportLayer (Node* node, Message* msg, NodeAddress sourceAddress, NodeAddress destinationAddress, int outgoingInterface, TosType priority, unsigned char protocol, BOOL isEcncapable)
	Called by transport layer protocols (UDP, TCP) to send UDP datagrams and TCP segments using IP. Simply calls NetworkIpSendRawMessage() .
void	NetworkIpSendRawMessage (Node* node, Message* msg, NodeAddress sourceAddress, NodeAddress destinationAddress, int outgoingInterface, TosType priority, unsigned char protocol, unsigned ttl)
	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() .
void	NetworkIpSendRawMessageWithDelay (Node* node, Message* msg, NodeAddress sourceAddress, NodeAddress destinationAddress, int outgoingInterface, TosType priority, unsigned char protocol, unsigned ttl, clocktype delay)
	Same as NetworkIpSendRawMessage() , but schedules event after a simulation delay.
void	NetworkIpSendRawMessageToMacLayer (Node* node, Message* msg, NodeAddress sourceAddress, NodeAddress destinationAddress, TosType priority, unsigned char protocol, unsigned ttl, int interfaceIndex, NodeAddress nextHop)
	Called by network-layer routing protocols (AODV, OSPF, etc.) to add an IP header to payload data, and with the resulting IP packet, calls NetworkIpSendPacketOnInterface() .
void	NetworkIpSendRawMessageToMacLayerWithDelay (Node* node, Message* msg, NodeAddress sourceAddress, NodeAddress destinationAddress, TosType priority, unsigned char protocol, unsigned ttl, int interfaceIndex, NodeAddress nextHop, clocktype delay)
	Same as NetworkIpSendRawMessageToMacLayer() , but schedules the event after a simulation delay by calling NetworkIpSendPacketOnInterfaceWithDelay() .
void	NetworkIpSendPacketToMacLayer (Node* node, Message* msg, int interfaceIndex, NodeAddress nextHop)
	This function is called once the outgoing interface index and next hop address to which to route an IP packet are known.
void	NetworkIpSendPacketOnInterface (Node* node, Message* msg, int incomingInterface, int outgoingInterface,

	<p>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 queues an IP packet for delivery to the MAC layer. This functions 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.</p>
void	<p>NetworkIpSendPacketToMacLayerWithDelay(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>NetworkIpSendPacketOnInterfaceWithDelay(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>NetworkIpSendRawPacketOnInterfaceWithDelay(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>NetworkIpSendPacketToMacLayerWithNewStrictSourceRoute(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>NetworkIpReceivePacketFromMacLayer(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>NetworkIpReceivePacket(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>NetworkIpNotificationOfPacketDrop(Node* node, Message* msg, NodeAddress nextHopNodeAddress, int interfaceIndex)</p> <p>Invoke callback functions when a packet is dropped.</p>
MacLayerStatusEventHandlerFunctionType	<p>NetworkIpGetMacLayerStatusEventHandlerFunction(Node* node, int interfaceIndex)</p> <p>Get the status event handler function pointer.</p>
void	<p>NetworkIpSetMacLayerStatusEventHandlerFunction(Node* node, MacLayerStatusEventHandlerFunctionType StatusEventHandlerPtr, int interfaceIndex)</p>

	Allows the MAC layer to send status messages (e.g., packet drop, link failure) to a network-layer routing protocol for routing optimization.
void	NetworkIpSneakPeekAtMacPacket (Node* node, const Message* msg, int interfaceIndex, NodeAddress prevHop) 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".
PromiscuousMessagePeekFunctionType	NetworkIpGetPromiscuousMessagePeekFunction (Node* node, int interfaceIndex) Returns the network-layer function which will promiscuously inspect packets. See NetworkIpSneakPeekAtMacPacket() .
void	NetworkIpSetPromiscuousMessagePeekFunction (Node* node, PromiscuousMessagePeekFunctionType PeekFunctionPtr, int interfaceIndex) Sets the network-layer function which will promiscuously inspect packets. See NetworkIpSneakPeekAtMacPacket() .
void	NetworkIpReceiveMacAck (Node* node, int interfaceIndex, const Message* msg, NodeAddress nextHop) MAC received an ACK, so call ACK handler function.
MacLayerAckHandlerType	NetworkIpGetMacLayerAckHandler (Node* node, int interfaceIndex) Get MAC layer ACK handler
void	NetworkIpSetMacLayerAckHandler (Node* node, MacLayerAckHandlerType macAckHandlerPtr, int interfaceIndex) Set MAC layer ACK handler
void	SendToUdp (Node* node, Message* msg, TosType priority, NodeAddress sourceAddress, NodeAddress destinationAddress, int incomingInterfaceIndex) 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.
void	SendToTcp (Node* node, Message* msg, TosType priority, NodeAddress sourceAddress, NodeAddress destinationAddress, BOOL aCongestionExperienced) 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..
void	SendToRsvp (Node* node, Message* msg, TosType priority, NodeAddress sourceAddress, NodeAddress destinationAddress, int interfaceIndex, unsigned ttl)

	<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>
	<p>Removes the IP header from a message while also returning all the fields of the header.</p>
void	<p>AddIpOptionField(Node* node, Message* msg, int optionCode, int optionSize)</p>
	<p>Inserts an option field in the header of an IP packet.</p>
void	<p>ExtractIpSourceAndRecordedRoute(Message* msg, NodeAddress[] RouteAddresses, int* NumAddresses, int* RouteAddressIndex)</p>
	<p>Retrieves a copy of the source and recorded route from the options field in the header.</p>
RouterFunctionType	<p>NetworkIpGetRouterFunction(Node* node, int interfaceIndex)</p>
	<p>Get the router function pointer.</p>
void	<p>NetworkIpSetRouterFunction(Node* node, RouterFunctionType RouterFunctionPtr, int interfaceIndex)</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. 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 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.</p>
void	<p>NetworkIpAddUnicastRoutingProtocolType(Node* node, NetworkRoutingProtocolType routingProtocolType, int interfaceIndex)</p>
	<p>Add unicast routing protocol type to interface.</p>
void	<p>NetworkIpAddUnicastIntraRegionRoutingProtocolType(Node* node, NetworkRoutingProtocolType routingProtocolType, int interfaceIndex)</p>
	<p>Add unicast intra region routing protocol type to interface.</p>
void*	<p>NetworkIpGetRoutingProtocol(Node* node, NetworkRoutingProtocolType routingProtocolType)</p>
	<p>Get routing protocol structure associated with routing protocol running on this interface.</p>
NetworkRoutingProtocolType	<p>NetworkIpGetUnicastRoutingProtocolType(Node* node, int interfaceIndex)</p>

	Get unicast routing protocol type on this interface.
void	NetworkIpSetHsrpOnInterface (Node* node, int interfaceIndex) To enable hsrp on a interface
BOOL	NetworkIpIsHsrpEnabled (Node* node, int interfaceIndex) 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,

	<p>NodeAddress destinationAddress, int outgoingInterface, int networkType, BOOL* queueIsFull)</p> <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>
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 <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>
BOOL	<p><code>NetworkIpOutputQueuePeekWithIndex</code>(Node* node, int interfaceIndex, int msgIndex, Message** msg, NodeAddress* nextHopAddress, QueuePriorityType* priority)</p> <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>
BOOL	<p><code>NetworkIpOutputQueueTopPacketForAPriority</code>(Node* node, int interfaceIndex, QueuePriorityType priority, Message** msg, NodeAddress* nextHopAddress, int posInQueue)</p> <p>Same as <code>NetworkIpOutputQueueDequeuePacketForAPriority()</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_OutputQueueTopPacketForAPriority()</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>
BOOL	<p><code>NetworkIpInputQueueIsEmpty</code>(Node* node, int incomingInterface)</p> <p>Calls the packet scheduler for an interface to determine whether the interface's input queue is empty</p>
BOOL	<p><code>NetworkIpOutputQueueIsEmpty</code>(Node* node, int interfaceIndex)</p> <p>Calls the packet scheduler for an interface to determine whether the interface's output queue is empty.</p>
int	<p><code>NetworkIpOutputQueueNumberInQueue</code>(Node* node, int interfaceIndex, BOOL specificPriorityOnly, QueuePriorityType priority)</p> <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>
NodeAddress	<p><code>NetworkIpOutputQueueDropPacket</code>(Node* node, int interfaceIndex, Message** msg)</p> <p>Drop a packet from the queue.</p>
void	<p><code>NetworkIpDeleteOutboundPacketsToANode</code>(Node* node, const NodeAddress nextHopAddress, const NodeAddress destinationAddress, const BOOL returnPacketsToRoutingProtocol)</p> <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>
unsigned	<p><code>GetQueueNumberFromPriority</code>(TosType userTos, int numQueues)</p> <p>Get queue number through which a given user priority will be forwarded.</p>

QueuePriorityType	ReturnPriorityForPHB (Node* node, TosType tos) Returns the priority queue corresponding to the DS/TOS field.
void	NetworkGetInterfaceAndNextHopFromForwardingTable (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	NetworkGetInterfaceAndNextHopFromForwardingTable (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	NetworkGetInterfaceAndNextHopFromForwardingTable (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	NetworkGetInterfaceAndNextHopFromForwardingTable (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	NetworkIpGetInterfaceIndexForNextHop (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	NetworkGetInterfaceIndexForDestAddress (Node* node, NodeAddress destAddress) Get interface for the destination address.
NetworkRoutingAdminDistanceType	NetworkRoutingGetAdminDistance (Node* node, NetworkRoutingProtocolType type) Get the administrative distance of a routing protocol. These values don't quite match those recommended by Cisco.
void	NetworkInitForwardingTable (Node* node) Initialize the IP forwarding table, allocate enough memory for number of rows.
void	NetworkUpdateForwardingTable (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 fuction 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 grouplist
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, Int32 interfaceId, NodeAddress mcastAddr, clocktype delay, char filterMode, vector sourceList) To join a multicast group
void	NetworkIpJoinMulticastGroup (Node* node, NodeAddress mcastAddr, clocktype delay, char filterMode, vector sourceList) To join a multicast group
void	NetworkIpLeaveMulticastGroup (Node* node, NodeAddress mcastAddr, clocktype delay) To leave a multicast group
void	NetworkIpLeaveMulticastGroup (Node* node, NodeAddress mcastAddr, clocktype delay) To leave a multicast group
void	NetworkIpSetMulticastTimer (Node* node, long eventType, NodeAddress mcastAddr, clocktype delay) To set a multicast timer to join or leave multicast groups
void	NetworkIpSetMulticastRoutingProtocol (Node* node, void* multicastRoutingProtocol, int interfaceIndex) Assign a multicast routing protocol to an interface
void *	NetworkIpGetMulticastRoutingProtocol (Node* node, NetworkRoutingProtocolType routingProtocolType) To get the Multicast Routing Protocol structure
void	NetworkIpAddMulticastRoutingProtocolType (Node* node, NetworkRoutingProtocolType multicastProtocolType, int interfaceIndex) Assign a multicast protocol type to an interface
void	NetworkIpSetMulticastRouterFunction (Node* node, MulticastRouterFunctionType routerFunctionPtr, int interfaceIndex) Set a multicast router function to an interface
MulticastRouterFunctionType	NetworkIpGetMulticastRouterFunction (Node* node, int interfaceIndex) Get the multicast router function for an interface
void	NetworkIpUpdateMulticastRoutingProtocolAndRouterFunction (Node* node, NetworkRoutingProtocolType routingProtocolType, int interfaceIndex)

	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.
void	NetworkIpUpdateUnicastRoutingProtocolAndRouterFunction (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 forwarding 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	RouteThePacketUsingLookupTable (Node* node, Message* msg, int incomingInterface)
	Tries to route and send the packet using the node's forwarding table.
int	GetNetworkIPFragUnit (Node* node, int interfaceIndex)
	Returns the network ip fragmentation unit.
void	NetworkIpUserProtocolInit (Node* node, const NodeInput* nodeInput, const char* routingProtocolString, NetworkRoutingProtocolType* routingProtocolType, void** routingProtocolData)
	Initialization of user protocol(disabled)
void	NetworkIpUserHandleProtocolEvent (Node* node, Message* msg)
	Event handler function of user protocol(disabled)
void	NetworkIpUserHandleProtocolPacket (Node* node, Message* msg, unsigned char ipProtocol, NodeAddress sourceAddress, NodeAddress destinationAddress, int ttl)
	Process a user protocol generated control packet(disabled)
void	NetworkIpUserProtocolFinalize (Node* node, int userProtocolNumber)
	Finalization of user protocol(disabled)
void	Atm_RouteThePacketUsingLookupTable (Node* node, NodeAddress* destAddr, int* outIntf, NodeAddress* nextHop)
	Routing packet received at ATM node
void	RouteThePacketUsingMulticastForwardingTable (Node* node, Message* msg, int incomingInterface)
	Tries to route the multicast packet using the multicast forwarding table.
int	NETWORKIpRoutingInit (Node * node, const NodeInput *nodeInput nodeInput)

	Initialization function for network layer. Initializes IP.
Int64	NetworkIpGetBandwidth (Node* node, int interfaceIndex) getting the bandwidth information
clocktype	NetworkIpGetPropDelay (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 clasee 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_FLASHOVERRIDE 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_HAIPE 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 ROSPFS 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 ROSPFS 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_CES_EPLRS 175
	IP protocol numbers for EPLRS protocol for CES.
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	IpMfmcParameter

	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
Structure	AddressChangeType
	enumeration to define address change events by DHCP

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()	void IpHeaderSetVersion() (UInt32* ip_v_hl_tos_len, unsigned int version)

Set the value of version number for IpHeaderType	<p>Parameters:</p> <ul style="list-style-type: none">• <code>ip_v_hl_tos_len</code> - The variable containing the value of <code>ip_v</code>• <code>version</code> - Input value for set operation <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
IpHeaderSetHLen() Set the value of header length for IpHeaderType	<p>void IpHeaderSetHLen() (UInt32* <code>ip_v_hl_tos_len</code>, unsigned int <code>hlen</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>ip_v_hl_tos_len</code> - The variable containing the value of <code>ip_v</code>• <code>hlen</code> - Input value for set operation <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
IpHeaderSetTOS() Set the value of Type of Service for IpHeaderType	<p>void IpHeaderSetTOS() (UInt32* <code>ip_v_hl_tos_len</code>, unsigned int <code>ipTos</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>ip_v_hl_tos_len</code> - The variable containing the value of <code>ip_v</code>• <code>ipTos</code> - Input value for set operation <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
IpHeaderSetIpLength() Set the value of ip length for IpHeaderType	<p>void IpHeaderSetIpLength() (UInt32* <code>ip_v_hl_tos_len</code>, unsigned int <code>ipLen</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>ip_v_hl_tos_len</code> - The variable containing the value of <code>ip_v</code>• <code>ipLen</code> - Input value for set operation <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
IpHeaderSetIpFragOffset() Set the value of <code>ip_fragment_offset</code> for IpHeaderType	<p>void IpHeaderSetIpFragOffset() (UInt16* <code>ipFragment</code>, UInt16 <code>offset</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>ipFragment</code> - The variable containing the value of• <code>offset</code> - Input value for set operation

	Returns: <ul style="list-style-type: none">void - None
IpHeaderSetIpReserved() Set the value of ipReserved for IpHeaderType	void IpHeaderSetIpReserved() (UInt16* ipFragment, UInt16 ipReserved) Parameters: <ul style="list-style-type: none">ipFragment - The variable containing the value ofipReserved - Input value for set operation Returns: <ul style="list-style-type: none">void - None
IpHeaderSetIpDontFrag() Set the value of ip_dont_fragment for IpHeaderType	void IpHeaderSetIpDontFrag() (UInt16* ipFragment, UInt16 dontFrag) Parameters: <ul style="list-style-type: none">ipFragment - The variable containing the value ofdontFrag - Input value for set operation Returns: <ul style="list-style-type: none">void - None
IpHeaderSetIpMoreFrag() Set the value of ip_more_fragment for IpHeaderType	void IpHeaderSetIpMoreFrag() (UInt16* ipFragment, UInt16 moreFrag) Parameters: <ul style="list-style-type: none">ipFragment - The variable containing the value ofmoreFrag - Input value for set operation Returns: <ul style="list-style-type: none">void - None
IpHeaderGetVersion() Returns the value of version number for IpHeaderType	unsigned int IpHeaderGetVersion() (UInt32 ip_v_hl_tos_len) Parameters: <ul style="list-style-type: none">ip_v_hl_tos_len - The variable containing the value of ip_v Returns: <ul style="list-style-type: none">unsigned int - None
IpHeaderGetHLen() Returns the value of header length for IpHeaderType	unsigned int IpHeaderGetHLen() (UInt32 ip_v_hl_tos_len) Parameters: <ul style="list-style-type: none">ip v hl tos len - The variable containing the value of ip_v

	<p>Returns:</p> <ul style="list-style-type: none">• unsigned int - None
<p>IpHeaderGetTOS()</p> <p>Returns the value of Type of Service for IpHeaderType</p>	<p>unsigned int IpHeaderGetTOS() (UInt32 ip_v_hl_tos_len)</p> <p>Parameters:</p> <ul style="list-style-type: none">• ip_v_hl_tos_len - The variable containing the value of ip_v <p>Returns:</p> <ul style="list-style-type: none">• unsigned int - None
<p>IpHeaderGetIpLength()</p> <p>Returns the value of ip length for IpHeaderType</p>	<p>unsigned int IpHeaderGetIpLength() (UInt32 ip_v_hl_tos_len)</p> <p>Parameters:</p> <ul style="list-style-type: none">• ip_v_hl_tos_len - The variable containing the value of ip_v <p>Returns:</p> <ul style="list-style-type: none">• unsigned int - None
<p>IpHeaderGetIpFragOffset()</p> <p>Returns the value of ip_fragment_offset for IpHeaderType</p>	<p>UInt16 IpHeaderGetIpFragOffset() (UInt16 ipFragment)</p> <p>Parameters:</p> <ul style="list-style-type: none">• ipFragment - The variable containing the value of <p>Returns:</p> <ul style="list-style-type: none">• UInt16 - None
<p>IpHeaderGetIpDontFrag()</p> <p>Returns the value of ip_dont_fragment for IpHeaderType</p>	<p>BOOL IpHeaderGetIpDontFrag() (UInt16 ipFragment)</p> <p>Parameters:</p> <ul style="list-style-type: none">• ipFragment - The variable containing the value of <p>Returns:</p> <ul style="list-style-type: none">• BOOL - None
<p>IpHeaderGetIpMoreFrag()</p> <p>Returns the value of ip_more_fragment for IpHeaderType</p>	<p>BOOL IpHeaderGetIpMoreFrag() (UInt16 ipFragment)</p> <p>Parameters:</p> <ul style="list-style-type: none">• ipFragment - The variable containing the value of <p>Returns:</p> <ul style="list-style-type: none">• BOOL - None

IpHeaderGetIpReserved() Returns the value of ipReserved for IpHeaderType	BOOL IpHeaderGetIpReserved() (UInt16 ipFragment) Parameters: <ul style="list-style-type: none">ipFragment - The variable containing the value of Returns: <ul style="list-style-type: none">BOOL - None
Ip_timestampSetFlag() Set the value of flag for ip_timestamp_str	void Ip_timestampSetFlag() (unsigned char flgOflw, unsigned char flag) Parameters: <ul style="list-style-type: none">flgOflw - The variable containing the value of flag andflag - Input value for set operation Returns: <ul style="list-style-type: none">void - None
Ip_timestampSetOvflw() Set the value of ovflw for ip_timestamp_str	void Ip_timestampSetOvflw() (unsigned char flgOflw, unsigned char ovflw) Parameters: <ul style="list-style-type: none">flgOflw - The variable containing the value of flag andovflw - Input value for set operation Returns: <ul style="list-style-type: none">void - None
Ip_timestampGetFlag() Returns the value of flag for ip_timestamp_str	unsigned char Ip_timestampGetFlag() (unsigned char flgOflw) Parameters: <ul style="list-style-type: none">flgOflw - The variable containing the value of flag and Returns: <ul style="list-style-type: none">unsigned char - None
Ip_timestampGetOvflw() Returns the value of overflow counter for ip_timestamp_str	unsigned char Ip_timestampGetOvflw() (unsigned char flgOflw) Parameters: <ul style="list-style-type: none">flgOflw - The variable containing the value of flag and Returns: <ul style="list-style-type: none">unsigned char - None

ConvertNumHostBitsToSubnetMask To generate subnetmask using number of host bit	NodeAddress ConvertNumHostBitsToSubnetMask (int numHostBits) Parameters: <ul style="list-style-type: none">numHostBits - number of host bit. Returns: <ul style="list-style-type: none">NodeAddress - subnetmask
ConvertSubnetMaskToNumHostBits To generate number of host bit using subnetmask.	int ConvertSubnetMaskToNumHostBits (NodeAddress subnetMask) Parameters: <ul style="list-style-type: none">subnetMask - subnetmask. Returns: <ul style="list-style-type: none">int - number of host bit.
MaskIpAddress To mask a ip address.	NodeAddress MaskIpAddress (NodeAddress address, NodeAddress mask) Parameters: <ul style="list-style-type: none">address - address of a nodemask - mask of subnet. Returns: <ul style="list-style-type: none">NodeAddress - masked node address.
MaskIpAddressWithNumHostBits To mask a ip address.	NodeAddress MaskIpAddressWithNumHostBits (NodeAddress address, int numHostBits) Parameters: <ul style="list-style-type: none">address - address of a node.numHostBits - number of host bit. Returns: <ul style="list-style-type: none">NodeAddress - masked node address.
CalcBroadcastIpAddress To generate broadcast address.	NodeAddress CalcBroadcastIpAddress (NodeAddress address, int numHostBits) Parameters: <ul style="list-style-type: none">address - address of a node.numHostBits - number of host bit. Returns: <ul style="list-style-type: none">NodeAddress - Broadcast address.

IsIpAddressInSubnet To check if a ip address belongs to a subnet.	BOOL IsIpAddressInSubnet (NodeAddress address, NodeAddress subnetAddress, int numHostbits) Parameters: <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. Returns: <ul style="list-style-type: none">• <code>BOOL</code> - TRUE if ip address belongs to a subnet else FALSE.
NetworkIpAddHeader Add an IP packet header to a message. Just calls AddIpHeader.	void NetworkIpAddHeader (Node* node, Message* msg, NodeAddress sourceAddress, NodeAddress destinationAddress, TosType priority, unsigned char protocol, unsigned ttl) Parameters: <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 Returns: <ul style="list-style-type: none">• <code>void</code> - None
FindAnIpOptionField Searches the IP header for the option field with option code that matches optionKey, and returns a pointer to the option field header.	IpOptionsHeaderType* FindAnIpOptionField (const IpHeaderType* ipHeader, const int optionKey) Parameters: <ul style="list-style-type: none">• <code>ipHeader</code> - Pointer to an IP header.• <code>optionKey</code> - Option code for desired option field. Returns: <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	void NetworkIpPreInit (Node* node) Parameters:

<p>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.</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
<p>NetworkIpInit</p> <p>Initialize IP variables, and all network-layer IP protocols..</p>	<p>void NetworkIpInit (Node* node, const NodeInput* nodeInput)</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
<p>NetworkIpLayer</p> <p>Handle IP layer events, incoming messages and messages sent to itself (timers, etc.).</p>	<p>void NetworkIpLayer (Node* node, Message* msg)</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>void</code> - 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">• <code>node</code> - Pointer to node. <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - 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">• <code>node</code> - Pointer to node.• <code>msg</code> - Pointer to message from transport• <code>sourceAddress</code> - Source IP address.• <code>destinationAddress</code> - Destination IP address.• <code>outgoingInterface</code> - outgoing interface to use to

	<ul style="list-style-type: none">• <code>priority</code> - Priority of packet.• <code>protocol</code> - IP protocol number.• <code>isEcnCapable</code> - Is this node ECN capable? <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
<p>NetworkIpSendRawMessage</p> <p>Called by <code>NetworkIpReceivePacketFromTransportLayer()</code> 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 <code>RoutePacketAndSendToMac()</code>.</p>	<p><code>void</code> 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">• <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> - 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
<p>NetworkIpSendRawMessageWithDelay</p> <p>Same as <code>NetworkIpSendRawMessage()</code>, but schedules event after a simulation delay.</p>	<p><code>void</code> NetworkIpSendRawMessageWithDelay (Node* node, Message* msg, NodeAddress sourceAddress, NodeAddress destinationAddress, int outgoingInterface, TosType priority, unsigned char protocol, unsigned ttl, 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 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.

	<ul style="list-style-type: none">• <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
<p>NetworkIpSendRawMessageToMacLayer</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>	<p>void NetworkIpSendRawMessageToMacLayer (Node* node, Message* msg, NodeAddress sourceAddress, NodeAddress destinationAddress, TosType priority, unsigned char protocol, unsigned ttl, int interfaceIndex, 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 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. <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
<p>NetworkIpSendRawMessageToMacLayerWithDelay</p> <p>Same as <code>NetworkIpSendRawMessageToMacLayer()</code>, but schedules the event after a simulation delay by calling <code>NetworkIpSendPacketOnInterfaceWithDelay()</code>.</p>	<p>void NetworkIpSendRawMessageToMacLayerWithDelay (Node* node, Message* msg, NodeAddress sourceAddress, NodeAddress destinationAddress, TosType priority, unsigned char protocol, unsigned ttl, 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 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.

	<ul style="list-style-type: none">• <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
<p>NetworkIpSendPacketToMacLayer</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>	<p>void NetworkIpSendPacketToMacLayer (Node* node, Message* msg, int interfaceIndex, 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>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
<p>NetworkIpSendPacketOnInterface</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 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
<p>NetworkIpSendPacketToMacLayerWithDelay</p> <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.

	<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.• <code>delay</code> - delay <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
<p>NetworkIpSendPacketOnInterfaceWithDelay</p> <p>Same as <code>NetworkIpSendPacketOnInterface()</code>, but schedules event after a simulation delay.</p>	<p><code>void</code> NetworkIpSendPacketOnInterfaceWithDelay (Node* node, Message* msg, int incommingInterface, int outgoingInterface, 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>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
<p>NetworkIpSendRawPacketOnInterfaceWithDelay</p> <p>Same as <code>NetworkIpSendPacketOnInterface()</code>, but schedules event after a simulation delay and denotes raw packet.</p>	<p><code>void</code> NetworkIpSendRawPacketOnInterfaceWithDelay (Node* node, Message* msg, int incommingInterface, int outgoingInterface, 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>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
<p>NetworkIpSendPacketToMacLayerWithNewStrictSourceRoute</p>	<p><code>void</code> NetworkIpSendPacketToMacLayerWithNewStrictSourceRoute (Node* node, Message* msg,</p>

<p>Tacks on a new source route to an existing IP packet and then sends the packet to the MAC layer.</p>	<p>NodeAddress[] newRouteAddresses, int numNewRouteAddresses, BOOL removeExistingRecordedRoute)</p> <p>Parameters:</p> <ul style="list-style-type: none">• node - Pointer to node.• msg - Pointer to message with ip packet.• newRouteAddresses - Source route (address array).• numNewRouteAddresses - Number of array elements.• removeExistingRecordedRoute - Flag to indicate previous record <p>Returns:</p> <ul style="list-style-type: none">• void - None
<p>NetworkIpReceivePacketFromMacLayer</p> <p>IP received IP packet from MAC layer. Updates the Stats database and then calls NetworkIpReceivePacket.</p>	<p>void NetworkIpReceivePacketFromMacLayer (Node* node, Message* msg, NodeAddress previousHopNodeId, int interfaceIndex)</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
<p>NetworkIpReceivePacket</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>	<p>void NetworkIpReceivePacket (Node* node, Message* msg, NodeAddress previousHopNodeId, int interfaceIndex)</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
<p>NetworkIpNotificationOfPacketDrop</p>	<p>void NetworkIpNotificationOfPacketDrop (Node* node, Message* msg, NodeAddress nextHopNodeAddress, int interfaceIndex)</p>

<p>Invoke callback functions when a packet is dropped.</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>nextHopNodeAddress</code> - next hop address of dropped packet.• <code>interfaceIndex</code> - interface that experienced the packet drop. <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
<p>NetworkIpGetMacLayerStatusEventHandlerFunction</p> <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">• <code>node</code> - Pointer to node.• <code>interfaceIndex</code> - interface associated with the status <p>Returns:</p> <ul style="list-style-type: none">• <code>MacLayerStatusEventHandlerFunctionType</code> - Status event handler function.
<p>NetworkIpSetMacLayerStatusEventHandlerFunction</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>	<p>void NetworkIpSetMacLayerStatusEventHandlerFunction (Node* node, MacLayerStatusEventHandlerFunctionType StatusEventHandlerPtr, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>node</code> - Pointer to node.• <code>StatusEventHandlerPtr</code> - interface• <code>interfaceIndex</code> - interface associated with the status <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
<p>NetworkIpSneakPeekAtMacPacket</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>	<p>void NetworkIpSneakPeekAtMacPacket (Node* node, const Message* msg, int interfaceIndex, NodeAddress prevHop)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>node</code> - Pointer to node.• <code>msg</code> - The message being peeked at from the• <code>interfaceIndex</code> - The interface of which the "peeked" message belongs to.• <code>prevHop</code> - next hop address of dropped packet.

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

	<ul style="list-style-type: none">• <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>void NetworkIpSetMacLayerAckHandler (Node* node, MacLayerAckHandlerType macAckHandlerPtr, int interfaceIndex)</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">• <code>void</code> - None
<p>SendToUdp</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>	<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">• <code>node</code> - Pointer to node.• <code>msg</code> - Pointer to message with UDP packet.• <code>priority</code> - TOS of UDP packet.• <code>sourceAddress</code> - Source IP address.• <code>destinationAddress</code> - Destination IP address.• <code>incomingInterfaceIndex</code> - interface that received the packet <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
<p>SendToTcp</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>	<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">• <code>node</code> - Pointer to node.• <code>msg</code> - Pointer to message with TCP packet.• <code>priority</code> - TOS of TCP packet.• <code>sourceAddress</code> - Source IP address.

	<ul style="list-style-type: none">• <code>destinationAddress</code> - Destination IP address.• <code>aCongestionExperienced</code> - Determine if congestion is Returns: <ul style="list-style-type: none">• <code>void</code> - None
SendToRsvp 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 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">• <code>node</code> - Pointer to node.• <code>msg</code> - Pointer to message with RSVP packet.• <code>priority</code> - TOS of UDP packet.• <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 Removes the IP header from a message while also returning all the fields of the header.	<p>void NetworkIpRemoveIpHeader (Node* node, Message* msg, NodeAddress* sourceAddress, NodeAddress* destinationAddress, 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• <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	void AddIpOptionField (Node* node, Message* msg, int optionCode, int optionSize)

<p>Inserts an option field in the header of an IP packet.</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
<p>ExtractIpSourceAndRecordedRoute</p> <p>Retrieves a copy of the source and recorded route from the options field in the header.</p>	<p><code>void ExtractIpSourceAndRecordedRoute (Message* msg, NodeAddress[] RouteAddresses, int* NumAddresses, int* RouteAddressIndex)</code></p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>msg</code> - Pointer to message with IP packet.• <code>RouteAddresses</code> - Storage for source/recorded 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><code>RouterFunctionType NetworkIpGetRouterFunction (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 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. <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;</p>	<p><code>void NetworkIpSetRouterFunction (Node* node, RouterFunctionType RouterFunctionPtr, int interfaceIndex)</code></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>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 PacketWasRouted must be set to FALSE.</p>	<p>Returns:</p> <ul style="list-style-type: none">• void - 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">• 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
<p>NetworkIpAddUnicastIntraRegionRoutingProtocolType</p> <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
<p>NetworkIpGetRoutingProtocol</p> <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.
<p>NetworkIpGetUnicastRoutingProtocolType</p> <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. Returns: <ul style="list-style-type: none">• NetworkRoutingProtocolType - The unicast routing protocol type.
NetworkIpSetHsrpOnInterface To enable hsrp on a interface	void NetworkIpSetHsrpOnInterface (Node* node, int interfaceIndex) Parameters: <ul style="list-style-type: none">• node - Pointer to node.• interfaceIndex - network interface. Returns: <ul style="list-style-type: none">• void - None
NetworkIpIsHsrpEnabled To test if any interface is hsrp enabled.	BOOL NetworkIpIsHsrpEnabled (Node* node, int interfaceIndex) Parameters: <ul style="list-style-type: none">• node - Pointer to node.• interfaceIndex - network interface. Returns: <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.	void NetworkIpAddNewInterface (Node* node, NodeAddress interfaceIpAddress, int numHostBits, int* newInterfaceIndex, const NodeInput* nodeInput) Parameters: <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 Returns: <ul style="list-style-type: none">• void - None
NetworkIpInitCpuQueueConfiguration	void NetworkIpInitCpuQueueConfiguration (Node* node, const NodeInput* nodeInput) Parameters:

Initializes cpu queue parameters during startup.	<ul style="list-style-type: none">• <code>node</code> - Pointer to node.• <code>nodeInput</code> - Pointer to node input. Returns: <ul style="list-style-type: none">• <code>void</code> - None
NetworkIpInitInputQueueConfiguration Initializes input queue parameters during startup.	void NetworkIpInitInputQueueConfiguration (Node* node, const NodeInput* nodeInput, int interfaceIndex) Parameters: <ul style="list-style-type: none">• <code>node</code> - Pointer to node.• <code>nodeInput</code> - Pointer to node input.• <code>interfaceIndex</code> - interface associated with queue. Returns: <ul style="list-style-type: none">• <code>void</code> - None
NetworkIpInitOutputQueueConfiguration Initializes queue parameters during startup.	void NetworkIpInitOutputQueueConfiguration (Node* node, const NodeInput* nodeInput, int interfaceIndex) Parameters: <ul style="list-style-type: none">• <code>node</code> - Pointer to node.• <code>nodeInput</code> - Pointer to node input.• <code>interfaceIndex</code> - interface associated with queue. Returns: <ul style="list-style-type: none">• <code>void</code> - None
NetworkIpCreateQueues Initializes input and output queue parameters during startup	void NetworkIpCreateQueues (Node* node, const NodeInput* nodeInput, int interfaceIndex) Parameters: <ul style="list-style-type: none">• <code>node</code> - Pointer to node.• <code>nodeInput</code> - Pointer to node input.• <code>interfaceIndex</code> - interface associated with queue. Returns: <ul style="list-style-type: none">• <code>void</code> - None
NetworkIpSchedulerParameterInit	void NetworkIpSchedulerParameterInit (Scheduler* schedulerPtr, const int numPriorities, Queue* queue) Parameters:

<p>Initialize the scheduler parameters and also allocate memory for queues if require.</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
<p>NetworkIpSchedulerInit</p> <p>Call initialization function for appropriate scheduler.</p>	<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• interfaceIndex - interface index• schedulerPtr - type of Scheduler• schedulerTypeString - Scheduler name <p>Returns:</p> <ul style="list-style-type: none">• void - None
<p>NetworkIpCpuQueueInsert</p> <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>void NetworkIpCpuQueueInsert (Node* node, Message* msg, NodeAddress nextHopAddress, NodeAddress destinationAddress, int outgoingInterface, int networkType, BOOL* queueIsFull, int incomingInterface)</p> <p>Parameters:</p> <ul style="list-style-type: none">• node - Pointer to node.• msg - Pointer to message with IP packet.• nextHopAddress - Packet's next hop address.• destinationAddress - Packet's destination address.• outgoingInterface - Used to determine where packet• networkType - Type of network packet is using (IP,• queueIsFull - Storage for boolean indicator.• incomingInterface - Incoming interface of packet. <p>Returns:</p> <ul style="list-style-type: none">• void - None

<div><div>NetworkIpInputQueueInsert</div><div><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></div></div>	<div><div>void NetworkIpInputQueueInsert (Node* node, int incomingInterface, Message* msg, NodeAddress nextHopAddress, NodeAddress destinationAddress, int outgoingInterface, int networkType, BOOL* queueIsFull)</div><div><div>Parameters:</div><ul style="list-style-type: none">node - Pointer to node.incomingInterface - interface of input queue.msg - Pointer to message with IP packet.nextHopAddress - Packet's next hop address.destinationAddress - Packet's destination address.outgoingInterface - Used to determine where packetnetworkType - Type of network packet is using (IP,queueIsFull - Storage for boolean indicator.</div><div><div>Returns:</div><ul style="list-style-type: none">void - None</div></div>
<div><div>NetworkIpOutputQueueInsert</div><div><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></div></div>	<div><div>void NetworkIpOutputQueueInsert (Node* node, int interfaceIndex, Message* msg, NodeAddress nextHopAddress, NodeAddress destinationAddress, int networkType, BOOL* queueIsFull)</div><div><div>Parameters:</div><ul style="list-style-type: none">node - Pointer to node.interfaceIndex - interface of input queue.msg - Pointer to message with IP packet.nextHopAddress - Packet's next hop address.destinationAddress - Packet's destination address.networkType - Type of network packet is using (IP,queueIsFull - Storage for boolean indicator.</div><div><div>Returns:</div><ul style="list-style-type: none">void - None</div></div>
<div><div>NetworkIpInputQueueDequeuePacket</div><div><p>Calls the packet scheduler for an interface to retrieve an IP packet</p></div></div>	<div><div>BOOL NetworkIpInputQueueDequeuePacket (Node* node, int incomingInterface, Message** msg, NodeAddress* nextHopAddress, int* outgoingInterface, int* networkType, QueuePriorityType* priority)</div><div><div>Parameters:</div></div></div>

<p>from the input queue associated with the interface.</p>	<p>node - Pointer to node.</p> <ul style="list-style-type: none">incomingInterface - interface to dequeue from.msg - Storage for pointer to messagenextHopAddress - Storage for Packet'soutgoingInterface - Used to determine where packetnetworkType - Type of network packet is using (IP,priority - Storage for <p>Returns:</p> <ul style="list-style-type: none">BOOL - TRUE if dequeued successfully, FALSE otherwise.
<p>NetworkIpOutputQueueDequeuePacket</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>	<p>BOOL NetworkIpOutputQueueDequeuePacket (Node* node, int interfaceIndex, Message** msg, NodeAddress* nextHopAddress, int* networkType, QueuePriorityType* priority)</p> <p>Parameters:</p> <ul style="list-style-type: none">node - Pointer to node.interfaceIndex - index to interface .msg - Storage for pointer to messagenextHopAddress - Storage for Packet's next hop address.networkType - Type of network packet is using (IP,priority - Storage for priority <p>Returns:</p> <ul style="list-style-type: none">BOOL - TRUE if dequeued successfully, FALSE otherwise.
<p>NetworkIpOutputQueueDequeuePacketForAPriority</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>	<p>BOOL NetworkIpOutputQueueDequeuePacketForAPriority (Node* node, int interfaceIndex, QueuePriorityType priority, Message** msg, NodeAddress* nextHopAddress, int* networkType, int posInQueue)</p> <p>Parameters:</p> <ul style="list-style-type: none">node - Pointer to node.interfaceIndex - index to interface .priority - priority of packet.msg - Storage for pointer to messagenextHopAddress - Storage for Packet's next hop address.

	<ul style="list-style-type: none">• <code>networkType</code> - Type of network packet is using (IP,• <code>posInQueue</code> - Position of packet in Queue. Returns: <ul style="list-style-type: none">• <code>BOOL</code> - TRUE if dequeued successfully, FALSE otherwise.
<p>NetworkIpOutputQueueDequeuePacketWithIndex</p> <p>Same as <code>NetworkIpOutputQueueDequeuePacket()</code>, except the packet dequeued is requested by a specific index This function is called by <code>MAC_OutputQueueDequeuePacketForAPriority()</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 NetworkIpOutputQueueDequeuePacketWithIndex (Node* node, int interfaceIndex, int msgIndex, Message** msg, NodeAddress* nextHopAddress, int* networkType)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>node</code> - Pointer to node.• <code>interfaceIndex</code> - index to interface .• <code>msgIndex</code> - index of packet.• <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.
<p>NetworkIpInputQueueTopPacket</p> <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 (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 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.
<p>NetworkIpOutputQueueTopPacket</p>	<p>BOOL 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>	<p>Parameters:</p> <ul style="list-style-type: none">• node - Pointer to node.• interfaceIndex - index to interface .• msg - Storage for pointer to message• nextHopAddress - Storage for Packet's next hop addr.• networkType - Type of network of the packet• priority - Storage for priority <p>Returns:</p> <ul style="list-style-type: none">• BOOL - TRUE if there is a packet, FALSE otherwise.
<p>NetworkIpOutputQueuePeekWithIndex</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>	<p>BOOL NetworkIpOutputQueuePeekWithIndex (Node* node, int interfaceIndex, int msgIndex, Message** msg, NodeAddress* nextHopAddress, QueuePriorityType* priority)</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.
<p>NetworkIpOutputQueueTopPacketForAPriority</p> <p>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.</p>	<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 .• priority - priority of packet• msg - Storage for pointer to message• nextHopAddress - Storage for packet's next hop address.• posInQueue - Position of packet in Queue.

	<p>Returns:</p> <ul style="list-style-type: none">• <code>BOOL</code> - TRUE if there is a packet, FALSE otherwise.
<p>NetworkIpInputQueueIsEmpty</p> <p>Calls the packet scheduler for an interface to determine whether the interface's input queue is empty</p>	<p>BOOL NetworkIpInputQueueIsEmpty (Node* node, int incomingInterface)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>node</code> - Pointer to node.• <code>incomingInterface</code> - Index of interface. <p>Returns:</p> <ul style="list-style-type: none">• <code>BOOL</code> - 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.
<p>NetworkIpOutputQueueIsEmpty</p> <p>Calls the packet scheduler for an interface to determine whether the interface's output queue is empty.</p>	<p>BOOL NetworkIpOutputQueueIsEmpty (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>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.
<p>NetworkIpOutputQueueNumberInQueue</p> <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.
<p>NetworkIpOutputQueueDropPacket</p> <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 .

	<ul style="list-style-type: none">• <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.
<p>NetworkIpDeleteOutboundPacketsToANode</p> <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">• <code>void</code> - None
<p>GetQueueNumberFromPriority</p> <p>Get queue number through which a given user priority will be forwarded.</p>	<p>unsigned GetQueueNumberFromPriority (TosType userTos, int numQueues)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>userTos</code> - user TOS.• <code>numQueues</code> - Number of queues. <p>Returns:</p> <ul style="list-style-type: none">• <code>unsigned</code> - Index of the queue.
<p>ReturnPriorityForPHB</p> <p>Returns the priority queue corresponding to the DS/TOS field.</p>	<p>QueuePriorityType ReturnPriorityForPHB (Node* node, TosType tos)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>node</code> - Pointer to node.• <code>tos</code> - TOS field <p>Returns:</p> <ul style="list-style-type: none">• <code>QueuePriorityType</code> - priority queue
<p>NetworkGetInterfaceAndNextHopFromForwardingTable</p> <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>void NetworkGetInterfaceAndNextHopFromForwardingTable (Node* node, NodeAddress destinationAddress, int* interfaceIndex, NodeAddress* nextHopAddress)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>node</code> - Pointer to node.

	<p>destinationAddress - Destination IP address.</p> <ul style="list-style-type: none">• interfaceIndex - Storage for index of outgoing• nextHopAddress - Storage for next hop IP address. <p>Returns:</p> <ul style="list-style-type: none">• void - None
<p>NetworkGetInterfaceAndNextHopFromForwardingTable</p> <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>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.• 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
<p>NetworkGetInterfaceAndNextHopFromForwardingTable</p> <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>void NetworkGetInterfaceAndNextHopFromForwardingTable (Node* node, NodeAddress destinationAddress, int* interfaceIndex, NodeAddress* nextHopAddress, BOOL testType, NetworkRoutingProtocolType type)</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.• testType - Same protocol's routes if true• type - routing protocol type. <p>Returns:</p> <ul style="list-style-type: none">• void - None
<p>NetworkGetInterfaceAndNextHopFromForwardingTable</p>	<p>void NetworkGetInterfaceAndNextHopFromForwardingTable (Node* node, int operatingInterface, NodeAddress destinationAddress, int* interfaceIndex, NodeAddress* nextHopAddress, BOOL testType, NetworkRoutingProtocolType type)</p>

<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>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
<p>NetworkIpGetInterfaceIndexForNextHop</p> <p>This function looks at the network address of each of a node's network interfaces. When <code>nextHopAddress</code> is matched to a network, the interface index corresponding to the network is returned. (used by <code>NetworkUpdateForwardingTable()</code> and <code>ospfv2.pc</code>)</p>	<p>int NetworkIpGetInterfaceIndexForNextHop (Node* node, NodeAddress nextHopAddress)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>node</code> - Pointer to node.• <code>nextHopAddress</code> - Destination IP address. <p>Returns:</p> <ul style="list-style-type: none">• <code>int</code> - Index of outgoing interface, if <code>nextHopAddress</code> is on a directly connected network. -1, otherwise
<p>NetworkGetInterfaceIndexForDestAddress</p> <p>Get interface for the destination address.</p>	<p>int NetworkGetInterfaceIndexForDestAddress (Node* node, NodeAddress destAddress)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>node</code> - Pointer to node.• <code>destAddress</code> - Destination IP address. <p>Returns:</p> <ul style="list-style-type: none">• <code>int</code> - interface index associated with destination.
<p>NetworkRoutingGetAdminDistance</p> <p>Get the administrative distance of a routing protocol. These values don't quite match those recommended by Cisco.</p>	<p>NetworkRoutingAdminDistanceType NetworkRoutingGetAdminDistance (Node* node, NetworkRoutingProtocolType type)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>node</code> - Pointer to node.• <code>type</code> - Type value of routing protocol. <p>Returns:</p>

	<ul style="list-style-type: none">• <code>NetworkRoutingAdminDistanceType</code> - The administrative distance of the routing protocol.
<p>NetworkInitForwardingTable</p> <p>Initialize the IP forwarding table, allocate enough memory for number of rows.</p>	<p><code>void NetworkInitForwardingTable (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
<p>NetworkUpdateForwardingTable</p> <p>Update or add entry to IP routing table. Search the routing table for an entry with an exact match for <code>destAddress</code>, <code>destAddressMask</code>, and routing protocol. Update this entry with the specified <code>nextHopAddress</code> (the outgoing interface is automatically determined from the <code>nextHopAddress</code> -- see code). If no matching entry found, then add a new route.</p>	<p><code>void NetworkUpdateForwardingTable (Node* node, NodeAddress destAddress, NodeAddress destAddressMask, NodeAddress nextHopAddress, int outgoingInterfaceIndex, int cost, NetworkRoutingProtocolType type)</code></p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>node</code> - Pointer to node.• <code>destAddress</code> - IP address of destination• <code>destAddressMask</code> - Netmask.• <code>nextHopAddress</code> - Next hop IP address.• <code>outgoingInterfaceIndex</code> - outgoing interface.• <code>cost</code> - Cost metric associated with• <code>type</code> - type value of <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
<p>NetworkRemoveForwardingTableEntry</p> <p>Remove single entries in the routing table</p>	<p><code>void NetworkRemoveForwardingTableEntry (Node* node, NodeAddress destAddress, NodeAddress destAddressMask, NodeAddress nextHopAddress, int outgoingInterfaceIndex)</code></p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>node</code> - Pointer to node.• <code>destAddress</code> - IP address of destination• <code>destAddressMask</code> - Netmask.• <code>nextHopAddress</code> - Next hop IP address.• <code>outgoingInterfaceIndex</code> - outgoing interface. <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None

NetworkEmptyForwardingTable Remove entries in the routing table corresponding to a given routing protocol.	void NetworkEmptyForwardingTable (Node* node, NetworkRoutingProtocolType type) Parameters: <ul style="list-style-type: none"> • <code>node</code> - Pointer to node. • <code>type</code> - Type of routing protocol whose Returns: <ul style="list-style-type: none"> • <code>void</code> - None
NetworkPrintForwardingTable Display all entries in node's routing table.	void NetworkPrintForwardingTable (Node* node) Parameters: <ul style="list-style-type: none"> • <code>node</code> - Pointer to node. Returns: <ul style="list-style-type: none"> • <code>void</code> - None
NetworkGetMetricForDestAddress Get the cost metric for a destination from the forwarding table.	int NetworkGetMetricForDestAddress (Node* node, NodeAddress destAddress) Parameters: <ul style="list-style-type: none"> • <code>node</code> - Pointer to node. • <code>destAddress</code> - destination to get cost metric from. Returns: <ul style="list-style-type: none"> • <code>int</code> - Cost metric associated with destination.
NetworkIpSetRouteUpdateEventFunction Set a callback fuction when a route changes from forwarding table.	void NetworkIpSetRouteUpdateEventFunction (Node* node, NetworkRouteUpdateEventType routeUpdateFunctionPtr) Parameters: <ul style="list-style-type: none"> • <code>node</code> - Pointer to node. • <code>routeUpdateFunctionPtr</code> - Route update Returns: <ul style="list-style-type: none"> • <code>void</code> - None
NetworkIpGetRouteUpdateEventFunction Print packet headers when packet tracing is enabled.	NetworkRouteUpdateEventType NetworkIpGetRouteUpdateEventFunction (Node* node) Parameters: <ul style="list-style-type: none"> • <code>node</code> - Pointer to node. Returns:

	<ul style="list-style-type: none">• <code>NetworkRouteUpdateEventType</code> - Route update callback function to set.
NetworkIpGetInterfaceAddress Get the interface address on this interface	<p>NodeAddress NetworkIpGetInterfaceAddress (Node* node, int interfaceIndex)</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> - IP address associated with the interface
NetworkIpGetInterfaceName To get the interface name associated with the interface	<p>char* NetworkIpGetInterfaceName (Node* node, int interfaceIndex)</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>char*</code> - interface name
NetworkIpGetInterfaceNetworkAddress To get network address associated with interface	<p>NodeAddress NetworkIpGetInterfaceNetworkAddress (Node* node, int interfaceIndex)</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 To retrieve subnet mask of the node interface	<p>NodeAddress NetworkIpGetInterfaceSubnetMask (Node* node, int interfaceIndex)</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 Get the number of host bits on this interface	<p>int NetworkIpGetInterfaceNumHostBits (Node* node, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>node</code> - 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>int</code> - Number of host bits on the specified interface
<p>NetworkIpGetInterfaceBroadcastAddress</p> <p>Get broadcast address on this interface</p>	<p>NodeAddress NetworkIpGetInterfaceBroadcastAddress (Node* node, int interfaceIndex)</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
<p>IsOutgoingBroadcast</p> <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">• <code>node</code> - Pointer to the node• <code>destAddress</code> - IP packet's destination IP address.• <code>outgoingInterface</code> - Outgoing interface index.• <code>outgoingBroadcastAddress</code> - Broadcast address <p>Returns:</p> <ul style="list-style-type: none">• <code>BOOL</code> - Returns true if destination is broadcast address
<p>NetworkIpIsMyIP</p> <p>In turn calls IsMyPacket()</p>	<p>BOOL NetworkIpIsMyIP (Node* node, NodeAddress ipAddress)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>node</code> - Pointer to the node• <code>ipAddress</code> - An IP packet's destination IP address. <p>Returns:</p> <ul style="list-style-type: none">• <code>BOOL</code> - Returns if it belongs to it.
<p>NetworkIpConfigurationError</p> <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">• <code>node</code> - Pointer to the node• <code>parameterName</code> - Error message to print

	<ul style="list-style-type: none">• <code>interfaceIndex</code> - interface number <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
<p>NetworkPrintIpHeader</p> <p>To print the IP header</p>	<p><code>void</code> NetworkPrintIpHeader (<code>Message*</code> msg)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>msg</code> - Pointer to Message <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
<p>NetworkIpAddToMulticastGroupList</p> <p>Add a specified node to a multicast group</p>	<p><code>void</code> NetworkIpAddToMulticastGroupList (<code>Node*</code> node, <code>NodeAddress</code> 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>NetworkIpRemoveFromMulticastGroupList</p> <p>To remove specified node from a multicast group</p>	<p><code>void</code> NetworkIpRemoveFromMulticastGroupList (<code>Node*</code> node, <code>NodeAddress</code> 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 grouplist</p>	<p><code>void</code> NetworkIpPrintMulticastGroupList (<code>Node*</code> 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><code>BOOL</code> NetworkIpIsPartOfMulticastGroup (<code>Node*</code> node, <code>NodeAddress</code> groupAddress)</p> <p>Parameters:</p>

check if a node is part of specified multicast group	<ul style="list-style-type: none">• <code>node</code> - Pointer to the node• <code>groupAddress</code> - group to check if node is part of Returns: <ul style="list-style-type: none">• <code>BOOL</code> - None
NetworkIpJoinMulticastGroup To join a multicast group	void NetworkIpJoinMulticastGroup (Node* node, NodeAddress mcastAddr, clocktype delay) Parameters: <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 Returns: <ul style="list-style-type: none">• <code>void</code> - None
NetworkIpJoinMulticastGroup To join a multicast group	void NetworkIpJoinMulticastGroup (Node* node, Int32 interfaceId, NodeAddress mcastAddr, clocktype delay, char filterMode, vector sourceList) Parameters: <ul style="list-style-type: none">• <code>node</code> - Pointer to the node• <code>interfaceId</code> - on which interface to join the group• <code>mcastAddr</code> - multicast group address• <code>delay</code> - delay after which to join• <code>filterMode</code> - filter mode of the interface (specific to IGMP version 3)• <code>sourceList</code> - list of sources from where multicast traffic Returns: <ul style="list-style-type: none">• <code>void</code> - None
NetworkIpJoinMulticastGroup To join a multicast group	void NetworkIpJoinMulticastGroup (Node* node, NodeAddress mcastAddr, clocktype delay, char filterMode, vector sourceList) Parameters: <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• <code>filterMode</code> - filter mode of the interface (specific to IGMP version 3)

	<ul style="list-style-type: none">• <code>sourceList</code> - list of sources from where multicast traffic Returns: <ul style="list-style-type: none">• <code>void</code> - None
NetworkIpLeaveMulticastGroup To leave a multicast group	void NetworkIpLeaveMulticastGroup (Node* node, NodeAddress mcastAddr, clocktype delay) Parameters: <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 leave Returns: <ul style="list-style-type: none">• <code>void</code> - None
NetworkIpLeaveMulticastGroup To leave a multicast group	void NetworkIpLeaveMulticastGroup (Node* node, NodeAddress mcastAddr, clocktype delay) Parameters: <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 leave Returns: <ul style="list-style-type: none">• <code>void</code> - None
NetworkIpSetMulticastTimer To set a multicast timer to join or leave multicast groups	void NetworkIpSetMulticastTimer (Node* node, long eventType, NodeAddress mcastAddr, clocktype delay) Parameters: <ul style="list-style-type: none">• <code>node</code> - Pointer to the node• <code>eventType</code> - the event type• <code>mcastAddr</code> - multicast group address• <code>delay</code> - delay after which to leave Returns: <ul style="list-style-type: none">• <code>void</code> - None
NetworkIpSetMulticastRoutingProtocol	void NetworkIpSetMulticastRoutingProtocol (Node* node, void* multicastRoutingProtocol, int interfaceIndex)

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

NetworkIpGetMulticastRouterFunction Get the multicast router function for an interface	<p>MulticastRouterFunctionType NetworkIpGetMulticastRouterFunction (Node* node, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none">node - Pointer to this nodeinterfaceIndex - interface number of the node <p>Returns:</p> <ul style="list-style-type: none">MulticastRouterFunctionType - Multicast router function on this interface.
NetworkIpUpdateMulticastRoutingProtocolAndRouterFunction 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.	<p>void NetworkIpUpdateMulticastRoutingProtocolAndRouterFunction (Node* node, NetworkRoutingProtocolType routingProtocolType, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none">node - this noderoutingProtocolType - multicast routinginterfaceIndex - interface index. <p>Returns:</p> <ul style="list-style-type: none">void - None
NetworkIpUpdateUnicastRoutingProtocolAndRouterFunction 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.	<p>void NetworkIpUpdateUnicastRoutingProtocolAndRouterFunction (Node* node, NetworkRoutingProtocolType routingProtocolType, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none">node - this noderoutingProtocolType - unicast routinginterfaceIndex - interface associated with unicast protocol. <p>Returns:</p> <ul style="list-style-type: none">void - None
NetworkIpGetInterfaceIndexFromAddress Get the interface index from an IP address.	<p>int NetworkIpGetInterfaceIndexFromAddress (Node* node, NodeAddress address)</p> <p>Parameters:</p> <ul style="list-style-type: none">node - this nodeaddress - address to determine interface index for <p>Returns:</p> <ul style="list-style-type: none">int - interface index associated with specified address.
NetworkIpGetInterfaceIndexFromSubnetAddress	<p>int NetworkIpGetInterfaceIndexFromSubnetAddress (Node* node, NodeAddress address)</p>

Get the interface index from an IP subnet address.	<p>Parameters:</p> <ul style="list-style-type: none">• <code>node</code> - this node• <code>address</code> - subnet address to determine interface <p>Returns:</p> <ul style="list-style-type: none">• <code>int</code> - interface index associated with specified subnet address.
NetworkIpIsMulticastAddress Check if an address is a multicast address.	<p>BOOL NetworkIpIsMulticastAddress (Node* node, NodeAddress address)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>node</code> - this node• <code>address</code> - address to determine if multicast address. <p>Returns:</p> <ul style="list-style-type: none">• <code>BOOL</code> - TRUE if address is multicast address, FALSE, otherwise.
NetworkInitMulticastForwardingTable initialize the multicast forwarding table, allocate enough memory for number of rows, used by ip	<p>void NetworkInitMulticastForwardingTable (Node* node)</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
NetworkEmptyMulticastForwardingTable empty out all the entries in the multicast forwarding table. basically set the size of table back to 0.	<p>void NetworkEmptyMulticastForwardingTable (Node* node)</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
NetworkGetOutgoingInterfaceFromMulticastForwardingTable get the interface Id node that lead to the (source, multicast group) pair.	<p>LinkedList* NetworkGetOutgoingInterfaceFromMulticastForwardingTable (Node* node, NodeAddress sourceAddress, NodeAddress groupAddress)</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)
<p>NetworkUpdateMulticastForwardingTable</p> <p>update entry with(sourceAddress,multicastGroupAddress) pair. search for the row with(sourceAddress,multicastGroupAddress) and update its interface.</p>	<p>void NetworkUpdateMulticastForwardingTable (Node* node, NodeAddress sourceAddress, NodeAddress multicastGroupAddress, int interfaceIndex)</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
<p>NetworkPrintMulticastForwardingTable</p> <p>display all entries in multicast forwarding table of the node.</p>	<p>void NetworkPrintMulticastForwardingTable (Node* node)</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
<p>NetworkPrintMulticastOutgoingInterface</p> <p>Print mulitcast outgoing interfaces.</p>	<p>void NetworkPrintMulticastOutgoingInterface (Node* node, list* list)</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
<p>NetworkInMulticastOutgoingInterface</p> <p>Determine if interface is in multicast outgoing interface list.</p>	<p>BOOL NetworkInMulticastOutgoingInterface (Node* node, List* list, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>node</code> - this node• <code>list</code> - list of outgoing interfaces.• <code>interfaceIndex</code> - interface to determine if in outgoing <p>Returns:</p>

	<ul style="list-style-type: none"> • <code>BOOL</code> - <code>TRUE</code> if interface is in multicast outgoing interface list, <code>FALSE</code> otherwise.
NetworkIpPrintTraceXML Print packet trace information in XML format.	void NetworkIpPrintTraceXML (Node* node, Message* msg) Parameters: <ul style="list-style-type: none"> • <code>node</code> - this node • <code>msg</code> - Packet to print headers from. Returns: <ul style="list-style-type: none"> • <code>void</code> - None
RouteThePacketUsingLookupTable Tries to route and send the packet using the node's forwarding table.	void RouteThePacketUsingLookupTable (Node* node, Message* msg, int incomingInterface) Parameters: <ul style="list-style-type: none"> • <code>node</code> - this node • <code>msg</code> - Pointer to message with IP packet. • <code>incomingInterface</code> - incoming interface of packet Returns: <ul style="list-style-type: none"> • <code>void</code> - <code>NULL</code>
GetNetworkIPFragUnit Returns the network ip fragmentation unit.	int GetNetworkIPFragUnit (Node* node, int interfaceIndex) Parameters: <ul style="list-style-type: none"> • <code>node</code> - this node • <code>interfaceIndex</code> - interface of node Returns: <ul style="list-style-type: none"> • <code>int</code> - None
NetworkIpUserProtocolInit Initialization of user protocol(disabled)	void NetworkIpUserProtocolInit (Node* node, const NodeInput* nodeInput, const char* routingProtocolString, NetworkRoutingProtocolType* routingProtocolType, void** routingProtocolData) Parameters: <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
<p>NetworkIpUserHandleProtocolEvent</p> <p>Event handler function of user protocol(disabled)</p>	<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
<p>NetworkIpUserHandleProtocolPacket</p> <p>Process a user protocol generated control packet(disabled)</p>	<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
<p>NetworkIpUserProtocolFinalize</p> <p>Finalization of user protocol(disabled)</p>	<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"><code>void</code> - None
<p>Atm_RouteThePacketUsingLookupTable</p> <p>Routing packet received at ATM node</p>	<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
<p>RouteThePacketUsingMulticastForwardingTable</p> <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.
<p>NETWORKIpRoutingInit</p> <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
<p>NetworkIpGetBandwidth</p> <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.• interfaceIndex - interface Index. <p>Returns:</p> <ul style="list-style-type: none">• Int64 - 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.
<p>NetworkIpGetPropDelay</p>	<p>clocktype NetworkIpGetPropDelay (Node* node, int interfaceIndex)</p> <p>Parameters:</p>

getting the propagation delay information	<ul style="list-style-type: none">• <code>node</code> - the node who's bandwidth is needed.• <code>interfaceIndex</code> - interface Index. Returns: <ul style="list-style-type: none">• <code>clocktype</code> - propagation delay ASSUMPTION : Array is exactly 3-byte long.
NetworkIpInterfaceIsEnabled To check the interface is enabled or not?	BOOL NetworkIpInterfaceIsEnabled (Node* node, int interfaceIndex) Parameters: <ul style="list-style-type: none">• <code>node</code> - node structure pointer.• <code>interfaceIndex</code> - interface Index. Returns: <ul style="list-style-type: none">• <code>BOOL</code> - None
NetworkIpIsWiredNetwork Determines if an interface is a wired interface.	BOOL NetworkIpIsWiredNetwork (Node* node, int interfaceIndex) Parameters: <ul style="list-style-type: none">• <code>node</code> - node structure pointer.• <code>interfaceIndex</code> - interface Index. Returns: <ul style="list-style-type: none">• <code>BOOL</code> - None
NetworkIpIsPointToPointNetwork Determines if an interface is a point-to-point.	BOOL NetworkIpIsPointToPointNetwork (Node* node, int interfaceIndex) Parameters: <ul style="list-style-type: none">• <code>node</code> - node structure pointer.• <code>interfaceIndex</code> - interface Index. Returns: <ul style="list-style-type: none">• <code>BOOL</code> - None
IsIPV4MulticastEnabledOnInterface To check if IPV4 Multicast is enabled on interface?	BOOL IsIPV4MulticastEnabledOnInterface (Node* node, int interfaceIndex) Parameters: <ul style="list-style-type: none">• <code>node</code> - node structure pointer.• <code>interfaceIndex</code> - interface Index. Returns:

	<ul style="list-style-type: none">• <code>BOOL</code> - None
IsIPv4RoutingEnabledOnInterface To check if IPV4 Routing is enabled on interface?	BOOL IsIPv4RoutingEnabledOnInterface (Node* node, int interfaceIndex) Parameters: <ul style="list-style-type: none">• <code>node</code> - node structure pointer.• <code>interfaceIndex</code> - interface Index. Returns: <ul style="list-style-type: none">• <code>BOOL</code> - None
NetworkIpGetNetworkProtocolType Get Network Protocol Type for the node	NetworkProtocolType NetworkIpGetNetworkProtocolType (Node* node, NodeAddress nodeId) Parameters: <ul style="list-style-type: none">• <code>node</code> - node structure pointer.• <code>nodeId</code> - node id. Returns: <ul style="list-style-type: none">• <code>NetworkProtocolType</code> - None
ResolveNetworkTypeFromSrcAndDestNodeId Resolve the NetworkType from source and destination node id's.	NetworkType ResolveNetworkTypeFromSrcAndDestNodeId (Node* node, NodeId sourceNodeId, NodeId destNodeId) Parameters: <ul style="list-style-type: none">• <code>node</code> - Pointer to the node.• <code>sourceNodeId</code> - Source node id.• <code>destNodeId</code> - Destination node id. Returns: <ul style="list-style-type: none">• <code>NetworkType</code> - None
NetworkIpIsWiredBroadcastNetwork Determines if an interface is a wired interface.	BOOL NetworkIpIsWiredBroadcastNetwork (Node* node, int interfaceIndex) Parameters: <ul style="list-style-type: none">• <code>node</code> - node structure pointer.• <code>interfaceIndex</code> - interface Index. Returns: <ul style="list-style-type: none">• <code>BOOL</code> - None
FindTraceRouteOption	<code>ip_traceroute*</code> FindTraceRouteOption (const IpHeaderType* ipHeader)

Searches the IP header for the Traceroute option field , and returns a pointer to traceroute header.

Parameters:

- `ipHeader` - Pointer to an IP header.

Returns:

- `ip_traceroute*` - 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 © 2001-2013 [SCALABLE Network Technologies, Inc.](#) All rights reserved.

QualNet 7.1 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 Lenght.
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.
CONSTANT	IP6_NHDR_IPCP
	Compression IPv6 Next header field value.
CONSTANT	IP6_NHDR_OSPF
	Compression IPv6 Next header field value.
CONSTANT	IP6_NHDR_DOPT
	Destination IPv6 Next header field value.
CONSTANT	IP6_NHDR_NONH
	No next header IPv6 Next header field value.
CONSTANT	IPV6_FLOWINFO_VERSION
	Flow infromation version.
CONSTANT	IPV6_VERSION
	IPv6 version no.
CONSTANT	IP6_MMTU
	Minimal MTU and reassembly.
CONSTANT	IPPROTO_ICMPV6
	ICMPv6 protocol no.
CONSTANT	IP6ANY_ANYCAST
	IPv6 anycast.
CONSTANT	ND_DEFAULT_HOPLIM
	Node Discovery hop count.
CONSTANT	IP6_INSOPT_NOALLOC

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

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

	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	IPV6_HEADER_LENGTH Length of IPv6 header
CONSTANT	IP6_LSRR 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	<u>IPV6_SET_CLASS(hdr, priority)</u> Sets the flow class.
MACRO	<u>IPV6_GET_CLASS(hdr)</u> Gets the flow class.
void	<u>ip6_hdrSetVersion()</u> (UInt32 ipv6HdrVcf, UInt32 version) Set the value of version for ip6_hdr
void	<u>ip6_hdrSetClass()</u> (UInt32 ipv6HdrVcf, unsigned char ipv6Class) Set the value of class for ip6_hdr
void	<u>ip6_hdrSetFlow()</u> (UInt32 ipv6HdrVcf, UInt32 flow) Set the value of flow for ip6_hdr
UInt32	<u>ip6_hdrGetVersion()</u> (unsigned int ipv6HdrVcf) Returns the value of version for ip6_hdr
UInt32	<u>ip6_hdrGetClass()</u> (unsigned int ipv6HdrVcf) Returns the value of ip6_class for ip6_hdr
UInt32	<u>ip6_hdrGetFlow()</u> (unsigned int ipv6HdrVcf) Returns the value of ip6_flow for ip6_hdr
int	<u>in6_isanycast</u> (Node* node, in6_addr addr) Checks whether the address is anycast address of the node.
None	<u>Ipv6AddIpv6Header</u> (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	<u>Ipv6AddFragmentHeader</u> (Node *node node, Message *msg msg, unsigned char nextHeader, unsigned short offset, unsigned int id)

	Adds fragment header
None	Ipv6RemoveIpv6Header (Node *node node, Message *msg msg, Address* sourceAddress, Address* 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 incomingInterface, 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 incomingInterface, 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)

	<p>Called by NetworkIpReceivePacketFromTransportLayer() to send to send UDP datagrams using IPv6. This function adds an IPv6 header and calls RoutePacketAndSendToMac().</p>
None	<p>Ipv6SendToUdp(Node* node, Message* msg, TosType priority, Address sourceAddress, Address destinationAddress, int incomingInterfaceIndex)</p> <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>
None	<p>Ipv6SendToTCP(Node* node, Message* msg, TosType priority, Address sourceAddress, Address destinationAddress, int incomingInterfaceIndex)</p> <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>
None	<p>Ipv6ReceivePacketFromMacLayer(Node* node, Message* msg, NodeAddress previousHopNodeId, int interfaceIndex)</p> <p>IPv6 received IPv6 packet from MAC layer. Determine whether the packet is to be delivered to this node, or needs to be forwarded.</p>
BOOL	<p>Ipv6AddMessageInBuffer(Node* node, Message* msg, in6_addr* nextHopAddr, int inCommingInterface)</p> <p>Adds an ipv6 packet in message in the hold buffer</p>
BOOL	<p>Ipv6DeleteMessageInBuffer(Node* node, messageBuffer* mBuf)</p> <p>Delets an ipv6 packet in the hold buffer</p>
void	<p>Ipv6DropMessageFromBuffer(Node* node, messageBuffer* mBuf)</p> <p>Drops an ipv6 packet from the hold buffer</p>
NetworkType	<p>Ipv6GetAddressTypeFromString(char* interfaceAddr)</p> <p>Returns network type from string ip address.</p>
IPv6 multicast address	<p>Ipv6GetInterfaceMulticastAddress(Node* node, int interfaceIndex)</p> <p>Get multicast address of this interface</p>
None	<p>Ipv6SolicitationMulticastAddress(in6_addr* dst_addr, in6_addr* target)</p> <p>Copies multicast solicitation address.</p>
None	<p>Ipv6AllRoutersMulticastAddress(in6_addr* dst dst)</p>

	Function to assign all routers multicast address.
None	IPv6GetLinkLocalAddress (node, int interface, in6_addr* addr) Gets ipv6 link local address of the interface in output parameter addr.
None	IPv6GetSiteLocalAddress (node, int interface, in6_addr* addr) Gets ipv6 site local address of the interface in output parameter addr.
None	IPv6GetSiteLocalAddress (node, int interface, in6_addr* addr) Gets ipv6 global agreeable address of the interface in output parameter addr.
None	Ipv6GetPrefix (in6_addr* addr, in6_addr* prefix) Gets ipv6 prefix from address.
Prefix for this interface	IPv6GetPrefixFromInterfaceIndex (Node* node, int interfaceIndex) Gets ipv6 prefix from address.
BOOL	Ipv6OutputQueueIsEmpty (Node *node node, int interfaceIndex) Check weather output queue is empty
None	Ipv6RoutingStaticInit (Node *node node, const NodeInput nodeInput, NetworkRoutingProtocolType type) Ipv6 Static routing initialization function.
None	Ipv6RoutingStaticEntry (Node *node node, char currentLine[] currentLine) Static routing route entry function
None	Ipv6AddDestination (Node* node node, route* ro ro) Adds destination in the destination cache.
None	Ipv6DeleteDestination (Node* node node) Deletes destination from the destination cache.
int	Ipv6CheckForValidPacket (Node* node node, SchedulerType* scheduler scheduler, unsigned int* pIndex pIndex) Checks the packet's validity

None	<div><div>Ipv6NdpProcessing(Node* node node)</div><div>Ipv6 Destination cache and neighbor cache : processing function</div></div>
None	<div><div>Ipv6UpdateForwardingTable(Node* node node, in6_addr destPrefix destPrefix, in6_addr nextHopPrefix nextHopPrefix, int interfaceIndex, int metric metric)</div><div>Updates Ipv6 Forwarding Table</div></div>
None	<div><div>Ipv6EmptyForwardingTable(Node* node node, NetworkRoutingProtocolType type type)</div><div>Empties Ipv6 Forwarding Table for a particular routing protocol entry</div></div>
None	<div><div>Ipv6PrintForwardingTable(Node* node node)</div><div>Prints the forwarding table.</div></div>
Interface index associated with specified subnet address.	<div><div>Ipv6InterfaceIndexFromSubnetAddress(Node* node node, in6_addr* address)</div><div>Get the interface index from an IPv6 subnet address.</div></div>
void	<div><div>Ipv6GetInterfaceAndNextHopFromForwardingTable(Node* node node, in6_addr destAddr, int* interfaceIndex, in6_addr* nextHopAddr)</div><div>Do a lookup on the routing table with a destination IPv6 address to obtain an outgoing interface and a next hop Ipv6 address.</div></div>
interface index associated with destination.	<div><div>Ipv6GetInterfaceIndexForDestAddress(Node* node node, in6_addr destAddr)</div><div>Get interface for the destination address.</div></div>
interface index associated with destination.	<div><div>Ipv6GetMetricForDestAddress(Node* node node, in6_addr destAddr)</div><div>Get the cost metric for a destination from the forwarding table.</div></div>
Interface index associated with destination if found,	<div><div>Ipv6IpGetInterfaceIndexForNextHop(Node* node node, in6_addr destAddr)</div><div>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.</div></div>
Ipv6RouterFunctionType	<div><div>Ipv6GetRouterFunction(Node* node, int interfaceIndex)</div><div>Get the router function pointer.</div></div>
void	<div><div>Ipv6SendPacketToMacLayer(Node* node node, Message* msg, in6_addr destAddr, in6_addr* nextHopAddr, int* interfaceIndex)</div></div>

	Used if IPv6 next hop address and outgoing interface is known.
void	Ipv6JoinMulticastGroup (Node* node, in6_addr mcastAddr, clocktype delay) Join a multicast group.
void	Ipv6AddToMulticastGroupList (Node* node, in6_addr groupAddress) Add group to multicast group list.
void	Ipv6LeaveMulticastGroup (Node* node, in6_addr mcastAddr) Leave a multicast group.
void	Ipv6RemoveFromMulticastGroupList (Node* node, in6_addr groupAddress) Remove group from multicast group list.
void	Ipv6NotificationOfPacketDrop (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,	Ipv6IsPartOfMulticastGroup (Node* node, Message* msg, in6_addr groupAddress) Check if destination is part of the multicast group.
TRUE if reserved multicast address, FALSE otherwise.	Ipv6IsReservedMulticastAddress (Node* node, in6_addr mcastAddr) Check if address is reserved multicast address.
TRUE if interface is in multicast outgoing interface	Ipv6InMulticastOutgoingInterface (Node* node, LinkedList* list, int interfaceIndex) Determine if interface is in multicast outgoing interface list.
void	Ipv6UpdateMulticastForwardingTable (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.	Ipv6GetOutgoingInterfaceFromMulticastTable (Node* node, in6_addr sourceAddress, in6_addr groupAddress) get the interface List that lead to the (source, multicast group) pair.
void	Ipv6CreateBroadcastAddress ()

	Create IPv6 Broadcast Address (ff02 followed by all one).
Prefix Length.	Ipv6GetPrefixLength () Get prefix length of an interface.
void	Ipv6SetMacLayerStatusEventHandlerFunction (Node* node, Ipv6MacLayerStatusEventHandlerFunctionType StatusEventHandlerPtr, int interfaceIndex) Allows the MAC layer to send status messages (e.g., packet drop, link failure) to a network-layer routing protocol for routing optimization.
void	Ipv6DeleteOutboundPacketsToANode (Node* node, const in6_addr nextHopAddress, const in6_addr destinationAddress, const BOOL returnPacketsToRoutingProtocol) 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.
void	Ipv6IsLoopbackAddress (Node* node, in6_addr address) Check if address is self loopback address.
TRUE if my Ip, FALSE otherwise.	Ipv6IsMyIp (Node* node, in6_addr* dst_addr) Check if address is self loopback address.
Scope value if valid multicast address, 0 otherwise.	Ipv6IsValidGetMulticastScope (Node* node, in6_addr multiAddr) Check if multicast address has valid scope.
BOOL	IsIPv6RoutingEnabledOnInterface (Node* node, int interfaceIndex) To check if IPV6 Routing is enabled on interface?

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 Lenght.
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 infromation 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	IPv6_HEADER_LENGTH 40 Length of IPv6 header
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_h2hdr_struct QualNet typedefs struct ipv6_h2hdr_struct to ipv6_h2hdr. struct ipv6_h2hdr_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_h2hdr_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_h2hdr_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	<div>destination_route_struct</div> <div>QualNet typedefs struct destination_route_struct to destinationRoute. struct destination_route_struct is destination information structure of a node.</div>
Structure	<div>DestinationCache</div> <div>Destination cache entry structure</div>
Structure	<div>Ipv6HashData</div> <div>Ipv6 hash data structure.</div>
Structure	<div>Ipv6HashBlockData</div> <div>Ipv6 hash block-data structure.</div>
Structure	<div>Ipv6HashBlock</div> <div>Ipv6 hash block structure.</div>
Structure	<div>Ipv6HashTable</div> <div>Ipv6 hash table structure</div>
Structure	<div>IPv6Data</div> <div>QualNet typedefs struct ipv6_data_struct to IPv6Data. struct ipv6_data_struct is ipv6 information structure of a node.</div>
Structure	<div>ndpNadvEvent</div> <div>QualNet typedefs struct ndp_event_struct to IPv6Data. struct ndp_event_struct is neighbor advertisement information structure.</div>

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

	<ul style="list-style-type: none">UInt32 - None
ip6_hdrGetClass() Returns the value of ip6_class for ip6_hdr	UInt32 ip6_hdrGetClass() (unsigned int ipv6HdrVcf) Parameters: <ul style="list-style-type: none">ipv6HdrVcf - The variable containing the value of ip6_v,ip6_class Returns: <ul style="list-style-type: none">UInt32 - None
ip6_hdrGetFlow() Returns the value of ip6_flow for ip6_hdr	UInt32 ip6_hdrGetFlow() (unsigned int ipv6HdrVcf) Parameters: <ul style="list-style-type: none">ipv6HdrVcf - The variable containing the value of ip6_v,ip6_class Returns: <ul style="list-style-type: none">UInt32 - None
in6_isanycast Checks whether the address is anycast address of the node.	int in6_isanycast (Node* node, in6_addr addr) Parameters: <ul style="list-style-type: none">node - Pointer to node structure.addr - ipv6 address. Returns: <ul style="list-style-type: none">int - None
Ipv6AddIpv6Header Add an IPv6 packet header to a message. Just calls AddIpHeader.	None Ipv6AddIpv6Header (Node* node, Message* msg, in6_addr srcaddr, in6_addr dst_addr, TosType priority, unsigned char protocol, unsigned hlim) Parameters: <ul style="list-style-type: none">node - Pointer to node.msg - Pointer to message.srcaddr - Source IPv6 address.dst_addr - Destination IPv6 address.priority - Current type of serviceprotocol - IPv6 protocol number.hlim - Hop limit. Returns:

	<ul style="list-style-type: none">None - None
<div><div>Ipv6AddFragmentHeader</div><div>Adds fragment header</div></div>	<div><div>None Ipv6AddFragmentHeader (Node *node node, Message *msg msg, unsigned char nextHeader, unsigned short offset, unsigned int id)</div><div>Parameters:<ul style="list-style-type: none">node - Pointer to nodemsg - Pointer to MessagenextHeader - nextHeaderoffset - offsetid - id</div><div>Returns:<ul style="list-style-type: none">None - None</div></div>
<div><div>Ipv6RemoveIpv6Header</div><div>Removes Ipv6 header</div></div>	<div><div>None Ipv6RemoveIpv6Header (Node *node node, Message *msg msg, Address* sourceAddress, Address* destinationAddress destinationAddress, TosType *priority priority, unsigned char *protocol protocol, unsigned *hLim hLim)</div><div>Parameters:<ul style="list-style-type: none">node - Pointer to nodemsg - Pointer to messagesourceAddress - Poineter Source addressdestinationAddress - Destination addresspriority - Priorityprotocol - protocolhLim - hLim</div><div>Returns:<ul style="list-style-type: none">None - None</div></div>
<div><div>Ipv6PreInit</div><div>IPv6 Pre Initialization.</div></div>	<div><div>None Ipv6PreInit (Node* node)</div><div>Parameters:<ul style="list-style-type: none">node - Pointer to node structure.</div><div>Returns:<ul style="list-style-type: none">None - None</div></div>

IPv6Init IPv6 Initialization.	None IPv6Init (Node* node, const NodeInput* nodeInput) Parameters: <ul style="list-style-type: none">node - Pointer to node structure.nodeInput - Node input. Returns: <ul style="list-style-type: none">None - None
Ipv6IsMyPacket Checks whether the packet is the nodes packet. if the packet is of the node then returns TRUE, otherwise FALSE.	BOOL Ipv6IsMyPacket (Node* node, in6_addr* dst_addr) Parameters: <ul style="list-style-type: none">node - Pointer to node structure.dst_addr - ipv6 packet destination address. Returns: <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.	BOOL Ipv6IsAddressInNetwork (const in6_addr* globalAddr, unsigned int tla, unsigned int vla, unsigned int sla) Parameters: <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. Returns: <ul style="list-style-type: none">BOOL - None
Ipv6GetLinkLayerAddress Returns 32 bit link layer address of the interface.	NodeAddress Ipv6GetLinkLayerAddress (Node* node, int interfaceId, char* ll_addr_str) Parameters: <ul style="list-style-type: none">node - Pointer to node structure.interfaceId - Interface Id.ll_addr_str - Pointer to character link layer Returns: <ul style="list-style-type: none">NodeAddress - None
Ipv6AddNewInterface	None Ipv6AddNewInterface (Node* node, in6_addr* globalAddr, unsigned int tla, unsigned int nla, unsigned int sla,

Adds an ipv6 interface to the node.	<div>int* newinterfaceIndex, const NodeInput* nodeInput)</div> <div>Parameters:</div> <div><ul style="list-style-type: none">node - Pointer to node structure.globalAddr - Global ipv6 address pointer.t1a - Top level id.n1a - Next level id.s1a - Site level id.newinterfaceIndex - Pointer to new interface index.nodeInput - Node Input.</div> <div>Returns:</div> <div><ul style="list-style-type: none">None - None</div>
<div>IPv6IsForwardingEnabled</div> <div>Checks whether the node is forwarding enabled.</div>	<div>BOOL IPv6IsForwardingEnabled (IPv6Data* ipv6)</div> <div>Parameters:</div> <div><ul style="list-style-type: none">ipv6 - Pointer to ipv6 data structure.</div> <div>Returns:</div> <div><ul style="list-style-type: none">BOOL - None</div>
<div>IPv6Layer</div> <div>Handle IPv6 layer events, incoming messages and messages sent to itself (timers, etc.).</div>	<div>None IPv6Layer (Node* node, Message* msg)</div> <div>Parameters:</div> <div><ul style="list-style-type: none">node - Pointer to node.msg - Pointer to message.</div> <div>Returns:</div> <div><ul style="list-style-type: none">None - None</div>
<div>IPv6Finalize</div> <div>Finalize function for the IPv6 model. Finalize functions for all network-layer IPv6 protocols are called here.</div>	<div>None IPv6Finalize (Node* node)</div> <div>Parameters:</div> <div><ul style="list-style-type: none">node - Pointer to node.</div> <div>Returns:</div> <div><ul style="list-style-type: none">None - None</div>
IPv6GetMTU	int IPv6GetMTU (Node* node, int interfaceId)

Returns the maximum transmission unit of the interface.	<p>Parameters:</p> <ul style="list-style-type: none">node - Pointer to node.interfaceId - Interface Id. <p>Returns:</p> <ul style="list-style-type: none">int - None
<p>Ipv6GetInterfaceIndexFromAddress</p> <p>Returns interface index of the specified address.</p>	<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
<p>Ipv6CpuQueueInsert</p> <p>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.</p>	<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 packetnetworkType - Type of network packet is using (IPv6,queueIsFull - Storage for boolean indicator. <p>Returns:</p> <ul style="list-style-type: none">None - None
<p>Ipv6InputQueueInsert</p> <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 IPv6 address. The packet's priority value is also returned.</p>	<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.

	<ul style="list-style-type: none">• <code>msg</code> - Pointer to message with IPv6 packet.• <code>nextHopAddress</code> - Packet's next hop link layer 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 (IPv6,• <code>queueIsFull</code> - Storage for boolean indicator. <p>Returns:</p> <ul style="list-style-type: none">• <code>None</code> - <code>None</code>
<p>Ipv6OutputQueueInsert</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 IPv6 address. The packet's priority value is also returned. Called by <code>QueueUpIpFragmentForMacLayer()</code>.</p>	<p>None Ipv6OutputQueueInsert (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 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> - <code>None</code>
<p>QueueUpIpv6FragmentForMacLayer</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 IPv6 address. The packet's priority value is also returned. Called by <code>QueueUpIpFragmentForMacLayer()</code>.</p>	<p>None QueueUpIpv6FragmentForMacLayer (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 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,

	<ul style="list-style-type: none">queueIsFull - Storage for boolean indicator. <p>Returns:</p> <ul style="list-style-type: none">None - None
<p>Ipv6SendPacketOnInterface</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>	<p>None Ipv6SendPacketOnInterface (Node* node, Message* msg, int incommingInterface, int outgoingInterface, NodeAddress nextHop)</p> <p>Parameters:</p> <ul style="list-style-type: none">node - Pointer to node.msg - Pointer to message with ip packet.incommingInterface - Index of incoming interface.outgoingInterface - Index of outgoing interface.nextHop - Next hop link layer address. <p>Returns:</p> <ul style="list-style-type: none">None - None
<p>Ipv6SendOnBackplane</p> <p>This function is called when the packet delivered through backplane delay. required,before calling the next function.</p>	<p>None Ipv6SendOnBackplane (Node* node, Message* msg, int incommingInterface, int outgoingInterface, NodeAddress hopAddr)</p> <p>Parameters:</p> <ul style="list-style-type: none">node - Pointer to node.msg - Pointer to message with ip packet.incommingInterface - Index of incomming interface.outgoingInterface - Index of outgoing interface.hopAddr - Next hop link layer address. <p>Returns:</p> <ul style="list-style-type: none">None - None
<p>Ipv6SendRawMessage</p> <p>Called by NetworkIpReceivePacketFromTransportLayer() to send to send UDP datagrams using IPv6. This function adds an IPv6 header and calls RoutePacketAndSendToMac().</p>	<p>None Ipv6SendRawMessage (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">node - Pointer to node.msg - Pointer to message with payload datasourceAddress - Source IPv6 address.

	<ul style="list-style-type: none">• <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.• <code>ttl</code> - Time to live. <p>Returns:</p> <ul style="list-style-type: none">• <code>None</code> - None
<p>Ipv6SendToUdp</p> <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 (Node* node, Message* msg, TosType priority, Address sourceAddress, Address destinationAddress, int incomingInterfaceIndex)</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
<p>Ipv6SendToTCP</p> <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 (Node* node, Message* msg, TosType priority, Address sourceAddress, Address destinationAddress, int incomingInterfaceIndex)</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

<div><div>IPv6ReceivePacketFromMacLayer</div><div>IPv6 received IPv6 packet from MAC layer. Determine whether the packet is to be delivered to this node, or needs to be forwarded.</div></div>	<div><div>None IPv6ReceivePacketFromMacLayer (Node* node, Message* msg, NodeAddress previousHopNodeId, int interfaceIndex)</div><div>Parameters:<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.</div><div>Returns:<ul style="list-style-type: none">None - None</div></div>
<div><div>IPv6AddMessageInBuffer</div><div>Adds an ipv6 packet in message in the hold buffer</div></div>	<div><div>BOOL IPv6AddMessageInBuffer (Node* node, Message* msg, in6_addr* nextHopAddr, int inCommingInterface)</div><div>Parameters:<ul style="list-style-type: none">node - Pointer to node structure.msg - Pointer to message with ip packet.nextHopAddr - Source IPv6 address.inCommingInterface - Incoming interface</div><div>Returns:<ul style="list-style-type: none">BOOL - None</div></div>
<div><div>IPv6DeleteMessageInBuffer</div><div>Delets an ipv6 packet in the hold buffer</div></div>	<div><div>BOOL IPv6DeleteMessageInBuffer (Node* node, messageBuffer* mBuf)</div><div>Parameters:<ul style="list-style-type: none">node - Pointer to node structure.mBuf - Pointer to messageBuffer tail.</div><div>Returns:<ul style="list-style-type: none">BOOL - None</div></div>
<div><div>IPv6DropMessageFromBuffer</div><div>Drops an ipv6 packet from the hold buffer</div></div>	<div><div>void IPv6DropMessageFromBuffer (Node* node, messageBuffer* mBuf)</div><div>Parameters:<ul style="list-style-type: none">node - Pointer to node structure.mBuf - Pointer to messageBuffer tail.</div><div>Returns:</div></div>

	<ul style="list-style-type: none">void - None
Ipv6GetAddressTypeFromString Returns network type from string ip address.	NetworkType Ipv6GetAddressTypeFromString (char* interfaceAddr) Parameters: <ul style="list-style-type: none">interfaceAddr - Character Pointer to ip address. Returns: <ul style="list-style-type: none">NetworkType - None
Ipv6GetInterfaceMulticastAddress Get multicast address of this interface	IPv6 multicast address Ipv6GetInterfaceMulticastAddress (Node* node, int interfaceIndex) Parameters: <ul style="list-style-type: none">node - Node pointerinterfaceIndex - interface for which multicast is required Returns: <ul style="list-style-type: none">IPv6 multicast address - None
Ipv6SolicitationMulticastAddress Copies multicast solicitation address.	None Ipv6SolicitationMulticastAddress (in6_addr* dst_addr, in6_addr* target) Parameters: <ul style="list-style-type: none">dst_addr - ipv6 address pointer.target - ipv6 multicast address pointer. Returns: <ul style="list-style-type: none">None - None
Ipv6AllRoutersMulticastAddress Function to assign all routers multicast address.	None Ipv6AllRoutersMulticastAddress (in6_addr* dst dst) Parameters: <ul style="list-style-type: none">dst - IPv6 address pointer, Returns: <ul style="list-style-type: none">None - None
IPv6GetLinkLocalAddress Gets ipv6 link local address of the interface in output parameter addr.	None IPv6GetLinkLocalAddress (node, int interface, in6_addr* addr) Parameters: <ul style="list-style-type: none">node - Pointer to the node structure.interface - interface Index.

	<ul style="list-style-type: none">• <code>addr</code> - ipv6 address pointer. <p>Returns:</p> <ul style="list-style-type: none">• <code>None</code> - <code>None</code>
<p>IPv6GetSiteLocalAddress</p> <p>Gets ipv6 site local address of the interface in output parameter <code>addr</code>.</p>	<p><code>None</code> IPv6GetSiteLocalAddress (<code>node</code>, <code>int</code> interface, <code>in6_addr*</code> <code>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">• <code>None</code> - <code>None</code>
<p>IPv6GetSiteLocalAddress</p> <p>Gets ipv6 global agreeable address of the interface in output parameter <code>addr</code>.</p>	<p><code>None</code> IPv6GetSiteLocalAddress (<code>node</code>, <code>int</code> interface, <code>in6_addr*</code> <code>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">• <code>None</code> - <code>None</code>
<p>Ipv6GetPrefix</p> <p>Gets ipv6 prefix from address.</p>	<p><code>None</code> Ipv6GetPrefix (<code>in6_addr*</code> <code>addr</code>, <code>in6_addr*</code> <code>prefix</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>addr</code> - ipv6 address pointer.• <code>prefix</code> - ipv6 prefix pointer. <p>Returns:</p> <ul style="list-style-type: none">• <code>None</code> - <code>None</code>
<p>Ipv6GetPrefixFromInterfaceIndex</p> <p>Gets ipv6 prefix from address.</p>	<p>Prefix for this interface Ipv6GetPrefixFromInterfaceIndex (<code>Node*</code> <code>node</code>, <code>int</code> <code>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">Prefix for this interface - None
Ipv6OutputQueueIsEmpty Check weather output queue is empty	<p>BOOL Ipv6OutputQueueIsEmpty (Node *node node, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none">node - Pointer to NodeinterfaceIndex - interfaceIndex <p>Returns:</p> <ul style="list-style-type: none">BOOL - None
Ipv6RoutingStaticInit Ipv6 Static routing initialization function.	<p>None Ipv6RoutingStaticInit (Node *node node, const NodeInput nodeInput, NetworkRoutingProtocolType type)</p> <p>Parameters:</p> <ul style="list-style-type: none">node - Pointer to nodenodeInput - *nodeInputtype - type <p>Returns:</p> <ul style="list-style-type: none">None - None
Ipv6RoutingStaticEntry Static routing route entry function	<p>None Ipv6RoutingStaticEntry (Node *node node, char currentLine[] currentLine)</p> <p>Parameters:</p> <ul style="list-style-type: none">node - Pointer to nodecurrentLine - Static entry's current line. <p>Returns:</p> <ul style="list-style-type: none">None - None
Ipv6AddDestination Adds destination in the destination cache.	<p>None Ipv6AddDestination (Node* node node, route* ro ro)</p> <p>Parameters:</p> <ul style="list-style-type: none">node - Pointer to nodero - Pointer to destination route. <p>Returns:</p> <ul style="list-style-type: none">None - None
Ipv6DeleteDestination	None Ipv6DeleteDestination (Node* node node)

Deletes destination from the destination cache.	<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 Checks the packet's validity	<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 nodescheduler - pointer to schedulerpIndex - packet index <p>Returns:</p> <ul style="list-style-type: none">int - None
Ipv6NdpProcessing Ipv6 Destination cache and neighbor cache : processing function	<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 Updates Ipv6 Forwarding Table	<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 nodedestPrefix - IPv6 destination addressnextHopPrefix - IPv6 next hop address for this destinationinterfaceIndex - interfaceIndexmetric - 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>

<p>Empties Ipv6 Forwarding Table for a particular routing protocol entry</p>	<ul style="list-style-type: none">node - Pointer to nodetype - Routing protocol type <p>Returns:</p> <ul style="list-style-type: none">None - None
<p>Ipv6PrintForwardingTable</p> <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
<p>Ipv6InterfaceIndexFromSubnetAddress</p> <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 nodeaddress - Subnet Address <p>Returns:</p> <ul style="list-style-type: none">Interface index associated with specified subnet address. - None
<p>Ipv6GetInterfaceAndNextHopFromForwardingTable</p> <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 nodedestAddr - Destination AddressinterfaceIndex - Pointer to interface indexnextHopAddr - Next Hop Addr for destination. <p>Returns:</p> <ul style="list-style-type: none">void - NULL.
<p>Ipv6GetInterfaceIndexForDestAddress</p> <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

	<ul style="list-style-type: none">destAddr - Destination Address <p>Returns:</p> <ul style="list-style-type: none">interface index associated with destination. - None
<p>Ipv6GetMetricForDestAddress</p> <p>Get the cost metric for a destination from the forwarding table.</p>	<p>interface index associated with destination. Ipv6GetMetricForDestAddress (Node* node node, in6_addr destAddr)</p> <p>Parameters:</p> <ul style="list-style-type: none">node - Pointer to nodedestAddr - Destination Address <p>Returns:</p> <ul style="list-style-type: none">interface index associated with destination. - None
<p>Ipv6IpGetInterfaceIndexForNextHop</p> <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 nodedestAddr - Destination Address <p>Returns:</p> <ul style="list-style-type: none">Interface index associated with destination if found, - None
<p>Ipv6GetRouterFunction</p> <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.
<p>Ipv6SendPacketToMacLayer</p> <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 nodemsg - Pointer to messagedestAddr - Destination AddressnextHopAddr - Next Hop Addr for destination.

	<ul style="list-style-type: none"><code>interfaceIndex</code> - Pointer to interface index Returns: <ul style="list-style-type: none"><code>void</code> - NULL.
Ipv6JoinMulticastGroup Join a multicast group.	void Ipv6JoinMulticastGroup (Node* node, in6_addr mcastAddr, clocktype delay) Parameters: <ul style="list-style-type: none"><code>node</code> - Pointer to node.<code>mcastAddr</code> - multicast group to join.<code>delay</code> - delay. Returns: <ul style="list-style-type: none"><code>void</code> - NULL.
Ipv6AddToMulticastGroupList Add group to multicast group list.	void Ipv6AddToMulticastGroupList (Node* node, in6_addr groupAddress) Parameters: <ul style="list-style-type: none"><code>node</code> - Pointer to node.<code>groupAddress</code> - Group to add to multicast group list. Returns: <ul style="list-style-type: none"><code>void</code> - NULL.
Ipv6LeaveMulticastGroup Leave a multicast group.	void Ipv6LeaveMulticastGroup (Node* node, in6_addr mcastAddr) Parameters: <ul style="list-style-type: none"><code>node</code> - Pointer to node.<code>mcastAddr</code> - multicast group to leave. Returns: <ul style="list-style-type: none"><code>void</code> - NULL.
Ipv6RemoveFromMulticastGroupList Remove group from multicast group list.	void Ipv6RemoveFromMulticastGroupList (Node* node, in6_addr groupAddress) Parameters: <ul style="list-style-type: none"><code>node</code> - Pointer to node.<code>groupAddress</code> - Group to be removed from multicast Returns:

	<ul style="list-style-type: none">void - NULL.
<div><div>Ipv6NotificationOfPacketDrop</div><div>Invoke callback functions when a packet is dropped.</div></div>	<div><div>void Ipv6NotificationOfPacketDrop (Node* node, Message* msg, const NodeAddress nextHopAddress, int interfaceIndex)</div><div>Parameters:<ul style="list-style-type: none">node - Pointer to node.msg - Pointer to message.nextHopAddress - Next Hop AddressinterfaceIndex - Interface Index</div><div>Returns:<ul style="list-style-type: none">void - NULL.</div></div>
<div><div>Ipv6IsPartOfMulticastGroup</div><div>Check if destination is part of the multicast group.</div></div>	<div><div>TRUE if node is part of multicast group, Ipv6IsPartOfMulticastGroup (Node* node, Message* msg, in6_addr groupAddress)</div><div>Parameters:<ul style="list-style-type: none">node - Pointer to node.msg - Pointer to message.groupAddress - Multicast Address</div><div>Returns:<ul style="list-style-type: none">TRUE if node is part of multicast group, - None</div></div>
<div><div>Ipv6IsReservedMulticastAddress</div><div>Check if address is reserved multicast address.</div></div>	<div><div>TRUE if reserved multicast address, FALSE otherwise. Ipv6IsReservedMulticastAddress (Node* node, in6_addr mcastAddr)</div><div>Parameters:<ul style="list-style-type: none">node - Pointer to node.mcastAddr - multicast group to join.</div><div>Returns:<ul style="list-style-type: none">TRUE if reserved multicast address, FALSE otherwise. - None</div></div>
<div><div>Ipv6InMulticastOutgoingInterface</div><div>Determine if interface is in multicast outgoing interface list.</div></div>	<div><div>TRUE if interface is in multicast outgoing interface Ipv6InMulticastOutgoingInterface (Node* node, LinkedList* list, int interfaceIndex)</div><div>Parameters:<ul style="list-style-type: none">node - Pointer to node.list - Pointer to Linked List.</div></div>

	<ul style="list-style-type: none">• <code>interfaceIndex</code> - Interface Index. <p>Returns:</p> <ul style="list-style-type: none">• <code>TRUE</code> if interface is in multicast outgoing interface - <code>None</code>
<p>Ipv6UpdateMulticastForwardingTable</p> <p>update entry with (<code>sourceAddress</code>, <code>multicastGroupAddress</code>) pair. search for the row with (<code>sourceAddress</code>, <code>multicastGroupAddress</code>) and update its interface.</p>	<p>void Ipv6UpdateMulticastForwardingTable (<code>Node*</code> node, <code>in6_addr</code> sourceAddress, <code>in6_addr</code> multicastGroupAddress)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>node</code> - Pointer to node.• <code>sourceAddress</code> - Source Address.• <code>multicastGroupAddress</code> - multicast group. <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - <code>NULL</code>.
<p>Ipv6GetOutgoingInterfaceFromMulticastTable</p> <p>get the interface List that lead to the (<code>source</code>, <code>multicast group</code>) pair.</p>	<p>Interface List if match found, <code>NULL</code> otherwise. Ipv6GetOutgoingInterfaceFromMulticastTable (<code>Node*</code> node, <code>in6_addr</code> sourceAddress, <code>in6_addr</code> groupAddress)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>node</code> - Pointer to node.• <code>sourceAddress</code> - Source Address• <code>groupAddress</code> - multicast group address <p>Returns:</p> <ul style="list-style-type: none">• Interface List if match found, <code>NULL</code> otherwise. - <code>None</code>
<p>Ipv6CreateBroadcastAddress</p> <p>Create IPv6 Broadcast Address (<code>ff02</code> followed by all one).</p>	<p>void Ipv6CreateBroadcastAddress ()</p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - <code>NULL</code>.
<p>Ipv6GetPrefixLength</p> <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. - <code>None</code>
<p>Ipv6SetMacLayerStatusEventHandlerFunction</p>	<p>void Ipv6SetMacLayerStatusEventHandlerFunction (<code>Node*</code> node, <code>Ipv6MacLayerStatusEventHandlerFunctionType</code> StatusEventHandlerPtr, <code>int</code> interfaceIndex)</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>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.
<p>Ipv6DeleteOutboundPacketsToANode</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>	<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.
<p>Ipv6IsLoopbackAddress</p> <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.
<p>Ipv6IsMyIp</p> <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
<p>Ipv6IsValidGetMulticastScope</p>	<p>Scope value if valid multicast address, 0 otherwise. Ipv6IsValidGetMulticastScope (Node* node, in6_addr multiAddr)</p>

Check if multicast address has valid scope.	<p>Parameters:</p> <ul style="list-style-type: none">• <code>node</code> - Pointer to node.• <code>multiAddr</code> - multicast address. <p>Returns:</p> <ul style="list-style-type: none">• Scope value if valid multicast address, 0 otherwise. - None
<p>IsIPv6RoutingEnabledOnInterface</p> <p>To check if IPV6 Routing is enabled on interface?</p>	<p>BOOL IsIPv6RoutingEnabledOnInterface (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



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 © 2001-2013 [SCALABLE Network Technologies, Inc.](#) All rights reserved.

QualNet 7.1 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

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 listlist - 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 listlist - 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
<p>ListInsert</p> <p>Insert an item at the end of the list</p>	<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
<p>FindItem</p> <p>Find an item from the list</p>	<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 identified.• <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
<p>FindItem</p> <p>Find an item from the list</p>	<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 identified.• <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
<p>ListGet</p> <p>Remove an item from the list</p>	<p>void ListGet (Node* node, List* list, ListItem* listItem, BOOL freeItem, BOOL isMsg)</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
<p>ListFree</p> <p>Free the entire list</p>	<p>void ListFree (Node* node, List* list, BOOL isMsg)</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 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> - NULL
<p>IntListInit</p> <p>Initialize the list</p>	<p>void IntListInit (Node* node, IntList** list)</p> <p>Parameters:</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> - NULL
<p>IntListIsEmpty</p> <p>Check if list is empty</p>	<p>BOOL IntListIsEmpty (Node* node, IntList* list)</p> <p>Parameters:</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, <code>TRUE</code>, non-empty, <code>FALSE</code>
<p>IntListGetSize</p> <p>Get the size of the list</p>	<p>int IntListGetSize (Node* node, IntList* list)</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
<p>ListInsert</p> <p>Insert an item at the end of the list</p>	<p>void ListInsert (Node* node, List* list, clocktype timeStamp, void* data)</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> - <code>NULL</code>
<p>IntListGet</p> <p>Remove an item from the list</p>	<p>void IntListGet (Node* node, IntList* list, IntListItem* listItem, BOOL freeItem, BOOL isMsg)</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> - <code>NULL</code>
<p>IntListFree</p> <p>Free the entire list</p>	<p>void IntListFree (Node* node, IntList* list, BOOL isMsg)</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 © 2001-2013 [SCALABLE Network Technologies, Inc.](#) All rights reserved.



QualNet 7.1 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_MacAddress_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_APPLETLK
	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	RandFault
	Structure containing random fault information.

Function / Macro Summary

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

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

	To allow a peek by network layer at packet before processing
void	MAC_SneakPeekAtMacPacket (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	MAC_SneakPeekAtMacPacket (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	MAC_MacLayerAcknowledgement (Node* node, int interfaceIndex, Message* msg, NodeAddress nextHop)
	To send acknowledgement from MAC
void	MAC_MacLayerAcknowledgement (Node* node, int interfaceIndex, Message* msg, MacHWAddress& nextHop)
	To send acknowledgement from MAC
void	MAC_MacLayerAcknowledgement (Node* node, int interfaceIndex, Message* msg, Mac802Address& nextHop)
	To send acknowledgement from MAC
void	MAC_HandOffSuccessfullyReceivedPacket (Node* node, int interfaceIndex, Message* msg, NodeAddress lastHopAddress)
	Pass packet successfully up to the network layer
void	MAC_HandOffSuccessfullyReceivedPacket (Node* node, int interfaceIndex, Message* msg, MacHWAddress* lastHopAddr)
	Pass packet successfully up to the network layer
void	MAC_HandOffSuccessfullyReceivedPacket (Node* node, int interfaceIndex, Message* msg, Mac802ddress* lastHopAddr)
	Pass packet successfully up to the network layer
void	MAC_HandOffSuccessfullyReceivedPacket (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	MAC_HandOffSuccessfullyReceivedPacket (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	MAC_OutputQueueTopPacket (Node* node, int interfaceIndex, Message** msg, NodeAddress* nextHopAddress, int networkType, TosType* priority)

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

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

	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)

	<p>Prints interface Mac Address</p>
void	<p>MAC_PrintMacAddr(Mac802Address* macAddr)</p>
	<p>Prints interface Mac Address</p>
void	<p>MAC_RandFaultInit(Node* node, int interfaceIndex, const char* currentLine)</p>
	<p>Initialization the Random Fault structure from input file</p>
void	<p>MAC_RandFaultFinalize(Node* node, int interfaceIndex)</p>
	<p>IPrint the statistics of Random link fault.</p>
TosType	<p>MAC_GetPacketsPriority(Message* msg)</p>
	<p>Returns the priority of the packet</p>
void	<p>MAC_TranslateMulticatIPv4AddressToMulticastMacAddress(NodeAddress multicastAddress, MacHWAddress* macMulticast)</p>
	<p>Convert the Multicast ip address to multicast MAC address</p>
BOOL	<p>MAC_OutputQueuePeekByIndex(Node* node, int interfaceIndex, int msgIndex, Message** msg, NodeAddress nextHopAddress, TosType priority)</p>
	<p>Look at the packet at the index of the output queue.</p>
BOOL	<p>MAC_OutputQueuePeekByIndex(Node* node, int interfaceIndex, int msgIndex, Message** msg, Mac802Address* nextHopAddress, TosType priority)</p>
	<p>Look at the packet at the index of the output queue.</p>
BOOL	<p>MAC_OutputQueuePeekByIndex(int interfaceIndex, int msgIndex, Message** msg, MacHWAddress* nextHopAddress, TosType priority)</p>
	<p>Look at the packet at the index of the output queue.</p>
BOOL	<p>MAC_OutputQueueDequeuePacketWithIndex(Node* node, int interfaceIndex, int msgIndex, Message** msg, NodeAddress nextHopAddress, int networkType)</p>
	<p>To remove the packet at specified index output queue.</p>
BOOL	<p>MAC_OutputQueueDequeuePacketWithIndex(Node* node, int interfaceIndex, int msgIndex, Message** msg, Mac802Address* nextHopMacAddress, int networkType)</p>

	<p>To remove the packet at specified index output queue.</p>
BOOL	<p>MAC_OutputQueueDequeuePacketWithIndex(Node* node, int interfaceIndex, int msgIndex, Message** msg, MacHWAddress nextHopMacAddress, int networkType)</p> <p>To remove the packet at specified index output queue.</p>
BOOL	<p>MAC_IPv4addressIsMulticastAddress(NodeAddress ipv4)</p> <p>Check the given address is Multicast address or not.</p>
BOOL	<p>MAC_IsBroadcastMacAddress(MacAddress* macAddr)</p> <p>Checks Broadcast MAC address.</p>
void	<p>IPv4AddressToDefaultMac802Address(Node* node, int index, NodeAddress ipv4Address, Mac802Address* macAddr)</p> <p>Retrieve the Mac802Address from IP address.</p>
Bool	<p>ConvertVariableHWAddressTo802Address(Node* node, MacHWAddress* macHWAddr, Mac802Address* mac802Addr)</p> <p>Convert Variable Hardware address to Mac 802 address</p>
void	<p>MAC_CopyMacHWAddress(MacHWAddress* destAddr, MacHWAddress* srcAddr)</p> <p>Copies Hardware address address</p>
NodeAddress	<p>DefaultMac802AddressToIpv4Address(Node* node, Mac802Address* macAddr)</p> <p>Retrieve IP address from.Mac802Address</p>
BOOL	<p>IPv4AddressToHWAddress(Node* node, int interfaceIndex, Message* msg, NodeAddress ipv4Address)</p> <p>Converts IP address.To MacHWAddress</p>
NodeAddress	<p>MacHWAddressToIpv4Address(Node * node, int interfaceIndex, MacHWAddress* macAddr)</p> <p>This functions converts variable length Mac address to IPv4 address It checks the type of hardware address and based on that conversion is done.</p>
char*	<p>decToHex(int dec)</p> <p>Convert one byte decimal number to hex number.</p>
void	<p>MAC_FourByteMacAddressToVariableHWAddress(Node * node, int interfaceIndex, MacHWAddress * macAddr, NodeAddress nodeAddr)</p>

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

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
<div>MacReportInterfaceStatus</div> <div>Callback funtion to report interface status</div>	<div>void MacReportInterfaceStatus (Node* node, int interfaceIndex, MacInterfaceState state)</div> <div>Parameters:</div> <div><ul style="list-style-type: none">node - Pointer to a network nodeinterfaceIndex - index of interfacestate - Wheather it enable or disable</div> <div>Returns:</div> <div><ul style="list-style-type: none">void - None</div>
<div>MAC_SetInterfaceStatusHandlerFunction</div> <div>Set the MAC interface handler function to be called when interface faults occurs</div>	<div>void MAC_SetInterfaceStatusHandlerFunction (Node* node, int interfaceIndex, MacReportInterfaceStatus statusHandler)</div> <div>Parameters:</div> <div><ul style="list-style-type: none">node - Pointer to a network nodeinterfaceIndex - index of interfacestatusHandler - Pointer to status Handler</div> <div>Returns:</div> <div><ul style="list-style-type: none">void - None</div>
<div>MAC_GetInterfaceStatusHandlerFunction</div> <div>To get the MACInterface status handling function for the system</div>	<div>MacReportInterfaceStatus MAC_GetInterfaceStatusHandlerFunction (Node* node, int interfaceIndex)</div> <div>Parameters:</div> <div><ul style="list-style-type: none">node - Pointer to a network nodeinterfaceIndex - index of interface</div>

	<p>Returns:</p> <ul style="list-style-type: none">• <code>MacReportInterfaceStatus</code> - Pointer to status handler
<p>MacHasFrameToSendFn</p> <p>Callback funtion 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
<p>MacReceiveFrameFn</p> <p>Callback funtion 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
<p>MAC_NetworkLayerHasPacketToSend</p> <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
<p>MAC_SwitchHasPacketToSend</p> <p>To inform MAC that the Switch has packets to 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

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

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

To notify MAC of packet drop	<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> - <code>interfaceIndex</code>• <code>msg</code> - Pointer to Message <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
MAC_NotificationOfPacketDrop To notify MAC of packet drop	<p><code>void MAC_NotificationOfPacketDrop (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> - <code>interfaceIndex</code>• <code>msg</code> - Pointer to Message <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
MAC_NotificationOfPacketDrop To notify MAC of packet drop	<p><code>void MAC_NotificationOfPacketDrop (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> - <code>interfaceIndex</code>• <code>msg</code> - Pointer to Message <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
MAC_OutputQueueTopPacketForAPriority To notify MAC of priority packet arrival	<p><code>BOOL MAC_OutputQueueTopPacketForAPriority (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> - <code>interfaceIndex</code>• <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.
<p>MAC_OutputQueueTopPacketForAPriority</p> <p>To notify MAC of priority packet arrival</p>	<p>BOOL MAC_OutputQueueTopPacketForAPriority (Node* node, int interfaceIndex, TosType priority, Message** msg, Mac802Address* nextHopAddress)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>node</code> - Pointer to a network node• <code>interfaceIndex</code> - <code>interfaceIndex</code>• <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.
<p>MAC_OutputQueueTopPacketForAPriority</p> <p>To notify MAC of priority packet arrival</p>	<p>BOOL MAC_OutputQueueTopPacketForAPriority (Node* node, int interfaceIndex, TosType priority, Message** msg, MacHWAddress* nextHopAddress)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>node</code> - Pointer to a network node• <code>interfaceIndex</code> - <code>interfaceIndex</code>• <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.
<p>MAC_OutputQueueDequeuePacketForAPriority</p>	<p>BOOL MAC_OutputQueueDequeuePacketForAPriority (Node* node, int interfaceIndex, TosType priority, Message** msg, NodeAddress* nextHopAddress, int* networkType)</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> - <code>interfaceIndex</code>• <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>BOOL MAC_OutputQueueDequeuePacketForAPriority (Node* node, int interfaceIndex, TosType priority, Message** msg, MacHWAddress* nextHopAddress, int* networkType)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>node</code> - Pointer to a network node• <code>interfaceIndex</code> - <code>interfaceIndex</code>• <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>BOOL MAC_OutputQueueDequeuePacketForAPriority (Node* node, int interfaceIndex, TosType priority, Message** msg, Mac802Address* nextHopAddress, int* networkType)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>node</code> - Pointer to a network node• <code>interfaceIndex</code> - <code>interfaceIndex</code>• <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 allow a peek by network layer at packet before processing It is overloading function used for ARP packet</p>	<p>BOOL MAC_OutputQueueDequeuePacketForAPriority (Node* node, int interfaceIndex, TosType* priority, Message** msg, MacHWAddress* destMacAddr, int* networkType, int* packType)</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 MAC_OutputQueueDequeuePacketForAPriority()
<p>MAC_OutputQueueDequeuePacketForAPriority</p> <p>To allow a peek by network layer at packet before processing It is overloading function used for ARP packet</p>	<p>BOOL MAC_OutputQueueDequeuePacketForAPriority (Node* node, int interfaceIndex, TosType* priority, Message** msg, Mac802Address* destMacAddr, int* networkType, int* packType)</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 MAC_OutputQueueDequeuePacketForAPriority()
<p>MAC_SneakPeekAtMacPacket</p> <p>To allow a peek by network layer at packet before processing</p>	<p>void MAC_SneakPeekAtMacPacket (Node* node, int interfaceIndex, const Message* msg, NodeAddress prevHop, NodeAddress destAddr, int messageType)</p> <p>Parameters:</p>

	<ul style="list-style-type: none">• <code>node</code> - Pointer to a network node• <code>interfaceIndex</code> - <code>interfaceIndex</code>• <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
<p>MAC_SneakPeekAtMacPacket</p> <p>To allow a peek by network layer at packet before processing</p>	<p><code>void</code> MAC_SneakPeekAtMacPacket (<code>Node*</code> node, <code>int</code> interfaceIndex, <code>const</code> <code>Message*</code> msg, <code>MacHWAddress</code> prevHop, <code>MacHWAddress</code> destAddr, <code>int</code> arpMessageType)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>node</code> - Pointer to a network node• <code>interfaceIndex</code> - <code>interfaceIndex</code>• <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
<p>MAC_SneakPeekAtMacPacket</p> <p>To allow a peek by network layer at packet before processing</p>	<p><code>void</code> MAC_SneakPeekAtMacPacket (<code>Node*</code> node, <code>int</code> interfaceIndex, <code>const</code> <code>Message*</code> msg, <code>Mac802Address</code> prevHop, <code>Mac802Address</code> destAddr, <code>int</code> messageType)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>node</code> - Pointer to a network node• <code>interfaceIndex</code> - <code>interfaceIndex</code>• <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
<div><div>MAC_MacLayerAcknowledgement</div><div>To send acknowledgement from MAC</div></div>	<div><div>void MAC_MacLayerAcknowledgement (Node* node, int interfaceIndex, Message* msg, NodeAddress nextHop)</div><div>Parameters:<ul style="list-style-type: none">node - Pointer to a network nodeinterfaceIndex - interfaceIndexmsg - Pointer to MessagenextHop - Pointer to Node address</div><div>Returns:<ul style="list-style-type: none">void - None</div></div>
<div><div>MAC_MacLayerAcknowledgement</div><div>To send acknowledgement from MAC</div></div>	<div><div>void MAC_MacLayerAcknowledgement (Node* node, int interfaceIndex, Message* msg, MacHWAddress& nextHop)</div><div>Parameters:<ul style="list-style-type: none">node - Pointer to a network nodeinterfaceIndex - interfaceIndexmsg - Pointer to MessagenextHop - Pointer to Node address</div><div>Returns:<ul style="list-style-type: none">void - None</div></div>
<div><div>MAC_MacLayerAcknowledgement</div><div>To send acknowledgement from MAC</div></div>	<div><div>void MAC_MacLayerAcknowledgement (Node* node, int interfaceIndex, Message* msg, Mac802Address& nextHop)</div><div>Parameters:<ul style="list-style-type: none">node - Pointer to a network nodeinterfaceIndex - interfaceIndexmsg - Pointer to MessagenextHop - Pointer to nexthop mac address</div><div>Returns:<ul style="list-style-type: none">void - None</div></div>
<div><div>MAC_HandOffSuccessfullyReceivedPacket</div></div>	<div><div>void MAC_HandOffSuccessfullyReceivedPacket (Node* node, int interfaceIndex, Message* msg,</div></div>

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

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

<p>To check packet at the top of output queue</p>	<p>MacHWAddress* nextHopAddress, int networkType, TosType* priority)</p> <p>Parameters:</p> <ul style="list-style-type: none">node - Pointer to a network nodeinterfaceIndex - interfaceIndexmsg - Pointer to MessagenextHopAddress - Next hop addressnetworkType - network typepriority - 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>To check packet at the top of output queue</p>	<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 nodeinterfaceIndex - interfaceIndexmsg - Pointer to MessagenextHopAddress - Next hop addressnetworkType - network typepriority - Message Priority <p>Returns:</p> <ul style="list-style-type: none">BOOL - TRUE if there is a packet, FALSE otherwise.
<p>MAC_OutputQueueDequeuePacket</p> <p>To remove packet from front of output queue</p>	<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 nodeinterfaceIndex - interfaceIndexmsg - Pointer to MessagenextHopAddress - Pointer to Node addressnetworkType - 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.
<p>MAC_OutputQueueDequeuePacket</p> <p>To remove packet from front of output queue</p>	<p>BOOL MAC_OutputQueueDequeuePacket (Node* node, int interfaceIndex, Message** msg, MacHWAddress* nextHopAddress, int networkType, TosType * priority)</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.
<p>MAC_OutputQueueDequeuePacket</p> <p>To remove packet from front of output queue, Its a overloaded function</p>	<p>BOOL MAC_OutputQueueDequeuePacket (Node* node, int interfaceIndex, Message** msg, Mac802Address* nextHopAddress, int networkType, TosType * priority)</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.
<p>MAC_OutputQueueDequeuePacket</p> <p>To remove packet(s) from front of output queue; process packets with options for example, pakcing multiple packets with same</p>	<p>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)</p>

<p>next hop address together</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• priority - Pointer to queuing priority type• dequeueOption - option• dequeueCriteria - criteria• numFreeByte - number of bytes can be packed in 1 transmission• numPacketPacked - number of packets packed• tracePrt - Trace Protocol Type• eachWithMacHeader - Each msg has its own MAC header?• maxHeaderSize - max mac header size• returnPackedMsg - return Packed msg or a list of msgs <p>Returns:</p> <ul style="list-style-type: none">• BOOL - 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, pakcing multiple packets with same next hop address together</p>	<p>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)</p> <p>Parameters:</p> <ul style="list-style-type: none">• node - Pointer to a network node• interfaceIndex - interfaceIndex• priority - Pointer to queuing priority type• msg - Pointer to Message• nextHopAddress - Pointer to Node address• networkType - network type• dequeueOption - 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.
<p>MAC_IsMyUnicastFrame</p> <p>Check if a packet (or frame) belongs to this node Should be used only for four byte mac address</p>	<p>BOOL MAC_IsMyUnicastFrame (Node* node, NodeAddress destAddr)</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
<p>MAC_IsWiredNetwork</p> <p>To check if an interface is a wired interface</p>	<p>BOOL MAC_IsWiredNetwork (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
<p>MAC_IsPointToPointNetwork</p> <p>Checks if an interface belongs to Point to PointNetwork</p>	<p>BOOL MAC_IsPointToPointNetwork (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_IsPointToMultiPointNetwork Checks if an interface belongs to Point to Multi-Point network.	BOOL MAC_IsPointToMultiPointNetwork (Node* node, int interfaceIndex) Parameters: <ul style="list-style-type: none">node - Pointer to a network nodeinterfaceIndex - interfaceIndex Returns: <ul style="list-style-type: none">BOOL - boolean
MAC_IsWiredBroadcastNetwork Determines if an interface is a wired broadcast interface	BOOL MAC_IsWiredBroadcastNetwork (Node* node, int interfaceIndex) Parameters: <ul style="list-style-type: none">node - Pointer to a network nodeinterfaceIndex - interfaceIndex Returns: <ul style="list-style-type: none">BOOL - boolean
MAC_IsWirelessNetwork Determine if a node's interface is a wireless interface	BOOL MAC_IsWirelessNetwork (Node* node, int interfaceIndex) Parameters: <ul style="list-style-type: none">node - Pointer to a network nodeinterfaceIndex - interfaceIndex Returns: <ul style="list-style-type: none">BOOL - boolean
MAC_IsWirelessAdHocNetwork Determine if a node's interface is a possible wireless ad hoc interface	BOOL MAC_IsWirelessAdHocNetwork (Node* node, int interfaceIndex) Parameters: <ul style="list-style-type: none">node - Pointer to a network nodeinterfaceIndex - interfaceIndex Returns: <ul style="list-style-type: none">BOOL - boolean
MAC_IsOneHopBroadcastNetwork Determines if an interface is a single Hop Broadcast interface	BOOL MAC_IsOneHopBroadcastNetwork (Node* node, int interfaceIndex) Parameters: <ul style="list-style-type: none">node - Pointer to a network nodeinterfaceIndex - interfaceIndex

	<p>Returns:</p> <ul style="list-style-type: none">• <code>BOOL</code> - boolean
<p>MAC_IsASwitch</p> <p>To check if a node is a switch</p>	<p>BOOL MAC_IsASwitch (Node* node)</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
<p>MAC_SetVirtualMacAddress</p> <p>To set MAC address</p>	<p>void MAC_SetVirtualMacAddress (Node* node, int interfaceIndex, NodeAddress virtualMacAddress)</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
<p>MacSetDefaultHWAddress</p> <p>Set Default interface Hardware Address of node</p>	<p>void MacSetDefaultHWAddress (NodeId nodeId, MacHWAddress* macAddr, int interfaceIndex)</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
<p>MAC_IsMyMacAddress</p> <p>To check if received mac address belongs to itself</p>	<p>NodeAddress MAC_IsMyMacAddress (Node* node, int interfaceIndex, NodeAddress destAddr)</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 Checks for own MAC address.	BOOL MAC_IsMyHWAddress (Node* node, int interfaceIndex, MacAddress* macAddr) Parameters: <ul style="list-style-type: none">node - Node pointerinterfaceIndex - Interface indexmacAddr - Mac Address Returns: <ul style="list-style-type: none">BOOL - None
MacValidateAndSetHWAddress Validate MAC Address String after fetching from user	void MacValidateAndSetHWAddress (char* macAddrStr, MacHWAddress* macAddr) Parameters: <ul style="list-style-type: none">macAddrStr - Pointer to address stringmacAddr - Pointer to hardware address structure Returns: <ul style="list-style-type: none">void - NULL
DefaultMacHWAddressToIpv4Address Retrieve the IP Address from Default HW Address . Default HW address is equal to 6 bytes	NodeAddress DefaultMacHWAddressToIpv4Address (Node* node, MacHWAddress* macAddr) Parameters: <ul style="list-style-type: none">node - Pointer to Node structuremacAddr - Pointer to hardware address structure Returns: <ul style="list-style-type: none">NodeAddress - Ip address
MacGetHardwareLength Retrieve the Hardware Length.	void MacGetHardwareLength (Node* node, int interface, unsigned short hwLength) Parameters: <ul style="list-style-type: none">node - Pointer to Node structureinterface - interface whose hardware length requiredhwLength - Pointer to hardware string Returns: <ul style="list-style-type: none">void - NULL
MacGetHardwareType	void MacGetHardwareType (Node* node, int interface, unsigned short* type)

Retrieve the Hardware Type.	<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
MacGetHardwareAddressString Retrieve the Hardware Address String.	<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
MacAddNewInterface To add a new Interface at MAC	<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
MacAddVlanInfoForThisInterface Init and read VLAN configuration from user input for node and interface passed as arguments	<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 Returns: <ul style="list-style-type: none">• <code>void</code> - None
MacReleaseVlanInfoForThisInterface To flush VLAN info for an interface	NodeAddress MacReleaseVlanInfoForThisInterface (Node* node, int interfaceIndex) Parameters: <ul style="list-style-type: none">• <code>node</code> - Pointer to a network node• <code>interfaceIndex</code> - interface index Returns: <ul style="list-style-type: none">• <code>NodeAddress</code> - Node Address
MAC_IsBroadcastHWAddress Checks Broadcast MAC address	BOOL MAC_IsBroadcastHWAddress (MacHWAddress* macAddr) Parameters: <ul style="list-style-type: none">• <code>macAddr</code> - structure to hardware address Returns: <ul style="list-style-type: none">• <code>BOOL</code> - TRUE or FALSE
MAC_IsIdenticalHWAddress Compares two MAC addresses	BOOL MAC_IsIdenticalHWAddress (MacHWAddress* macAddr1, MacHWAddress* macAddr2) Parameters: <ul style="list-style-type: none">• <code>macAddr1</code> - Pointer to hardware address structure• <code>macAddr2</code> - Pointer to hardware address structure Returns: <ul style="list-style-type: none">• <code>BOOL</code> - TRUE or FALSE
MAC_PrintHWAddr Prints interface Mac Address	void MAC_PrintHWAddr (MacHWAddress* macAddr) Parameters: <ul style="list-style-type: none">• <code>macAddr</code> - Mac address Returns: <ul style="list-style-type: none">• <code>void</code> - None
MAC_PrintMacAddr	void MAC_PrintMacAddr (Mac802Address* macAddr) Parameters:

<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>void MAC_RandFaultInit (Node* node, int interfaceIndex, const char* currentLine)</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>void MAC_RandFaultFinalize (Node* node, int interfaceIndex)</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>TosType MAC_GetPacketsPriority (Message* msg)</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_TranslateMulticatIPv4AddressToMulticastMacAddress</p> <p>Convert the Multicast ip address to multicast MAC address</p>	<p>void MAC_TranslateMulticatIPv4AddressToMulticastMacAddress (NodeAddress multicastAddress, MacHWAddress* macMulticast)</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

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

	<ul style="list-style-type: none">• <code>priority</code> - priority <p>Returns:</p> <ul style="list-style-type: none">• <code>BOOL</code> - TRUE if the messeage found, FALSE otherwise
<p>MAC_OutputQueueDequeuePacketWithIndex</p> <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">• <code>node</code> - Node pointer• <code>interfaceIndex</code> - Interface index• <code>msgIndex</code> - Message index• <code>msg</code> - Double pointer to message• <code>nextHopAddress</code> - Next hop IP address• <code>networkType</code> - Type of network <p>Returns:</p> <ul style="list-style-type: none">• <code>BOOL</code> - TRUE if the messeage dequeued properly, FALSE otherwise
<p>MAC_OutputQueueDequeuePacketWithIndex</p> <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">• <code>node</code> - Node pointer• <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 messeage dequeued properly, FALSE otherwise
<p>MAC_OutputQueueDequeuePacketWithIndex</p> <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">• <code>node</code> - 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 messeage dequeued properly, FALSE otherwise
<p>MAC_IPv4addressIsMulticastAddress</p> <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
<p>MAC_IsBroadcastMacAddress</p> <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
<p>IPv4AddressToDefaultMac802Address</p> <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
<p>ConvertVariableHWAddressTo802Address</p>	<p>Bool ConvertVariableHWAddressTo802Address (Node* node, MacHWAddress* macHWAddr, Mac802Address* mac802Addr)</p>

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

<p>This functions 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">• <code>node</code> - Pointer to node which indicates the host• <code>interfaceIndex</code> - Interface index of a node• <code>macAddr</code> - Pointer to MacHWAddress Structure. <p>Returns:</p> <ul style="list-style-type: none">• <code>NodeAddress</code> - 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">• <code>dec</code> - decimal number <p>Returns:</p> <ul style="list-style-type: none">• <code>char*</code> - 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">• <code>node</code> - Pointer to node which indicates the host• <code>interfaceIndex</code> - Interface index of a node• <code>macAddr</code> - Pointer to source MacHWAddress Structure• <code>nodeAddr</code> - Ip address <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - 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">• <code>node</code> - Pointer to Node structure• <code>macAddr</code> - Pointer to hardware address structure <p>Returns:</p> <ul style="list-style-type: none">• <code>NodeAddress</code> - Ipv4 Address
<p>GetBroadCastAddress</p>	<p>MacHWAddress GetBroadCastAddress (Node* node, int interfaceIndex)</p> <p>Parameters:</p>

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

	<ul style="list-style-type: none">• <code>int</code> - interfaceIndex of node
MAC_Reset Reset the Mac protocols use by the node	<p>void MAC_Reset (Node* node, int InterfaceIndex)</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 Add which protocols in the Mac layer to be reset to a fuction list pointer.	<p>void MAC_AddResetFunctionList (Node* node, int InterfaceIndex, void* param)</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 © 2001-2013 [SCALABLE Network Technologies, Inc.](#) All rights reserved.



QualNet 7.1 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

	<p>This is a special addresses used in the MAC and network layers. It defines any IP.</p>
CONSTANT	<p>ANY_INTERFACE</p> <p>This is a special addresses used in the MAC and network layers. It defines any Interface.</p>
CONSTANT	<p>CPU_INTERFACE</p> <p>This is a special addresses used in the MAC and network layers. It defines CPU Interface.</p>
CONSTANT	<p>INVALID_ADDRESS</p> <p>It defines Invalid Address. Used only by mac/mac_802_11.c.</p>
CONSTANT	<p>MAX_STRING_LENGTH</p> <p>Generic maximum length of a string. The maximum length of any line in the input file is 3x this value.</p>
CONSTANT	<p>BIG_STRING_LENGTH</p> <p>maximum length of a string.</p>
CONSTANT	<p>MAX_CLOCK_STRING_LENGTH</p> <p>Generic maximum length of a clock string.</p>
CONSTANT	<p>MAX_NW_PKT_SIZE</p> <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	<p>MIN_NW_PKT_SIZE</p> <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	<p>MIN_IPv6_PKT_SIZE</p> <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>

	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 thelast 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 thelast 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 thelast 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 thelast 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 thelast 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 thelast 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 0xfffffffffff This is a special addresses used in the MAC and network layers. It defines any destination of six byte.
Constant	INVALID_802ADDRESS 0xffffffffffe 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

	<p>maximum length of a string.</p>
Constant	<p>MAX_CLOCK_STRING_LENGTH 24</p> <p>Generic maximum length of a clock string.</p>
Constant	<p>MAX_NW_PKT_SIZE 2048</p> <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	<p>MIN_NW_PKT_SIZE 40</p> <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	<p>MIN_IPv6_PKT_SIZE 60</p> <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	<p>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.</p>
Structure	<p>in6_addr</p> <p>Describes the IPv6 address</p>
Structure	<p>AtmAddress</p> <p>Describes the ATM address</p>
Structure	<p>Address</p>

	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	NodeAddress GetIPv4Address (Address addr) Parameters: <ul style="list-style-type: none">addr - generic address. Returns: <ul style="list-style-type: none">NodeAddress - IPv4 address
GetIPv6Address Get IPv6 address from generic address	in6_addr GetIPv6Address (Address addr) Parameters: <ul style="list-style-type: none">addr - generic address. Returns: <ul style="list-style-type: none">in6_addr - IPv6 address
SetIPv4AddressInfo Set IPv4 address and network type to generic address	void SetIPv4AddressInfo (Address address, NodeAddress addr) Parameters: <ul style="list-style-type: none">address - generic address.addr - IPv4 interface address.

	<div>Returns:</div> <div><ul style="list-style-type: none"><code>void</code> - NULL</div>
<div>SetIPv6AddressInfo</div> <div>Set IPv6 address and network type to generic address</div>	<div>void SetIPv6AddressInfo (Address address, in6_addr addr)</div> <div>Parameters:</div> <div><ul style="list-style-type: none"><code>address</code> - generic address.<code>addr</code> - IPv6 interface address.</div> <div>Returns:</div> <div><ul style="list-style-type: none"><code>void</code> - NULL</div>
<div>RoundToInt</div> <div>Round a float point number to an integer. This function tries to get consistent value on different platforms</div>	<div>int RoundToInt (double x)</div> <div>Parameters:</div> <div><ul style="list-style-type: none"><code>x</code> - The float point number to be rounded</div> <div>Returns:</div> <div><ul style="list-style-type: none"><code>int</code> - Returns the rounded integer</div>
<div>MEM_malloc</div> <div>Allocates memory block of a given size.</div>	<div>void* MEM_malloc (size_t size, char* filename, int lineno)</div> <div>Parameters:</div> <div><ul style="list-style-type: none"><code>size</code> - Size of the memory block to be allocated.<code>filename</code> - Name of file allocating the memory<code>lineno</code> - Line in the file where the API is called</div> <div>Returns:</div> <div><ul style="list-style-type: none"><code>void*</code> - Returns the pointer of allocated memory otherwise NULL if allocation fails.</div>
<div>MEM_free</div> <div>Deallocates the memory in turn it calls free().</div>	<div>void MEM_free (void* ptr)</div> <div>Parameters:</div> <div><ul style="list-style-type: none"><code>ptr</code> - Pointer of memory to be freed.</div> <div>Returns:</div> <div><ul style="list-style-type: none"><code>void</code> - None</div>
<div>maskChar</div>	<div>UInt8 maskChar (UInt8 sposition, UInt8 eposition)</div> <div>Parameters:</div>

<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</code> maskShort (<code>UInt16</code> sposition, <code>UInt16</code> eposition)</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</code> maskInt (<code>int</code> sposition, <code>int</code> eposition)</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</code> LshiftChar (<code>UInt8</code> x, <code>UInt8</code> eposition)</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</code> LshiftShort (<code>UInt16</code> x, <code>UInt16</code> eposition)</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

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



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 © 2001-2013 [SCALABLE Network Technologies, Inc.](#) All rights reserved.



QualNet 7.1 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 nodeIds 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	IS_IPV4ADDR6(a) Checks whether it is ipv4 address.
MACRO	IS_LOCALADDR6(a) Checks whether it is local address.
MACRO	IS_LINKLADDR6(a) Checks whether it is link local address.
MACRO	IS_SITEADDR6(a) Checks whether it is site local address.
MACRO	SAME_ADDR4(a, b) Checks whether IPv4 addresses match.
MACRO	IS_ANYADDR4(a) Checks whether IPv4 address is ANY_DEST.
BOOL	Address_IsSameAddress (Address* addr1 addr1, Address* addr2 addr2) Check whether both addresses(i.e. addr1 and addr2) are same.
BOOL	Address_IsAnyAddress (Address* addr addr) Check whether addr is any address of the same type
BOOL	Address_IsMulticastAddress (Address* addr addr) Check whether addr is a multicast address
BOOL	Address_IsSubnetBroadcastAddress (Node* node node, Address* addr addr) Check whether addr is a subnet broadcast address
void	Address_SetToAnyAddress (Address* addr addr, Address* refAddr refAddr) Set addr to any address of the same type as refAddr.

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

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

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	MAPPING_GetNetworkType (const char* addrString) Identify network type from addrString.
void	MAPPING_GetIpv6InterfaceInfoForInterface (Node *node node, NodeId nodeId nodeId, int interfaceIndex, in6_addr* globalAddr, in6_addr* subnetAddr, unsigned int* subnetPrefixLen) Get IPV6 interface information for a interface.
BOOL	MAPPING_GetIpv6GlobalAddress (Node *node node, NodeId nodeId nodeId, unsigned int tla, unsigned int nla, unsigned int sla, in6_addr * addr6) Get IPV6 global address.
BOOL	MAPPING_GetIpv6GlobalAddressForInterface (Node * node, NodeId nodeId, int interfaceIndex, in6_addr * addr6, BOOL isDeprecated) Get IPV6 global address for a node's nth interface.
void	MAPPING_CreateIpv6GlobalUnicastAddr (unsigned int tla, unsigned int nla, unsigned int sla, int addressCounter, in6_addr* globalAddr) Create IPv6 Global Unicast Address from tla nla sla
void	MAPPING_CreateIpv6GlobalUnicastAddr (AddressMapType * map, in6_addr IPv6subnetAddress, unsigned int IPv6subnetPrefixLen, int addressCounter, in6_addr* globalAddr) Create IPv6 Global Unicast Address.
void	MAPPING_CreateIpv6LinkLocalAddr (Node* node, Int32 interfaceId, in6_addr* globalAddr, in6_addr* linkLocalAddr, unsigned int subnetPrefixLen) Create IPv6 link local Address.
void	MAPPING_CreateIpv6SiteLocalAddr (in6_addr* globalAddr, in6_addr* siteLocalAddr, unsigned short siteCounter, unsigned int subnetPrefixLen) Create IPv6 site local Address.
void	MAPPING_CreateIpv6MulticastAddr (in6_addr* globalAddr, in6_addr* multicastAddr) Create ipv6 multicast address.
void	MAPPING_CreateIpv6SubnetAddr (unsigned int tla, unsigned int nla, unsigned int sla, unsigned int* IPv6subnetPrefixLen, in6_addr* IPv6subnetAddress) create subnet addr for IPV6 address.

NodeId	MAPPING_GetNodeIdFromGlobalAddr (Node * node, in6_addr* globalAddr) 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.
NetworkType	MAPPING_GetNetworkTypeFromInterface (Node* node, Int32 interfaceIndex)
	This function determines the network type of a particular interface of a node

Constant / Data Structure Detail

Constant	INVALID_MAPPING 0xffffffff

	Indicates Invalid Mapping
Constant	MAX_INTERFACE_ADDRESSES 4 max no of addressees assigned to an interface
Constant	NODE_HASH_SIZE 32 Defines node hash size. Hashes the nodeIds 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 nodeId 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	BOOL Address_IsSameAddress (Address* addr1 addr1, Address* addr2 addr2) Parameters: <ul style="list-style-type: none">addr1 - Pointer to 1st addressaddr2 - Pointer to 2nd address Check whether both addresses(i.e. addr1 and addr2) are same.

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

	<p>Returns:</p> <ul style="list-style-type: none"><code>void</code> - NULL
<p>Ipv6CompareAddr6</p> <p>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>	<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
<p>Ipv6IsAddressInNetwork</p> <p>Checks whether the address is in the same network. : if in the same network then returns TRUE, otherwise FALSE.</p>	<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
<p>Ipv6IsAddressInNetwork</p> <p>Checks whether the address is in the same network. : if in the same network then returns TRUE, otherwise FALSE.</p>	<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
<p>Ipv6CheckNetworkParams</p> <p>Checks network parameters (tla, nla, sla)</p>	<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.

	<ul style="list-style-type: none">sla - Site level aggregaton. <p>Returns:</p> <ul style="list-style-type: none">BOOL - None
<p>MAPPING_HashNodeId</p> <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 pointernodeId - Node id.nodePtr - Node poniter <p>Returns:</p> <ul style="list-style-type: none">void - None
<p>MAPPING_GetNodePtrFromHash</p> <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 pointernodeId - Node id. <p>Returns:</p> <ul style="list-style-type: none">Node* - Node pointer for nodeId.
<p>MAPPING_MallocAddressMap</p> <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.
<p>MAPPING_InitAddressMap</p> <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
<p>MAPPING_BuildAddressMap</p>	<p>void MAPPING_BuildAddressMap (const NodeInput* nodeInput, NodeAddress** nodeIdArrayPtr, AddressMapType* map)</p> <p>Parameters:</p>

Builds the address map	<ul style="list-style-type: none">• <code>nodeInput</code> - A pointer to const <code>NodeInput</code>.• <code>nodeIdArrayPtr</code> - A pointer to pointer of <code>NodeAddress</code>• <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 Gives Interface address for a Subnet.	<p><code>NodeAddress</code> MAPPING_GetInterfaceAddressForSubnet (<code>Node*</code> node, <code>NodeAddress</code> nodeId, <code>NodeAddress</code> subnetAddress, <code>int</code> 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>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 Gives Interface address for a Subnet.	<p><code>NodeAddress</code> MAPPING_GetInterfaceAddressForSubnet (<code>const AddressMapType*</code> map, <code>NodeAddress</code> nodeId, <code>NodeAddress</code> subnetAddress, <code>int</code> numHostBits)</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 Gives the Subnet address for an interface.	<p><code>NodeAddress</code> MAPPING_GetSubnetAddressForInterface (<code>Node*</code> node, <code>NodeAddress</code> nodeId, <code>int</code> 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

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

MAPPING_GetInterfaceAddressForInterface Gives the Interface address for an interface.	NodeAddress MAPPING_GetInterfaceAddressForInterface (Node* node, NodeAddress nodeId, int interfaceIndex) Parameters: <ul style="list-style-type: none">node - A pointer to the node being initialized.nodeId - Node idinterfaceIndex - Interface index Returns: <ul style="list-style-type: none">NodeAddress - Interface address for an interface.
MAPPING_GetInterfaceAddressForInterface Get node interface Address according to the network specific interface index. Overloaded function for ATM compatibility.	Address MAPPING_GetInterfaceAddressForInterface (NetworkType netType, int relativeInfInx) Parameters: <ul style="list-style-type: none">netType - Network type of the interface.relativeInfInx - Inerface index related to networkType. Returns: <ul style="list-style-type: none">Address - Return Address.
MAPPING_GetNodeIdFromInterfaceAddress Gives Node id from an interface address.	NodeAddress MAPPING_GetNodeIdFromInterfaceAddress (Node* node, NodeAddress interfaceAddress) Parameters: <ul style="list-style-type: none">node - A pointer to node being initialized.interfaceAddress - Interface address Returns: <ul style="list-style-type: none">NodeAddress - None
MAPPING_GetNodeIdFromInterfaceAddress Gives Node id from an interface address. Overloaded for IPv6	NodeAddress MAPPING_GetNodeIdFromInterfaceAddress (Node* node, Address interfaceAddress) Parameters: <ul style="list-style-type: none">node - A pointer to node being initialized.interfaceAddress - Interface address Returns: <ul style="list-style-type: none">NodeAddress - None
MAPPING_GetDefaultInterfaceAddressFromNodeId	NodeAddress MAPPING_GetDefaultInterfaceAddressFromNodeId (Node* node, NodeAddress nodeId) Parameters:

<p>Gives default interface address from a node id.</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.
<p>MAPPING_GetNumNodesInSubnet</p> <p>Gives the number of nodes in a subnet.</p>	<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.
<p>MAPPING_GetSubnetAddressCounter</p> <p>Gives the subnet address counter.</p>	<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.
<p>MAPPING_UpdateSubnetAddressCounter</p> <p>Updates the subnet address counter.</p>	<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
<p>MAPPING_GetInterfaceIndexFromInterfaceAddress</p> <p>Gets the node's interface index for the given address.</p>	<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.
<p>MAPPING_GetNodeInfoFromAtmNetInfo</p> <p>Get node interface Address, generic interfaceIndex and Atm related interfaceIndex from ATM Network information.</p>	<p>Address MAPPING_GetNodeInfoFromAtmNetInfo (unsigned int* index, unsigned int* genIndex)</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>unsigned int MAPPING_GetInterfaceIdForDestAddress (Node* node, NodeId nodeId, NodeAddress destAddr)</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>NodeAddress MAPPING_GetSubnetMaskForDestAddress (Node* node, NodeId nodeId, NodeAddress destAddr)</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>NodeAddress MAPPING_GetInterfaceAddrForNodeIdAndIntfId (Node* node, NodeId nodeId, int intfId)</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.

	<div>Returns:</div> <div><ul style="list-style-type: none">NodeAddress - None</div>
<div>MAPPING_GetIPv6NetworkAddressCounter</div> <div>Get IPV6 network address counter.</div>	<div>unsigned int MAPPING_GetIPv6NetworkAddressCounter (AddressMapType* map, in6_addr subnetAddr, unsigned int subnetPrefixLen)</div> <div>Parameters:</div> <div><ul style="list-style-type: none">map - The address map.subnetAddr - The IPv6 address.subnetPrefixLen - The prefix length.</div> <div>Returns:</div> <div><ul style="list-style-type: none">unsigned int - The current counter.</div>
<div>MAPPING_UpdateIPv6NetworkAddressCounter</div> <div>Update IPV6 network address counter.</div>	<div>void MAPPING_UpdateIPv6NetworkAddressCounter (AddressMapType* map, in6_addr subnetAddr, unsigned int subnetPrefixLen, int addressCounter)</div> <div>Parameters:</div> <div><ul style="list-style-type: none">map - The address map.subnetAddr - The IPv6 address.subnetPrefixLen - The prefix length.addressCounter - The new counter value.</div> <div>Returns:</div> <div><ul style="list-style-type: none">void - None</div>
<div>MAPPING_GetNumNodesInIPv6Network</div> <div>Get Num of nodes in IPV6 network.</div>	<div>unsigned int MAPPING_GetNumNodesInIPv6Network (Node* node, in6_addr subnetAddr, unsigned int subnetPrefixLen)</div> <div>Parameters:</div> <div><ul style="list-style-type: none">node - The pointer to the node.subnetAddr - The IPv6 address.subnetPrefixLen - The prefix length.</div> <div>Returns:</div> <div><ul style="list-style-type: none">unsigned int - None</div>
<div>MAPPING_GetNetworkIPVersion</div>	<div>NetworkType MAPPING_GetNetworkIPVersion (const char* addrString)</div> <div>Parameters:</div>

Get Network version IPv4/IPv6.	<ul style="list-style-type: none">addrString - The address string Returns: <ul style="list-style-type: none">NetworkType - None
MAPPING_GetNetworkType Identify network type from addrString.	NetworkType MAPPING_GetNetworkType (const char* addrString) Parameters: <ul style="list-style-type: none">addrString - The address string Returns: <ul style="list-style-type: none">NetworkType - None
MAPPING_GetIpv6InterfaceInfoForInterface Get IPV6 interface information for a interface.	void MAPPING_GetIpv6InterfaceInfoForInterface (Node *node node, NodeId nodeId nodeId, int interfaceIndex, in6_addr* globalAddr, in6_addr* subnetAddr, unsigned int* subnetPrefixLen) Parameters: <ul style="list-style-type: none">node - The node.nodeId - Node IdinterfaceIndex - The interface index.globalAddr - The global IPv6 address.subnetAddr - The subnet IPv6 address.subnetPrefixLen - THe subnet prefex length. Returns: <ul style="list-style-type: none">void - None
MAPPING_GetIpv6GlobalAddress Get IPV6 global address.	BOOL MAPPING_GetIpv6GlobalAddress (Node *node node, NodeId nodeId nodeId, unsigned int tla, unsigned int nla, unsigned int sla, in6_addr * addr6) Parameters: <ul style="list-style-type: none">node - The nodenodeId - The node's idtla - Top level aggregationnla - Next level aggregationsla - Site level aggregationaddr6 - The global IPv6 address. Returns:

	<ul style="list-style-type: none">• <code>BOOL</code> - None
<div><div>MAPPING_GetIpv6GlobalAddressForInterface</div><div>Get IPV6 global address for a node's nth interface.</div></div>	<div><div>BOOL MAPPING_GetIpv6GlobalAddressForInterface (Node * node, NodeId nodeId, int interfaceIndex, in6_addr * addr6, BOOL isDeprecated)</div><div>Parameters:<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)</div><div>Returns:<ul style="list-style-type: none">• <code>BOOL</code> - None</div></div>
<div><div>MAPPING_CreateIpv6GlobalUnicastAddr</div><div>Create IPv6 Global Unicast Address from tla nla sla</div></div>	<div><div>void MAPPING_CreateIpv6GlobalUnicastAddr (unsigned int tla, unsigned int nla, unsigned int sla, int addressCounter, in6_addr* globalAddr)</div><div>Parameters:<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.</div><div>Returns:<ul style="list-style-type: none">• <code>void</code> - None</div></div>
<div><div>MAPPING_CreateIpv6GlobalUnicastAddr</div><div>Create IPv6 Global Unicast Address.</div></div>	<div><div>void MAPPING_CreateIpv6GlobalUnicastAddr (AddressMapType * map, in6_addr IPv6subnetAddress, unsigned int IPv6subnetPrefixLen, int addressCounter, in6_addr* globalAddr)</div><div>Parameters:<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.</div></div>

	<div>Returns:</div> <div><ul style="list-style-type: none">void - None</div>
<div>MAPPING_CreateIpv6LinkLocalAddr</div> <div>Create IPv6 link local Address.</div>	<div>void MAPPING_CreateIpv6LinkLocalAddr (Node* node, Int32 interfaceId, in6_addr* globalAddr, in6_addr* linkLocalAddr, unsigned int subnetPrefixLen)</div> <div>Parameters:</div> <div><ul style="list-style-type: none">node - pointer to node structureinterfaceId - interface IdglobalAddr - The global IPv6 address.linkLocalAddr - The subnet IPv6 address.subnetPrefixLen - The subnet prefix length.</div> <div>Returns:</div> <div><ul style="list-style-type: none">void - None</div>
<div>MAPPING_CreateIpv6SiteLocalAddr</div> <div>Create IPv6 site local Address.</div>	<div>void MAPPING_CreateIpv6SiteLocalAddr (in6_addr* globalAddr, in6_addr* siteLocalAddr, unsigned short siteCounter, unsigned int subnetPrefixLen)</div> <div>Parameters:</div> <div><ul style="list-style-type: none">globalAddr - The global IPv6 address.siteLocalAddr - The subnet IPv6 address.siteCounter - The counter to use.subnetPrefixLen - The subnet prefix length.</div> <div>Returns:</div> <div><ul style="list-style-type: none">void - None</div>
<div>MAPPING_CreateIpv6MulticastAddr</div> <div>Create ipv6 multicast address.</div>	<div>void MAPPING_CreateIpv6MulticastAddr (in6_addr* globalAddr, in6_addr* multicastAddr)</div> <div>Parameters:</div> <div><ul style="list-style-type: none">globalAddr - The global IPv6 address.multicastAddr - The multicast IPv6 address.</div> <div>Returns:</div> <div><ul style="list-style-type: none">void - None</div>
<div>MAPPING_CreateIpv6SubnetAddr</div>	<div>void MAPPING_CreateIpv6SubnetAddr (unsigned int tla, unsigned int nla, unsigned int sla, unsigned int* IPv6subnetPrefixLen, in6_addr* IPv6subnetAddress)</div>

create subnet addr for IPV6 address.	<p>Parameters:</p> <ul style="list-style-type: none">• tla - Top level aggregation.• nla - Next level aggregation.• sla - Site level aggregation.• IPv6subnetPrefixLen - The IPv6 prefix length.• IPv6subnetAddress - The IPv6 subnet address. <p>Returns:</p> <ul style="list-style-type: none">• void - None
<p>MAPPING_GetNodeIdFromGlobalAddr</p> <p>Get node id from Global Address.</p>	<p>NodeId MAPPING_GetNodeIdFromGlobalAddr (Node * node, in6_addr* globalAddr)</p> <p>Parameters:</p> <ul style="list-style-type: none">• node - The node.• globalAddr - The global IPv6 address. <p>Returns:</p> <ul style="list-style-type: none">• NodeId - None
<p>MAPPING_GetNodeIdFromLinkLayerAddr</p> <p>Get node id from Link layer Address.</p>	<p>NodeId MAPPING_GetNodeIdFromLinkLayerAddr (Node * node, NodeAddress linkLayerAddr)</p> <p>Parameters:</p> <ul style="list-style-type: none">• node - The node.• linkLayerAddr - The link layer address. <p>Returns:</p> <ul style="list-style-type: none">• NodeId - None
<p>MAPPING_CreateIpv6LinkLayerAddr</p> <p>Create IPv6 link layer Address.</p>	<p>NodeAddress MAPPING_CreateIpv6LinkLayerAddr (unsigned int nodeId, int interfaceId)</p> <p>Parameters:</p> <ul style="list-style-type: none">• nodeId - The node's id.• interfaceId - The interface id. <p>Returns:</p> <ul style="list-style-type: none">• NodeAddress - None
<p>MAPPING_IsIpv6AddressOfThisNode</p>	<p>BOOL MAPPING_IsIpv6AddressOfThisNode (Node* node, const NodeAddress nodeId, in6_addr* globalAddr)</p>

checks whether the ipv6 address is of this node.	<p>Parameters:</p> <ul style="list-style-type: none">node - The node id.nodeId - The node's address.globalAddr - The global IPv6 address. <p>Returns:</p> <ul style="list-style-type: none">BOOL - None
<p>MAPPING_IsNodeInThisIpRange</p> <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
<p>MAPPING_IsIpAddressOfThisNode</p> <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
<p>MAPPING_GetInterfaceAddressForSubnet</p> <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.

	<ul style="list-style-type: none">• <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
<p>MAPPING_GetInterfaceAddressForSubnet</p> <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
<p>MAPPING_GetInterfaceAddressForSubnet</p> <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
<p>; MAPPING_GetInterfaceAddressForSubnet</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>Get interface address for subnet using tla nla sla.</p>	<p>Parameters:</p> <ul style="list-style-type: none">map - The address map.nodeId - The node id.tla - Top level aggregation.nla - Next level aggregation.sla - Site level aggregation.ipv6Addr - The ipv6 interface address.interfaceIndex - The interface index. <p>Returns:</p> <ul style="list-style-type: none">BOOL - None
<p>MAPPING_GetInterfaceFromLinkLayerAddress</p> <p>Get interface from link layer address.</p>	<p>int MAPPING_GetInterfaceFromLinkLayerAddress (Node* node, const NodeAddress linkLayerAddr)</p> <p>Parameters:</p> <ul style="list-style-type: none">node - The node.linkLayerAddr - The link layer address. <p>Returns:</p> <ul style="list-style-type: none">int - None
<p>MAPPING_GetInterfaceIndexFromInterfaceAddress</p> <p>Get interface index from interface address.</p>	<p>int MAPPING_GetInterfaceIndexFromInterfaceAddress (Node* node, Address interfaceAddress)</p> <p>Parameters:</p> <ul style="list-style-type: none">node - The node.interfaceAddress - The interface address. <p>Returns:</p> <ul style="list-style-type: none">int - None
<p>MAPPING_GetIpv6GlobalAddress</p> <p>Get ipv6 global address</p>	<p>BOOL MAPPING_GetIpv6GlobalAddress (Node* node, NodeId nodeId, in6_addr subnetAddr, UInt32 prefixLen, in6_addr* addr6)</p> <p>Parameters:</p> <ul style="list-style-type: none">node - The node.nodeId - The node idsubnetAddr - The subnet address.

	<ul style="list-style-type: none">• <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
<p>MAPPING_GetDefaultInterfaceAddressInfoFromNodeId</p> <p>Get default interface address based on network type</p>	<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
<p>Mapping_AutoCreateIPv6SubnetAddress</p> <p>Create IPv6 Testing Address Prefix (RFC 2471)from : ipv4 address.</p>	<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
<p>MAPPING_GetSubnetAddressFromInterfaceAddress</p> <p>Get subnet address from interface address.</p>	<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
<p>MAPPING_GetSubnetAddressFromInterfaceAddress</p> <p>Get ipv6 network Prefix from interface address.</p>	<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.

	<ul style="list-style-type: none">• <code>ipv6InterfaceAddr</code> - The IPv6 interface address.• <code>ipv6SubnetAddr</code> - The subnet address pointer . Returns: <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">• <code>node</code> - The node.• <code>ipv6InterfaceAddr</code> - The IPV6 interface address.• <code>prefixLenth</code> - The interface prefix length. Returns: <ul style="list-style-type: none">• <code>BOOL</code> - 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">• <code>nodeId</code> - The node id.• <code>nodeInput</code> - The node input file Returns: <ul style="list-style-type: none">• <code>NetworkProtocolType</code> - None
MAPPING_GetNetworkTypeFromInterface This function determines the network type of a particular interface of a node	NetworkType MAPPING_GetNetworkTypeFromInterface (Node* node, Int32 interfaceIndex) Parameters: <ul style="list-style-type: none">• <code>node</code> - Pointer to a node• <code>interfaceIndex</code> - interface index Returns: <ul style="list-style-type: none">• <code>NetworkType</code> - network type of an interface of a node



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 © 2001-2013 [SCALABLE Network Technologies, Inc.](#) All rights reserved.

QualNet 7.1 API Reference

MEMORY

This file describes the memory management data structures and functions.

Constant / Data Structure Summary

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

Function / Macro Summary

Return Type	Summary
void	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_InitializeThreadData (MemoryUsageData* data) Sets the partition-specific memory data for this partition.
void	MEM_PrintThreadData () Prints the partition-specific memory data.
void	MEM_ReportPartitionUsage (int partitionId, UInt32 totalAllocatedMemory, UInt32 totalFreedMemory, UInt32 totalPeakUsage) Prints out the total memory used by this partition.
void	MEM_ReportTotalUsage (UInt32 totalAllocatedMemory, UInt32 totalFreedMemory, UInt32 totalPeakUsage) 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.

Constant / Data Structure Detail

Structure	<div>MemoryUsageData</div> <div>Defines the parameters collected by the memory system. Restricted to kernel use.</div>

Function / Macro Detail

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

	<ul style="list-style-type: none">totalPeakUsage - peak usage of allocated memory Returns: <ul style="list-style-type: none">void - None
MEM_ReportTotalUsage 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.	void MEM_ReportTotalUsage (UInt32 totalAllocatedMemory, UInt32 totalFreedMemory, UInt32 totalPeakUsage) Parameters: <ul style="list-style-type: none">totalAllocatedMemory - sum of all MEM_malloc callstotalFreedMemory - sum of all MEM_free callstotalPeakUsage - peak usage of allocated memory Returns: <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 © 2001-2013 [SCALABLE Network Technologies, Inc.](#) All rights reserved.



QualNet 7.1 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*	MESSAGE_Alloc (PartitionData* partition, int layerType, int protocol, int eventType)
	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*	MESSAGE_AllocMT (PartitionData* partition, int layerType, int protocol, int eventType)
	Mutli-thread safe version of MESSAGE_Alloc for use by worker threads.
char*	MESSAGE_InfoFieldAlloc (Node* node, int infoSize)
	Allocate space for one "info" field
char*	MESSAGE_InfoFieldAlloc (PartitionData* partition, int infoSize)
	Allocate space for one "info" field
char*	MESSAGE_InfoFieldAllocMT (PartitionData* partition, int infoSize)
	Multi-thread safe version of MESSAGE_InfoFieldAlloc
void	MESSAGE_InfoFieldFree (Node* node, MessageInfoHeader* hdrPtr)
	Free space for one "info" field
char*	MESSAGE_AddInfo (Node* node, Message* msg, int infoSize, unsigned short infoType)
	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.
char*	MESSAGE_AddInfo (PartitionData* partition, Message* msg, int infoSize, unsigned short infoType)
	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.
void	MESSAGE_RemoveInfo (Node* node, Message* msg, unsigned short infoType)
	Remove one "info" field with given info type from the info array of the message.
char *	MESSAGE_InfoAlloc (Node* node, Message* msg, int infoSize)

	<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>
char *	<p>MESSAGE_InfoAlloc(PartitionData* partition, Message* msg, int infoSize)</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>
int	<p>MESSAGE_ReturnInfoSize(Message* msg, unsigned short infoType, int fragmentNumber)</p>
	<p>Returns the size of a "info" field with given info type in the info array of the message.</p>
int	<p>MESSAGE_ReturnInfoSize(Message* msg, unsigned short infoType)</p>
	<p>Returns the size of a "info" field with given info type in the info array of the message.</p>
char*	<p>MESSAGE_ReturnInfo(Message* msg, unsigned short infoType)</p>
	<p>Returns a pointer to the "info" field with given info type in the info array of the message.</p>
void	<p>MESSAGE_CopyInfo(Node* node, Message* dsgMsg, Message* srcMsg)</p>
	<p>Copy the "info" fields of the source message to the destination message.</p>
void	<p>MESSAGE_CopyInfo(Node* node, Message* dsgMsg, MessageInfoHeader* srcInfo)</p>
	<p>Copy the "info" fields of the source info header to the destination message.</p>
void	<p>MESSAGE_FragmentPacket(Node* node, Message* msg, int fragUnit, Message*** fragList, int* numFrag, TraceProtocolType protocolType)</p>
	<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>
Message*	<p>MESSAGE_ReassemblePacket(Node* node, Message** fragList, int numFrag, TraceProtocolType protocolType)</p>
	<p>Reassemble multiple fragments into one packet Note: All the fragments will be freed in this function.</p>
Message*	<p>MESSAGE_PackMessage(Node* node, Message* msgList, TraceProtocolType origProtocol, int* actualPktSize)</p>
	<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>
Message*	<p>MESSAGE_UnpackMessage(Node* node, Message* msg, bool copyInfo, bool freeOld)</p>

	<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>MESSAGE_PacketAlloc(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>MESSAGE_PacketAlloc(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>MESSAGE_AddHeader(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>MESSAGE_RemoveHeader(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>MESSAGE_ReturnHeader(Message* msg, int header)</p> <p>This is kind of a hack so that MAC protocols (dot11) that need to peak 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>MESSAGE_ExpandPacket(Node* node, Message* msg, int size)</p> <p>Expand packet by a specified size</p>
void	<p>MESSAGE_ShrinkPacket(Node* node, Message* msg, int size)</p> <p>This function is called to shrink packet by a specified size.</p>
void	<p>MESSAGE_Free(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>MESSAGE_Free(Node* node, 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>MESSAGE_FreeList(Node* node, Message* msg)</p> <p>Free a list of message until the next pointer of the message is NULL.</p>
Message*	<p>MESSAGE_Duplicate(Node* node, Message* msg)</p> <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>
Message*	<p>MESSAGE_Duplicate(PartitionData* partition, Message* msg, bool isMT)</p> <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>
Message*	<p>MESSAGE_DuplicateMT(PartitionData* partition, Message* msg)</p> <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>
char*	<p>MESSAGE_PayloadAlloc(Node* node, int payloadSize)</p> <p>Allocate a character payload out of the free list, if possible otherwise via malloc.</p>
char*	<p>MESSAGE_PayloadAlloc(PartitionData* partition, int payloadSize, bool isMT)</p> <p>Allocate a character payload out of the free list, if possible otherwise via malloc.</p>
void	<p>MESSAGE_PayloadFree(PartitionData* partition, Char* payload, int payloadSize)</p> <p>Return a character payload to the free list, if possible otherwise free it.</p>
void	<p>MESSAGE_PayloadFree(Node* node, Char* payload, int payloadSize)</p> <p>Return a character payload to the free list, if possible otherwise free it.</p>
void	<p>MESSAGE_FreeList(Node* node, Message* msg)</p> <p>Free a list of messages until the next pointer of the message is NULL.</p>
int	<p>MESSAGE_ReturnNumFrags (Message* msg)</p> <p>Returns the number of fragments used to create a TCP packet.</p>
int	<p>MESSAGE_ReturnFragSeqNum(Message* msg, int fragmentNumber)</p>

	Returns the sequence number of a particular fragments in the TCP packet.
int	MESSAGE_ReturnFragSize (Message* msg, int fragmentNumber)
	Returns the size of a particular fragment in the TCP packet.
int	MESSAGE_ReturnFragNumInfos (Message* msg, int fragmentNumber)
	Returns the number of info fields associated with a particular fragment in the TCP packet.
void	MESSAGE_AppendInfo (PartitionData* partitionData, Message* msg, int infosize, short infoType)
	Appends the "info" fields of the source message to the destination message.
void	MESSAGE_AppendInfo (Node* node, Message* msg, int infosize, short infoType)
	Appends the "info" fields of the source message to the destination message.
void	MESSAGE_AppendInfo (Node* node, Message* dsgMsg, MessageInfoHeader* srcInfo)
	Appends the "info" fields of the source message to the destination message.
void	MESSAGE_AppendInfo (Node* node, Message* dsgMsg, Message* srcMsg)
	Appends the "info" fields of the source message to the destination message.
size_t	MESSAGE_SizeOf ()
	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.
BOOL	MESSAGE_FragmentPacket (Node* node, Message& msg, Message& fragmentedMsg, Message& remainingMsg, int fragUnit, TraceProtocolType protocolType, bool freeOriginalMsg)
	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*	MESSAGE_ReassemblePacket (Node* node, Message* fragMsg1, Message* fragMsg2, TraceProtocolType protocolType)
	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
void	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	<div>MSG_MAX_HDR_SIZE 512</div> <div>Maximum Header Size</div>
Constant	<div>SMALL_INFO_SPACE_SIZE 112</div> <div>Size of small Info field. Should be larger than all commonly used info field data structures, especially PropTxInfo and PropRxInfo.</div>
Constant	<div>MSG_PAYLOAD_LIST_MAX 1000</div> <div>Maximum message payload list</div>
Constant	<div>MAX_CACHED_PAYLOAD_SIZE 1024</div> <div>Maximum cached payload size</div>
Constant	<div>MSG_INFO_LIST_MAX 1000</div> <div>Maximum message info list</div>
Constant	<div>MAX_INFO_FIELDS 12</div> <div>Maximum number of info fields</div>
Constant	<div>MAX_HEADERS 10</div> <div>Maximum number of headers</div>
Enumeration	<div>MessageInfoType</div> <div>Type of information in the info field. One message can only have up to one info field with a specific info type.</div>
Structure	<div>MessageInfoHeader</div>

	This is a structure which contains information about a info field.
Structure	<div>Message</div> <div>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.</div>

Function / Macro Detail

Function / Macro	Format
MESSAGE_PrintMessage	<div>void MESSAGE_PrintMessage (Node* node, Message* msg)</div> <div>Parameters:</div> <div><ul style="list-style-type: none">node - node which is sending messagemsg - message to be printed</div> <div>Returns:</div> <div><ul style="list-style-type: none">void - NULL</div>
MESSAGE_Send 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.	<div>void MESSAGE_Send (Node* node, Message* msg, clocktype delay, bool isMT)</div> <div>Parameters:</div> <div><ul style="list-style-type: none">node - node which is sending messagemsg - message to be delivereddelay - delay suffered by this message.isMT - is the function being called from a thread?</div> <div>Returns:</div> <div><ul style="list-style-type: none">void - NULL</div>
MESSAGE_SendMT Function call used to send a message from independent threads running within QualNet, for example those associated with external interfaces.	<div>void MESSAGE_SendMT (Node* node, Message* msg, clocktype delay)</div> <div>Parameters:</div> <div><ul style="list-style-type: none">node - node which is sending messagemsg - message to be delivereddelay - delay suffered by this message.</div> <div>Returns:</div> <div>void</div>

	- NULL
<p>MESSAGE_RemoteSend</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>	<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
<p>MESSAGE_RouteReceivedRemoteEvent</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>	<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
<p>MESSAGE_CancelSelfMsg</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>	<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
<p>MESSAGE_Alloc</p> <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 (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
<p>MESSAGE_Alloc</p> <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 <code>layerType</code>, <code>protocol</code> and the <code>eventType</code> that will be set for this message.</p>	<p><code>Message*</code> MESSAGE_Alloc (<code>PartitionData*</code> 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
<p>MESSAGE_AllocMT</p> <p>Mutli-thread safe version of <code>MESSAGE_Alloc</code> for use by worker threads.</p>	<p><code>Message*</code> MESSAGE_AllocMT (<code>PartitionData*</code> 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
<p>MESSAGE_InfoFieldAlloc</p> <p>Allocate space for one "info" field</p>	<p><code>char*</code> MESSAGE_InfoFieldAlloc (<code>Node*</code> 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.
<p>MESSAGE_InfoFieldAlloc</p> <p>Allocate space for one "info" field</p>	<p><code>char*</code> MESSAGE_InfoFieldAlloc (<code>PartitionData*</code> 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">char* - pointer to the allocated space.
<p>MESSAGE_InfoFieldAllocMT</p> <p>Multi-thread safe version of MESSAGE_InfoFieldAlloc</p>	<p>char* MESSAGE_InfoFieldAllocMT (PartitionData* partition, int infoSize)</p> <p>Parameters:</p> <ul style="list-style-type: none">partition - partition which is allocating the space.infoSize - size of the space to be allocated <p>Returns:</p> <ul style="list-style-type: none">char* - pointer to the allocated space.
<p>MESSAGE_InfoFieldFree</p> <p>Free space for one "info" field</p>	<p>void MESSAGE_InfoFieldFree (Node* node, MessageInfoHeader* hdrPtr)</p> <p>Parameters:</p> <ul style="list-style-type: none">node - node which is allocating the space.hdrPtr - pointer to the "info" field <p>Returns:</p> <ul style="list-style-type: none">void - NULL
<p>MESSAGE_AddInfo</p> <p>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.</p>	<p>char* MESSAGE_AddInfo (Node* node, Message* msg, int infoSize, unsigned short infoType)</p> <p>Parameters:</p> <ul style="list-style-type: none">node - node which is allocating the info field.msg - message for which "info" fieldinfoSize - size of the "info" field to be allocatedinfoType - type of the "info" field to be allocated. <p>Returns:</p> <ul style="list-style-type: none">char* - Pointer to the added info field
<p>MESSAGE_AddInfo</p> <p>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.</p>	<p>char* MESSAGE_AddInfo (PartitionData* partition, Message* msg, int infoSize, unsigned short infoType)</p> <p>Parameters:</p> <ul style="list-style-type: none">partition - partition which is allocating the info field.msg - message for which "info" fieldinfoSize - 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>void MESSAGE_RemoveInfo (Node* node, Message* msg, unsigned short infoType)</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>char * MESSAGE_InfoAlloc (Node* node, Message* msg, int infoSize)</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>char * MESSAGE_InfoAlloc (PartitionData* partition, Message* msg, int infoSize)</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>int MESSAGE_ReturnInfoSize (Message* msg, unsigned short infoType, int fragmentNumber)</p> <p>Parameters:</p>

Returns the size of a "info" field with given info type in the info array of the message.	<p><code>msg</code> - message for which "info" field</p> <ul style="list-style-type: none"><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 Returns the size of a "info" field with given info type in the info array of the message.	<p><code>int MESSAGE_ReturnInfoSize (Message* msg, unsigned short infoType)</code></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 Returns a pointer to the "info" field with given info type in the info array of the message.	<p><code>char* MESSAGE_ReturnInfo (Message* msg, unsigned short infoType)</code></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 Copy the "info" fields of the source message to the destination message.	<p><code>void MESSAGE_CopyInfo (Node* node, Message* dsgMsg, Message* srcMsg)</code></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 Copy the "info" fields of the source info header to the destination message.	<p><code>void MESSAGE_CopyInfo (Node* node, Message* dsgMsg, MessageInfoHeader* srcInfo)</code></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
<p>MESSAGE_FragmentPacket</p> <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* numFrag, 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>numFrag</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
<p>MESSAGE_ReassemblePacket</p> <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 numFrag, 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>numFrag</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.
<p>MESSAGE_PackMessage</p> <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 list 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

	<div>Returns:</div> <ul style="list-style-type: none">• <code>Message*</code> - The super msg contains a list of msgs as payload
<div>MESSAGE_UnpackMessage</div> <div>Unpack a super message to the original list of messages The list of messages were stored as payload of this super message.</div>	<div>Message* MESSAGE_UnpackMessage (Node* node, Message* msg, bool copyInfo, bool freeOld)</div> <div>Parameters:</div> <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 <div>Returns:</div> <ul style="list-style-type: none">• <code>Message*</code> - A list of messages unpacked from original msg
<div>MESSAGE_PacketAlloc</div> <div>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.</div>	<div>void MESSAGE_PacketAlloc (Node* node, Message* msg, int packetSize, TraceProtocolType originalProtocol)</div> <div>Parameters:</div> <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 <div>Returns:</div> <ul style="list-style-type: none">• <code>void</code> - NULL
<div>MESSAGE_PacketAlloc</div> <div>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.</div>	<div>void MESSAGE_PacketAlloc (PartitionData* partition, Message* msg, int packetSize, TraceProtocolType originalProtocol, bool isMT)</div> <div>Parameters:</div> <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 <div>Returns:</div> <ul style="list-style-type: none">• <code>void</code> - NULL

<div>MESSAGE_AddHeader</div> <div>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.</div>	<div>void MESSAGE_AddHeader (Node* node, Message* msg, int hdrSize, TraceProtocolType traceProtocol)</div> <div>Parameters:</div> <div><ul style="list-style-type: none">node - node which is adding headermsg - message for which header has to be addedhdrSize - size of the header to be addedtraceProtocol - protocol name, from trace.h</div> <div>Returns:</div> <div><ul style="list-style-type: none">void - NULL</div>
<div>MESSAGE_RemoveHeader</div> <div>This function is called to remove a header from the packet. The "packetSize" variable in the message will be decreased by "hdrSize".</div>	<div>void MESSAGE_RemoveHeader (Node* node, Message* msg, int hdrSize, TraceProtocolType traceProtocol)</div> <div>Parameters:</div> <div><ul style="list-style-type: none">node - node which is removing the packet headermsg - message for which header is being removedhdrSize - size of the header being removedtraceProtocol - protocol removing this header.</div> <div>Returns:</div> <div><ul style="list-style-type: none">void - NULL</div>
<div>MESSAGE_ReturnHeader</div> <div>This is kind of a hack so that MAC protocols (dot11) that need to peak at a packet that still has the PHY header can return the contents after the first (N) headers without first removing those headers.</div>	<div>char* MESSAGE_ReturnHeader (Message* msg, int header)</div> <div>Parameters:</div> <div><ul style="list-style-type: none">msg - message containing a packet with headersheader - number of the header to return.</div> <div>Returns:</div> <div><ul style="list-style-type: none">char* - the packet starting at the header'th header</div>
<div>MESSAGE_ExpandPacket</div> <div>Expand packet by a specified size</div>	<div>void MESSAGE_ExpandPacket (Node* node, Message* msg, int size)</div> <div>Parameters:</div> <div><ul style="list-style-type: none">node - node which is expanding the packetmsg - message which is to be expandedsize - size to expand</div>

	<div>Returns:</div> <div><ul style="list-style-type: none"><code>void</code> - NULL</div>
<div>MESSAGE_ShrinkPacket</div> <div>This function is called to shrink packet by a specified size.</div>	<div>void MESSAGE_ShrinkPacket (Node* node, Message* msg, int size)</div> <div>Parameters:</div> <div><ul style="list-style-type: none"><code>node</code> - node which is shrinking packet<code>msg</code> - message whose packet is be shrinked<code>size</code> - size to shrink</div> <div>Returns:</div> <div><ul style="list-style-type: none"><code>void</code> - NULL</div>
<div>MESSAGE_Free</div> <div>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.</div>	<div>void MESSAGE_Free (PartitionData* partition, Message* msg)</div> <div>Parameters:</div> <div><ul style="list-style-type: none"><code>partition</code> - partition which is freeing the message<code>msg</code> - message which has to be freed</div> <div>Returns:</div> <div><ul style="list-style-type: none"><code>void</code> - NULL</div>
<div>MESSAGE_Free</div> <div>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.</div>	<div>void MESSAGE_Free (Node* node, Message* msg)</div> <div>Parameters:</div> <div><ul style="list-style-type: none"><code>node</code> - node which is freeing the message<code>msg</code> - message which has to be freed</div> <div>Returns:</div> <div><ul style="list-style-type: none"><code>void</code> - NULL</div>
<div>MESSAGE_FreeList</div> <div>Free a list of message until the next pointer of the message is NULL.</div>	<div>void MESSAGE_FreeList (Node* node, Message* msg)</div> <div>Parameters:</div> <div><ul style="list-style-type: none"><code>node</code> - node which is freeing the message<code>msg</code> - message which has to be freed</div> <div>Returns:</div> <div><ul style="list-style-type: none"><code>void</code> - NULL</div>
<div>MESSAGE_Duplicate</div>	<div>Message* MESSAGE_Duplicate (Node* node, Message* msg)</div>

<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 copymsg - message for which duplicate has to be made <p>Returns:</p> <ul style="list-style-type: none">Message* - Pointer to the new message
<p>MESSAGE_Duplicate</p> <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>Message* MESSAGE_Duplicate (PartitionData* partition, Message* msg, bool isMT)</p> <p>Parameters:</p> <ul style="list-style-type: none">partition - partition is calling message copymsg - message for which duplicate has to be madeisMT - Is this function being called from the context <p>Returns:</p> <ul style="list-style-type: none">Message* - Pointer to the new message
<p>MESSAGE_DuplicateMT</p> <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>Message* MESSAGE_DuplicateMT (PartitionData* partition, Message* msg)</p> <p>Parameters:</p> <ul style="list-style-type: none">partition - partition is calling message copymsg - message for which duplicate has to be made <p>Returns:</p> <ul style="list-style-type: none">Message* - Pointer to the new message
<p>MESSAGE_PayloadAlloc</p> <p>Allocate a character payload out of the free list, if possible otherwise via malloc.</p>	<p>char* MESSAGE_PayloadAlloc (Node* node, int payloadSize)</p> <p>Parameters:</p> <ul style="list-style-type: none">node - node which is allocating payloadpayloadSize - size of the field to be allocated <p>Returns:</p> <ul style="list-style-type: none">char* - pointer to the allocated memory
<p>MESSAGE_PayloadAlloc</p> <p>Allocate a character payload out of the free list, if possible otherwise via malloc.</p>	<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 Returns: <ul style="list-style-type: none">• <code>char*</code> - pointer to the allocated memory
MESSAGE_PayloadFree Return a character payload to the free list, if possible otherwise free it.	void MESSAGE_PayloadFree (PartitionData* partition, Char* payload, int payloadSize) Parameters: <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 Returns: <ul style="list-style-type: none">• <code>void</code> - NULL
MESSAGE_PayloadFree Return a character payload to the free list, if possible otherwise free it.	void MESSAGE_PayloadFree (Node* node, Char* payload, int payloadSize) Parameters: <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 Returns: <ul style="list-style-type: none">• <code>void</code> - NULL
MESSAGE_FreeList Free a list of messages until the next pointer of the message is NULL.	void MESSAGE_FreeList (Node* node, Message* msg) Parameters: <ul style="list-style-type: none">• <code>node</code> - node which is freeing the message• <code>msg</code> - message which has to be freed Returns: <ul style="list-style-type: none">• <code>void</code> - NULL
MESSAGE_ReturnNumFrag Returns the number of fragments used to create a TCP packet.	int MESSAGE_ReturnNumFrag (Message* msg) Parameters: <ul style="list-style-type: none">• <code>msg</code> - message for which "info" field Returns:

	<ul style="list-style-type: none">• <code>int</code> - Number of Fragments. 0 if none.
<p>MESSAGE_ReturnFragSeqNum</p> <p>Returns the sequence number of a particular fragments in the TCP packet.</p>	<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.
<p>MESSAGE_ReturnFragSize</p> <p>Returns the size of a particular fragment in the TCP packet.</p>	<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.
<p>MESSAGE_ReturnFragNumInfos</p> <p>Returns the number of info fields associated with a particular fragment in the TCP packet.</p>	<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.
<p>MESSAGE_AppendInfo</p> <p>Appends the "info" fields of the source message to the destination message.</p>	<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 Appends the "info" fields of the source message to the destination message.	void MESSAGE_AppendInfo (Node* node, Message* msg, int infosize, short infoType) Parameters: <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. Returns: <ul style="list-style-type: none">• void - NULL
MESSAGE_AppendInfo Appends the "info" fields of the source message to the destination message.	void MESSAGE_AppendInfo (Node* node, Message* dsgMsg, MessageInfoHeader* srcInfo) Parameters: <ul style="list-style-type: none">• node - Node which is copying the info fields• dsgMsg - Destination message• srcInfo - Source message info vector Returns: <ul style="list-style-type: none">• void - NULL
MESSAGE_AppendInfo Appends the "info" fields of the source message to the destination message.	void MESSAGE_AppendInfo (Node* node, Message* dsgMsg, Message* srcMsg) Parameters: <ul style="list-style-type: none">• node - Node which is copying the info fields• dsgMsg - Destination message• srcMsg - Source message Returns: <ul style="list-style-type: none">• void - NULL
MESSAGE_SizeOf 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.	size_t MESSAGE_SizeOf () Parameters: Returns: <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 © 2001-2013 [SCALABLE Network Technologies, Inc.](#) All rights reserved.



QualNet 7.1 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 granurality
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	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 Summary

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

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

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

Constant / Data Structure Detail

Constant	DEFAULT_DISTANCE_GRANULARITY 1 Defines the default distance granurality
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	void MOBILITY_InsertEvent (MobilityHeap* heapPtr, Node* node)
Inserts an event.	Parameters: <ul style="list-style-type: none">heapPtr - A pointer of type MobilityHeap.node - A pointer to node. Returns:

	<ul style="list-style-type: none">void - None
MOBILITY_DeleteEvent Deletes an event.	<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 Inserts an event and sort out the heap downwards	<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 Allocates memory for nodePositions and mobilityData Note: This function is called before NODE_CreateNode(). It cannot access Node structure	<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 nodesnodeIdArray - array of nodeIdnodePositions - pointer to the arraynodePlacementTypeCounts - array of placement type countsnodeInput - configuration inputseedVal - seed for random number seeds <p>Returns:</p> <ul style="list-style-type: none">void - None
MOBILITY_PreInitialize Initializes most variables in mobilityData. (Node positions are set in MOBILITY_SetNodePositions().) Note: This	<p>void MOBILITY_PreInitialize (NodeAddress nodeId, MobilityData* mobilityData, NodeInput* nodeInput, int seedVal)</p> <p>Parameters:</p> <ul style="list-style-type: none">nodeId - nodeIdmobilityData - mobilityData to be initialized

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

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

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

Returns the node orientation.	<ul style="list-style-type: none">node - Pointer to node.orientation - Pointer to Orientation. Returns: <ul style="list-style-type: none">void - None
MOBILITY_ReturnInstantaneousSpeed Returns instantaneous speed of a node.	void MOBILITY_ReturnInstantaneousSpeed (Node* node, double* speed) Parameters: <ul style="list-style-type: none">node - Pointer to node.speed - Speed of the node, double pointer. Returns: <ul style="list-style-type: none">void - None
MOBILITY_ReturnSequenceNum Returns a sequence number for the current position.	void MOBILITY_ReturnSequenceNum (Node* node, int* sequenceNum) Parameters: <ul style="list-style-type: none">node - Pointer to node.sequenceNum - Sequence number. Returns: <ul style="list-style-type: none">void - None
MOBILITY_SetNodePositions Set positions of nodes	void MOBILITY_SetNodePositions (int numNodes, NodePositions* nodePositions, int* nodePlacementTypeCounts, TerrainData* terrainData, NodeInput* nodeInput, RandomSeed seed, clocktype maxSimTime) Parameters: <ul style="list-style-type: none">numNodes - Defines the number of nodes to be distributed.nodePositions - Pointer to NodePositionInfo. StatesnodePlacementTypeCounts - Array of placement type countsterrainData - Terrain data.nodeInput - Pointer to NodeInput, defines theseed - Stores the seed value.maxSimTime - Maximum simulation time. Returns: <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	void MOBILITY_PostInitializePartition (PartitionData* partitionData) Parameters: <ul style="list-style-type: none">partitionData - Pointer to the partition data Returns: <ul style="list-style-type: none">void - None
MOBILITY_NodePlacementFinalize Finalize mobility models	void MOBILITY_NodePlacementFinalize (PartitionData* partitionData) Parameters: <ul style="list-style-type: none">partitionData - Pointer to the partition data Returns: <ul style="list-style-type: none">void - None
MOBILITY_ChangeGroundNode Change GroundNode value..	void MOBILITY_ChangeGroundNode (Node* node, BOOL before, BOOL after) Parameters: <ul style="list-style-type: none">node - Pointer to node being initialized.before - Orginal value for Ground-Node variableafter - new value for Ground-Node variable. Returns: <ul style="list-style-type: none">void - None
MOBILITY_ChangePositionGranularity Change Mobility-Position-Granularity value..	void MOBILITY_ChangePositionGranularity (Node* node) Parameters: <ul style="list-style-type: none">node - Pointer to node being initialized. Returns: <ul style="list-style-type: none">void - None



[Qualnet®](#) is a Registered Trademark of [SCALABLE Network Technologies](#).

Copyright © 2001-2013 [SCALABLE Network Technologies, Inc.](#) All rights reserved.

QualNet 7.1 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	QNThreadLock (QNThreadMutex mutex) This constructor is used to begin a critical region.
None	QNPartitionLock (QNPartitionMutex mutex) This constructor is used to begin a critical region.

Function / Macro Detail

Function / Macro	Format
QNThreadLock This constructor is used to begin a critical region.	None QNThreadLock (QNThreadMutex mutex) Parameters: <ul style="list-style-type: none">mutex - Pointer to the Thread mutex to lock for this Returns: <ul style="list-style-type: none">None - None
QNPartitionLock This constructor is used to begin a critical region.	None QNPartitionLock (QNPartitionMutex mutex) Parameters: <ul style="list-style-type: none">mutex - Pointer to the Partition mutex to lock Returns: <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 © 2001-2013 [SCALABLE Network Technologies, Inc.](#) All rights reserved.



QualNet 7.1 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)
ENUMERATION	NetworkRoutingAdminDistanceType
	Administrative distance of different routing protocol
ENUMERATION	NetworkRoutingProtocolType
	Enlisted different network/routing protocol
ENUMERATION	ManagementReportType
	Type of management report message
ENUMERATION	ManagementResponseType
	Type of management response message
STRUCT	NetworkData
	Main data structure of network layer
STRUCT	ManagementReport
	data structure of management report
STRUCT	ManagementResponse
	data structure of management response

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

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	<p>DEFAULT_ETHERNET_MTU 1500</p> <p>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.</p>
Constant	<p>IP_MAXPACKET 65535</p> <p>Maximum IP packet size</p>
Constant	<p>NETWORK_IP_UNLIMITED_BACKPLANE_THROUGHPUT 0</p> <p>Maximum throughput of backplane of network.</p>
Enumeration	<p>NetworkIpBackplaneStatusType</p> <p>Status of backplane (either busy or idle)</p>
Enumeration	<p>NetworkRoutingAdminDistanceType</p> <p>Administrative distance of different routing protocol</p>
Enumeration	<p>NetworkRoutingProtocolType</p> <p>Enlisted different network/routing protocol</p>
Enumeration	<p>ManagementReportType</p> <p>Type of management report message</p>
Enumeration	<p>ManagementResponseType</p> <p>Type of management response message</p>
Structure	<p>NetworkData</p> <p>Main data structure of network layer</p>
Structure	<p>ManagementReport</p>

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

	<ul style="list-style-type: none">void - None
NetworkIpGetInterfaceType Returns type of network (ipv4 or ipv6) the interface.	NetworkType NetworkIpGetInterfaceType (Node* node, int interfaceIndex) Parameters: <ul style="list-style-type: none">node - Pointer to node.interfaceIndex - Interface index. Returns: <ul style="list-style-type: none">NetworkType - None
NETWORK_ReceivePacketFromMacLayer Network-layer receives packets from MAC layer, now check Overloaded Function to support Mac Address type of IP and call proper function	void NETWORK_ReceivePacketFromMacLayer (Node* node, Message* message, NodeAddress lastHopAddress, int interfaceIndex) Parameters: <ul style="list-style-type: none">node - Pointer to nodemessage - Message receivedlastHopAddress - last hop addressinterfaceIndex - incoimg interface Returns: <ul style="list-style-type: none">void - None
NETWORK_Reset Reset Network protocols and/or layer.	void NETWORK_Reset (Node* node, int interfaceIndex) Parameters: <ul style="list-style-type: none">node - Pointer to node.interfaceIndex - Interface index. Returns: <ul style="list-style-type: none">void - None
NETWORK_AddResetFunctionList Add which protocols to be reset to a fuction list pointer.	void NETWORK_AddResetFunctionList (Node* node, int interfaceIndex) Parameters: <ul style="list-style-type: none">node - Pointer to node.interfaceIndex - Interface index. Returns: <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 © 2001-2013 [SCALABLE Network Technologies, Inc.](#) All rights reserved.

QualNet 7.1 API Reference

NODE

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

Constant / Data Structure Summary

Type	Name
ENUMERATION	NodeGlobalIndex This enumeration contains indexes into the nodeGlobal array used for module data.
STRUCT	Node This struct includes all the information for a particular node. State information for each layer can be accessed from this structure.
STRUCT	NodePositions Contains information about the initial positions of nodes.

Function / Macro Summary

Return Type	Summary
clocktye	getNodeTime () Get current time at the node. When processing events, nodes should always use this function. The PartitionData::getGlobalTime should only be used for timing at the partition or global level.
void	NODE_CreateNode (PartitionData* partitionData, NodeId nodeId, int index) Function used to allocate and initialize a node.
void	NODE_ProcessEvent (Node* node, Message* msg) Function used to call the appropriate layer to execute instructions for the message
void	NODE_PrintLocation (Node* node, int coordinateSystemType)

	Prints the node's three dimensional coordinates.
TerrainData*	NODE_GetTerrainPtr (Node* node) Get terrainData pointer.

Constant / Data Structure Detail

Enumeration	NodeGlobalIndex This enumeration contains indexes into the nodeGlobal array used for module data.
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
getNodeTime Get current time at the node. When processing events, nodes should always use this function. The PartitionData::getGlobalTime should only be used for timing at the partition or global level.	clocktye getNodeTime () Parameters: Returns: <ul style="list-style-type: none">clocktye - The current time
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 nodenodeId - the node's IDindex - the node's index within the partition

	<p>Returns:</p> <ul style="list-style-type: none">void - None
<p>NODE_ProcessEvent</p> <p>Function used to call the appropriate layer to execute instructions for the message</p>	<p>void NODE_ProcessEvent (Node* node, Message* msg)</p> <p>Parameters:</p> <ul style="list-style-type: none">node - node for which message is to be deliveredmsg - message for which instructions are to be executed <p>Returns:</p> <ul style="list-style-type: none">void - None
<p>NODE_PrintLocation</p> <p>Prints the node's three dimensional coordinates.</p>	<p>void NODE_PrintLocation (Node* node, int coordinateSystemType)</p> <p>Parameters:</p> <ul style="list-style-type: none">node - the nodecoordinateSystemType - Cartesian or LatLonAlt <p>Returns:</p> <ul style="list-style-type: none">void - None
<p>NODE_GetTerrainPtr</p> <p>Get terrainData pointer.</p>	<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

QualNet 7.1 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.
ENUMERATION	BarrierType Type of barrier for synchronization integrity checking. There should be a unique value for each location in the code that calls the parallel processing barrier, either by a call to PARALLEL_SynchronizePartitions or to PARALLEL_GetRemoteMessagesAndBarrier. When adding a new barrier call, add a new enum value here to use.
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	<p>PARALLEL_AllocateLookaheadHandle(Node* node)</p> <p>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.</p>
void	<p>PARALLEL_AddLookaheadHandleToLookaheadCalculator(Node* node, LookaheadHandle lookaheadHandle, clocktype eotOfNode)</p> <p>Adds a new LookaheadHandle to the lookahead calculator.</p>
void	<p>PARALLEL_SetLookaheadHandleEOT(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>PARALLEL_RemoveLookaheadHandleFromLookaheadCalculator(Node* node, LookaheadHandle lookaheadHandle, clocktype* eotOfNode)</p> <p>Removes a LookaheadHandle from the lookahead calculator.</p>
void	<p>PARALLEL_SetMinimumLookaheadForInterface(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>PARALLEL_InitLookaheadCalculator(LookaheadCalculator lookaheadCalculator)</p> <p>Initializes lookahead calculation. For kernel use only.</p>
int	<p>PARALLEL_AssignNodesToPartitions(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>PARALLEL_GetPartitionForNode(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>PARALLEL_InitializeParallelRuntime(int numberOfThreads)</p> <p>Sets global variables and stuff. For kernel use only.</p>
void	<p>PARALLEL_CreatePartitionThreads(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>PARALLEL_GetRemoteMessages(PartitionData* partitionData)</p>

	<p>Collects all the messages received from other partitions. For kernel use only.</p>
void	<p>PARALLEL_GetRemoteMessagesAndBarrier(PartitionData* partitionData, BarrierType barrierType)</p> <p>Collects all the messages received from other partitions. This function also acts as a barrier. For kernel use only.</p>
	<p>PARALLEL_SendRemoteMessages(Message* msgList, PartitionData* partitionData, int partitionId)</p> <p>Sends one or more messages to a remote partition. For kernel use only.</p>
	<p>PARALLEL_DeliverRemoteMessages(PartitionData* partitionData)</p> <p>Delivers cached messages to all remote partitions. For kernel use only.</p>
void	<p>PARALLEL_SendRemoteMessagesOob(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>PARALLEL_SendMessageToAllPartitions(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>PARALLEL_SendRemoteLinkMessage(Node* node, Message* msg, LinkData* link, clocktype txDelay)</p> <p>Sends one LINK message to a remote partition.</p>
void	<p>PARALLEL_UpdateSafeTime(PartitionData* partitionData)</p> <p>A generic function for calculating the window of safe events For kernel use only.</p>
clocktype	<p>PARALLEL_ReturnEarliestGlobalEventTime(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>PARALLEL_Exit(PartitionData* partitionData)</p> <p>Exits from the parallel system, killing threads, etc. For kernel use only.</p>
void	<p>PARALLEL_SetProtocolIsNotEOTCapable(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</p>

	lookahead is used.
void	<div><div>PARALLEL_EnableDynamicMobility()</div><div>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.</div></div>
void	<div><div>PARALLEL_SetGreedy(bool greedy)</div><div>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.</div></div>
bool	<div><div>PARALLEL_IsGreedy()</div><div>Checks whether SetGreedy has been called.</div></div>
void	<div><div>PARALLEL_PreFlight(PartitionData* partitionData)</div><div>Initializes parallel operation.</div></div>
void	<div><div>PARALLEL_ScheduleMessagesOnPartition(PartitionData* partitionData, Message* msgList, Message** oobMessage, bool* gotOobMessage, bool isMT)</div><div>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.</div></div>
void	<div><div>PARALLEL_EndSimulation(PartitionData* partitionData)</div><div>Shuts down the parallel engine, including whatever synchronization is required.</div></div>
void	<div><div>PARALLEL_BuildStatFile(int numPartitions, char* statFileName, char* experimentPrefix)</div><div>Builds the final stat file when running in parallel node. Should only be called once from partition 0.</div></div>
void	<div><div>PARALLEL_NumberOfSynchronizations()</div><div>Return the number of synchronizations performed per partition</div></div>
void	<div><div>PARALLEL_StartRealTimeThread(PartitionData* partitionData)</div><div>Tells the kernel to use an independent thread to constantly update realtime.</div></div>

Constant	<p>MAX_THREADS 512</p> <p>The maximum number of processes that can be used in parallel QualNet. Customers do not receive parallel.cpp, so cannot effectively change this value.</p>
Enumeration	<p>SynchronizationAlgorithm</p> <p>Possible algorithms to use in the parallel runtime. Synchronous is used by default.</p>
Enumeration	<p>BarrierType</p> <p>Type of barrier for synchronization integrity checking. There should be a unique value for each location in the code that calls the parallel processing barrier, either by a call to PARALLEL_SynchronizePartitions or to PARALLEL_GetRemoteMessagesAndBarrier. When adding a new barrier call, add a new enum value here to use.</p>
Structure	<p>LookaheadLocator</p> <p>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.</p>
Structure	<p>EotHeapElement</p> <p>Basic data structure for simplifying lookahead calculation.</p>
Structure	<p>LookaheadCalculator</p> <p>Stores a heap of EOT elements to calculate lookahead.</p>

Function / Macro Detail

Function / Macro	Format
<p>PARALLEL_AllocateLookaheadHandle</p> <p>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.</p>	<p>LookaheadHandle PARALLEL_AllocateLookaheadHandle (Node* node)</p> <p>Parameters:</p> <ul style="list-style-type: none">node - the active node <p>Returns:</p> <ul style="list-style-type: none">LookaheadHandle - Returns a reference to the node's lookahead data.

<div><div>PARALLEL_AddLookaheadHandleToLookaheadCalculator</div><div>Adds a new LookaheadHandle to the lookahead calculator.</div></div>	<div><div>void PARALLEL_AddLookaheadHandleToLookaheadCalculator (Node* node, LookaheadHandle lookaheadHandle, clocktype eotOfNode)</div><div>Parameters:</div><div><ul style="list-style-type: none">node - the active nodelookaheadHandle - the node's lookahead handleeotOfNode - the node's EOT</div><div>Returns:</div><div><ul style="list-style-type: none">void - None</div></div>
<div><div>PARALLEL_SetLookaheadHandleEOT</div><div>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 <code>_must_</code> use EOT pervasively.</div></div>	<div><div>void PARALLEL_SetLookaheadHandleEOT (Node* node, LookaheadHandle lookaheadHandle, clocktype eot)</div><div>Parameters:</div><div><ul style="list-style-type: none">node - the active nodelookaheadHandle - the node's lookahead handleeot - the node's current EOT</div><div>Returns:</div><div><ul style="list-style-type: none">void - None</div></div>
<div><div>PARALLEL_RemoveLookaheadHandleFromLookaheadCalculator</div><div>Removes a LookaheadHandle from the lookahead calculator.</div></div>	<div><div>void PARALLEL_RemoveLookaheadHandleFromLookaheadCalculator (Node* node, LookaheadHandle lookaheadHandle, clocktype* eotOfNode)</div><div>Parameters:</div><div><ul style="list-style-type: none">node - the active nodelookaheadHandle - the node's lookahead handleeotOfNode - the node's current EOT</div><div>Returns:</div><div><ul style="list-style-type: none">void - None</div></div>
<div><div>PARALLEL_SetMinimumLookaheadForInterface</div><div>Sets a minimum delay for messages going out on this interface. This is typically set by the protocol running on that interface.</div></div>	<div><div>void PARALLEL_SetMinimumLookaheadForInterface (Node* node, clocktype minLookahead)</div><div>Parameters:</div><div><ul style="list-style-type: none">node - the active nodeminLookahead - the protocol's minimum lookahead</div><div>Returns:</div></div>

	<ul style="list-style-type: none">void - None
<p>PARALLEL_InitLookaheadCalculator</p> <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
<p>PARALLEL_AssignNodesToPartitions</p> <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">numNodes - the number of nodesnumberOfPartitions - the number of partitionsnodeInput - the input configuration filenodePos - the node positionsmap - node ID <--> IP address mappings <p>Returns:</p> <ul style="list-style-type: none">int - the number of partitions used
<p>PARALLEL_GetPartitionForNode</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>	<p>int PARALLEL_GetPartitionForNode (NodeId nodeId)</p> <p>Parameters:</p> <ul style="list-style-type: none">nodeId - the node's ID <p>Returns:</p> <ul style="list-style-type: none">int - the partition to which the node is assigned
<p>PARALLEL_InitializeParallelRuntime</p> <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">numberOfThreads - the number of processors to use. <p>Returns:</p> <ul style="list-style-type: none">void - None
<p>PARALLEL_CreatePartitionThreads</p>	<p>void PARALLEL_CreatePartitionThreads (int numberOfThreads, NodeInput* nodeInput, PartitionData* partitionArray)</p>

<p>Creates the threads for parallel execution and starts them running. For kernel use only.</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
<p>PARALLEL_GetRemoteMessages</p> <p>Collects all the messages received from other partitions. For kernel use only.</p>	<p><code>void PARALLEL_GetRemoteMessages (PartitionData* partitionData)</code></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
<p>PARALLEL_GetRemoteMessagesAndBarrier</p> <p>Collects all the messages received from other partitions. This function also acts as a barrier. For kernel use only.</p>	<p><code>void PARALLEL_GetRemoteMessagesAndBarrier (PartitionData* partitionData, BarrierType barrierType)</code></p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>partitionData</code> - a pointer to the partition• <code>barrierType</code> - unique ident for verification <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
<p>PARALLEL_SendRemoteMessages</p> <p>Sends one or more messages to a remote partition. For kernel use only.</p>	<p><code>PARALLEL_SendRemoteMessages (Message* msgList, PartitionData* partitionData, int partitionId)</code></p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>msgList</code> - a linked list of Messages• <code>partitionData</code> - a pointer to the partition• <code>partitionId</code> - the partition's ID <p>Returns:</p> <ul style="list-style-type: none">• - None
<p>PARALLEL_DeliverRemoteMessages</p> <p>Delivers cached messages to all remote partitions. For kernel use only.</p>	<p><code>PARALLEL_DeliverRemoteMessages (PartitionData* partitionData)</code></p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>partitionData</code> - a pointer to the partition

	<div>Returns:</div> <div><ul style="list-style-type: none">- None</div>
<div>PARALLEL_SendRemoteMessagesOob</div> <div>Sends one or more messages to a remote partition. These messages are oob messages and will be processed immediately. For kernel use only.</div>	<div>void PARALLEL_SendRemoteMessagesOob (Message* msgList, PartitionData* partitionData, int partitionId, bool isResponse)</div> <div>Parameters:</div> <div><ul style="list-style-type: none">msgList - a linked list of MessagespartitionData - a pointer to the partitionpartitionId - the partition's IDisResponse - if it's a response to an OOB message</div> <div>Returns:</div> <div><ul style="list-style-type: none">void - None</div>
<div>PARALLEL_SendMessageToAllPartitions</div> <div>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.</div>	<div>void PARALLEL_SendMessageToAllPartitions (Message* msg, PartitionData* partitionData, bool freeMsg)</div> <div>Parameters:</div> <div><ul style="list-style-type: none">msg - the message(s) to sendpartitionData - the sending partitionfreeMsg - whether or not to free the original</div> <div>Returns:</div> <div><ul style="list-style-type: none">void - None</div>
<div>PARALLEL_SendRemoteLinkMessage</div> <div>Sends one LINK message to a remote partition.</div>	<div>PARALLEL_SendRemoteLinkMessage (Node* node, Message* msg, LinkData* link, clocktype txDelay)</div> <div>Parameters:</div> <div><ul style="list-style-type: none">node - the sending nodemsg - the message to be sentlink - info about the linktxDelay - the transmission delay, not including propagation</div> <div>Returns:</div> <div><ul style="list-style-type: none">- None</div>
<div>PARALLEL_UpdateSafeTime</div>	<div>void PARALLEL_UpdateSafeTime (PartitionData* partitionData)</div>

<p>A generic function for calculating the window of safe events For kernel use only.</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
<p>PARALLEL_ReturnEarliestGlobalEventTime</p> <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"><code>partitionData</code> - 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
<p>PARALLEL_Exit</p> <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
<p>PARALLEL_SetProtocolIsNotEOTCapable</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>	<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
<p>PARALLEL_EnableDynamicMobility</p> <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
<p>PARALLEL_SetGreedy</p> <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">void - None
<p>PARALLEL_IsGreedy</p> <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">bool - true if greedy mode is enabled.
<p>PARALLEL_PreFlight</p> <p>Initializes parallel operation.</p>	<p>void PARALLEL_PreFlight (PartitionData* partitionData)</p> <p>Parameters:</p> <ul style="list-style-type: none">partitionData - the partition to initialize. <p>Returns:</p> <ul style="list-style-type: none">void - None
<p>PARALLEL_ScheduleMessagesOnPartition</p> <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.gotOobMessage - returns true if Oob response is receivedisMT - is this called from a worker thread <p>Returns:</p> <ul style="list-style-type: none">void - None
<p>PARALLEL_EndSimulation</p> <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
<p>PARALLEL_BuildStatFile</p> <p>Builds the final stat file when running in parallel node. Should only be</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

called once from partition 0.	<ul style="list-style-type: none">• statFileName - name of stat file• experimentPrefix - experiment prefix Returns: <ul style="list-style-type: none">• void - None
PARALLEL_NumberOfSynchronizations Return the number of synchronizations performed per partition	void PARALLEL_NumberOfSynchronizations () Parameters: Returns: <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) Parameters: <ul style="list-style-type: none">• partitionData - a pointer to the partition Returns: <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 © 2001-2013 [SCALABLE Network Technologies, Inc.](#) All rights reserved.



QualNet 7.1 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
ENUMERATION	PartitionGlobalDataIndex This enumeration contains indexes into the PartitionGlobalData array used for module data.
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
clocktye	getGlobalTime () Returns the simulation time at a global level For the current time of a node, use Node::getNodeTime PartitionData::getGlobalTime should only be used for timing at the partition or global level.
void	PARTITION_GetTerrainPtr (PartitionData* partitionData) Inline function used to get terrainData pointer.
void	PARTITION_CreateEmptyPartition (int partitionId, int numPartitions) Function used to allocate and perform inititlaization of of an empty partition data structure.
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) Function used to initialize a partition.
void	PARTITION_InitializeNodes (PartitionData* partitionData) Function used to allocate and initialize the nodes on a partition.
void	PARTITION_Finalize (PartitionData* partitionData) Finalizes the nodes on the partition.
void	PARTITION_ProcessPartition (PartitionData* partitionData) Creates and initializes the nodes, then processes events on this partition.
void	PARTITION_ProcessSendMT (PartitionData* partitionData) 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.
BOOL	PARTITION_ReturnNodePointer (PartitionData* partitionData, Node** node, NodeId nodeId, BOOL remoteOK)

	<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>PARTITION_NodeExists(PartitionData* partitionData, NodeId nodeId)</p> <p>Determines whether the node ID exists in the scenario. Must follow node creation.</p>
void	<p>PARTITION_PrintRunTimeStats(PartitionData* partitionData)</p> <p>If dynamic statistics reporting is enabled, generates statistics for enabled layers.</p>
void	<p>PARTITION_SchedulePartitionEvent(PartitionData* partitionData, Message* msg, clocktype eventTime, bool scheduleBeforeNodes)</p> <p>Schedules a generic partition-level event.</p>
void	<p>PARTITION_HandlePartitionEvent(PartitionData* partitionData, Message* msg)</p> <p>An empty function for protocols to use that need to schedule and handle partition-level events.</p>
void	<p>PARTITION_ClientStateSet(PartitionData* partitionData, const char* stateName, void* clientState)</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>
void*	<p>PARTITION_ClientStateFind(PartitionData* partitionData, const char* stateName)</p> <p>Looks up the requested client-state by name. Returns NULL if the state isn't present.</p>
CommunicatorId	<p>PARTITION_COMMUNICATION_RegisterCommunicator(PartitionData* partitionData, const char* name, PartitionCommunicationHandler handler)</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>
CommunicatorId	<p>PARTITION_COMMUNICATION_FindCommunicator(PartitionData* partitionData, std name)</p> <p>Locate an already registered commincator.</p>
void	<p>PARTITION_COMMUNICATION_SendToPartition(PartitionData* partitionData, int partitionId, Message* msg, clocktype delay)</p> <p>Transmit a message to a partition.</p>
void	<p>PARTITION_COMMUNICATION_SendToAllPartitions(PartitionData* partitionData, Message* msg, clocktype delay)</p>

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

Print standard QualNet progress log

Constant / Data Structure Detail

Constant	<div>NUM_SIM_TIME_STATUS_PRINTS 100</div> <div>The number of percentage complete statements to print</div>
Constant	<div>COMMUNICATION_ID_INVALID 0</div> <div>A default uninitialized communication ID.</div>
Constant	<div>COMMUNICATION_DELAY_REAL_TIME -1</div> <div>A value to indicate real time interpartition communication</div>
Enumeration	<div>PartitionGlobalDataIndex</div> <div>This enumeration contains indexes into the PartitionGlobalData array used for module data.</div>
Structure	<div>SubnetMemberData</div> <div>Data structure containing interfaceIndex and Node* for a node in a single subnet</div>
Structure	<div>PartitionSubnetMemberList</div> <div>Data structure containing member data info for all nodes in a single subnet</div>
Structure	<div>PartitionSubnetData</div> <div>Data structure containing subnet member data for all subnets</div>
Structure	<div>PartitionData</div> <div>Contains global information for this partition.</div>
Structure	<div>SimulationProperties</div>

Global properties of the simulation for all partitions.

Function / Macro Detail

Function / Macro	Format
<p>getGlobalTime</p> <p>Returns the simulation time at a global level For the current time of a node, use Node::getNodeTime PartitionData::getGlobalTime should only be used for timing at the partition or global level.</p>	<p>clocktype getGlobalTime ()</p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none">clocktype - The current global simulation time
<p>PARTITION_GetTerrainPtr</p> <p>Inline function used to get terrainData pointer.</p>	<p>void PARTITION_GetTerrainPtr (PartitionData* partitionData)</p> <p>Parameters:</p> <ul style="list-style-type: none">partitionData - pointer to partitionData <p>Returns:</p> <ul style="list-style-type: none">void - None
<p>PARTITION_CreateEmptyPartition</p> <p>Function used to allocate and perform initilaization of of an empty partition data structure.</p>	<p>void PARTITION_CreateEmptyPartition (int partitionId, int numPartitions)</p> <p>Parameters:</p> <ul style="list-style-type: none">partitionId - the partition ID, used for parallelnumPartitions - for parallel <p>Returns:</p> <ul style="list-style-type: none">void - None
<p>PARTITION_InitializePartition</p> <p>Function used to initialize a partition.</p>	<p>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)</p> <p>Parameters:</p> <ul style="list-style-type: none">partitionData - an empty partition data structureterrainData - dimensions, terrain database, etc.maxSimClock - length of the scenariostartRealTime - for synchronizing with the realtime

	<ul style="list-style-type: none">• <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 placemt• <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
<p>PARTITION_InitializeNodes</p> <p>Function used to allocate and initialize the nodes on a partition.</p>	<p>void PARTITION_InitializeNodes (PartitionData* partitionData)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>partitionData</code> - an pre-initialized partition data structure <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
<p>PARTITION_Finalize</p> <p>Finalizes the nodes on the partition.</p>	<p>void PARTITION_Finalize (PartitionData* partitionData)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>partitionData</code> - an pre-initialized partition data structure <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
<p>PARTITION_ProcessPartition</p> <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">• <code>partitionData</code> - an pre-initialized partition data structure <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
<p>PARTITION_ProcessSendMT</p>	<p>void PARTITION_ProcessSendMT (PartitionData* partitionData)</p> <p>Parameters:</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>	<ul style="list-style-type: none">• <code>partitionData</code> - an pre-initialized partition data structure <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
<p>PARTITION_ReturnNodePointer</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>	<p>BOOL PARTITION_ReturnNodePointer (PartitionData* partitionData, Node** node, NodeId nodeId, BOOL remoteOK)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>partitionData</code> - an pre-initialized partition data structure• <code>node</code> - for returning the node pointer• <code>nodeId</code> - the node's ID• <code>remoteOK</code> - is it ok to return a pointer to proxy node? <p>Returns:</p> <ul style="list-style-type: none">• <code>BOOL</code> - returns TRUE if the node was succesfully found
<p>PARTITION_NodeExists</p> <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">• <code>partitionData</code> - an pre-initialized partition data structure• <code>nodeId</code> - the node's ID <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
<p>PARTITION_PrintRunTimeStats</p> <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">• <code>partitionData</code> - an pre-initialized partition data structure <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
<p>PARTITION_SchedulePartitionEvent</p> <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">• <code>partitionData</code> - an pre-initialized partition data structure• <code>msg</code> - an event

	<ul style="list-style-type: none">• <code>eventTime</code> - the time the event should occur• <code>scheduleBeforeNodes</code> - process event before or after node events Returns: <ul style="list-style-type: none">• <code>void</code> - None
PARTITION_HandlePartitionEvent An empty function for protocols to use that need to schedule and handle partition-level events.	void PARTITION_HandlePartitionEvent (PartitionData* partitionData, Message* msg) Parameters: <ul style="list-style-type: none">• <code>partitionData</code> - an pre-initialized partition data structure• <code>msg</code> - an event Returns: <ul style="list-style-type: none">• <code>void</code> - None
PARTITION_ClientStateSet 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 PARTITION_ClientStateSet (PartitionData* partitionData, const char* stateName, void* clientState) Parameters: <ul style="list-style-type: none">• <code>partitionData</code> - an pre-initialized partition data structure• <code>stateName</code> - Name used to locate this client state• <code>clientState</code> - Pointer to whatever data-structure the Returns: <ul style="list-style-type: none">• <code>void</code> - None
PARTITION_ClientStateFind Looks up the requested client-state by name. Returns NULL if the state isn't present.	void* PARTITION_ClientStateFind (PartitionData* partitionData, const char* stateName) Parameters: <ul style="list-style-type: none">• <code>partitionData</code> - an pre-initialized partition data structure• <code>stateName</code> - Name used to locate this client state Returns: <ul style="list-style-type: none">• <code>void*</code> - returns the client state
PARTITION_COMMUNICATION_RegisterCommunicator Allocates a message id and registers the handler that will be invoked to receive callbacks when messages are with the id are sent.	CommunicatorId PARTITION_COMMUNICATION_RegisterCommunicator (PartitionData* partitionData, const char* name, PartitionCommunicationHandler handler) Parameters: <ul style="list-style-type: none">• <code>partitionData</code> - an pre-initialized partition data structure• <code>name</code> - Your name for this type of message.

	<ul style="list-style-type: none">• 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
<p>PARTITION_COMMUNICATION_SendToAllPartitions</p> <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
<p>IO_Return_Qualnet_Directory</p>	<p>std IO_Return_Qualnet_Directory ()</p> <p>Parameters:</p>

<p>This will return in a string the current directory qualnet is executing from</p>	<p>Returns:</p> <ul style="list-style-type: none"><code>std</code> - string containing current qualnet directory
<p>IO_SourceFileExists</p> <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
<p>IO_CheckSourceLibrary</p> <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"><code>std</code> - string containing list of libraries
<p>IO_ReturnSourceAndCompiledLibraries</p> <p>This will return in a string a list of libraries currently compiled into product as well as those which have source code available.</p>	<p>std IO_ReturnSourceAndCompiledLibraries ()</p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"><code>std</code> - string containing list of libraries
<p>IO_ReturnExpirationDateFromLicenseFeature</p> <p>This will return in a string a list of libraries currently compiled into product as well as those which have source code available.</p>	<p>std IO_ReturnExpirationDateFromLicenseFeature ()</p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"><code>std</code> - string containing expiration date for this feature
<p>IO_ReturnExpirationDateFromNumericalDate</p> <p>This will return in a string the expiration date of the library</p>	<p>std IO_ReturnExpirationDateFromNumericalDate ()</p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"><code>std</code> - string containing expiration date for this feature
<p>IO_ReturnExpirationDateFromNumericalDate</p> <p>This will return in a string the expiration date of the library</p>	<p>std IO_ReturnExpirationDateFromNumericalDate ()</p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"><code>std</code> - string containing expiration date for this feature
<p>IO_ParseFlexLMDate</p>	<p>UInt64 containing the date IO_ParseFlexLMDate ()</p>

Parse a FlexLM date in a platform safe way	<div>Parameters:</div> <div>Returns:</div> <ul style="list-style-type: none">• UInt64 containing the date - None
<div>IO_ReturnStatusMessageFromLibraryInfo</div> <div>This will return in a string the status message for the library used with the -print_libraries option</div>	<div>std IO_ReturnStatusMessageFromLibraryInfo ()</div> <div>Parameters:</div> <div>Returns:</div> <ul style="list-style-type: none">• std - string containing status message for library
<div>IO_ReturnStatusMessageFromLibraryInfo</div> <div>This will return in a string the library name from its index :: because flexlm won't allow std structs in main.cpp :: but main.cpp is the only place flex structs are allowed</div>	<div>std IO_ReturnStatusMessageFromLibraryInfo ()</div> <div>Parameters:</div> <div>Returns:</div> <ul style="list-style-type: none">• std - string containing library name
<div>PARTITION_ShowProgress</div> <div>Print standard QualNet progress log</div>	<div>PARTITION_ShowProgress ()</div> <div>Parameters:</div> <div>Returns:</div> <ul style="list-style-type: 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 © 2001-2013 [SCALABLE Network Technologies, Inc.](#) All rights reserved.



QualNet 7.1 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 PHY_DEFAULT_NOISE_FACTOR Default noise factor in physical medium
CONSTANT	PHY_DEFAULT_TEMPERATURE 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

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

PHYSICAL LAYER

void	<div><div>PHY_ProcessEvent(Node* node, Message* msg)</div><div>Models the behaviour of the Phy Layer on receiving the message encapsulated in msgHdr</div></div>
PhyStatusType	<div><div>PHY_GetStatus(Node * node, int phyNum)</div><div>Retrieves the Phy's current status</div></div>
void	<div><div>PHY_SetTransmitPower(Node * node, int phyIndex, double newTxPower_mW)</div><div>Sets the Radio's transmit power in mW</div></div>
void	<div><div>PHY_SetRxSNRThreshold(Node * node, int phyIndex, double snr)</div><div>Sets the Radio's Rx SNR Threshold</div></div>
void	<div><div>PHY_SetDataRate(Node * node, int phyIndex, Int64 dataRate)</div><div>Sets the Radio's Data Rate for both Tx and Rx</div></div>
void	<div><div>PHY_SetTxDataRate(Node * node, int phyIndex, Int64 dataRate)</div><div>For radios that support different Tx and Rx data rates, this will set the Rx Data Rate. For others, it will call PHY_SetDataRate.</div></div>
void	<div><div>PHY_SetRxDataRate(Node * node, int phyIndex, Int64 dataRate)</div><div>For radios that support different Tx and Rx data rates, this will set the Rx Data Rate. For others, it will call PHY_SetDataRate.</div></div>
void	<div><div>PHY_GetTransmitPower(Node * node, int phyIndex, double* txPower_mW)</div><div>Gets the Radio's transmit power in mW.</div></div>
clocktype	<div><div>PHY_GetTransmissionDelay(Node * node, int phyIndex, int size)</div><div>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</div></div>
clocktype	<div><div>PHY_GetTransmissionDuration(Node * node, int phyIndex, int dataRateIndex, int size)</div><div>Get transmission duration of a structured signal fragment.</div></div>
PhyModel	<div><div>PHY_GetFrameModel(Node * node, int phyNum)</div><div>Get Physical Model</div></div>

PhyModel	PHY_GetAntennaModelType (Node * node, int phyNum) Get Antenna Model type
void	PHY_StartTransmittingSignal (Node * node, int phyNum, Message* msg, BOOL useMacLayerSpecifiedDelay, clocktype delayUntilAirborne) Starts transmitting a packet.
void	PHY_StartTransmittingSignal (Node * node, int phyNum, Message* msg, clocktype duration, BOOL useMacLayerSpecifiedDelay, clocktype delayUntilAirborne) Starts transmitting a packet. Function is being overloaded
void	PHY_StartTransmittingSignal (Node * node, int phyNum, Message* msg, int bitSize, BOOL useMacLayerSpecifiedDelay, clocktype delayUntilAirborne) Starts transmitting a packet.
void	PHY_SignalArrivalFromChannel (Node * node, int phyIndex, int channelIndex, PropRxInfo* propRxInfo) Called when a new signal arrives
void	PHY_SignalEndFromChannel (Node * node, int phyIndex, int channelIndex, PropRxInfo* propRxInfo) Called when the current signal ends
Int64	PHY_GetTxDataRate (Node * node, int phyIndex) Get transmission data rate
Int64	PHY_GetRxDataRate (Node * node, int phyIndex) Get reception data rate
int	PHY_GetTxDataRateType (Node * node, int phyIndex) Get transmission data rate type
int	PHY_GetRxDataRateType (Node * node, int phyIndex) Get reception data rate type
void	PHY_SetTxDataRateType (Node * node, int phyIndex, int dataRateType)

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

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

	<p>Terminate the current signal reception</p>
double	<p>PHY_PropagationRange(Node* txnode, Node* node, int txInterfaceIndex, int interfaceIndex, int channelIndex, BOOL printAll)</p>
	<p>Calculates an estimated radio range for the PHY. Supports only TWO-RAY and FREE-SPACE.</p>
void	<p>ENERGY_Init(Node* node, const int phyIndex, NodeInput* nodeInput)</p>
	<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>
void	<p>ENERGY_PrintStats(Node* node, const int phyIndex)</p>
	<p>To print the statistic of Energy Model.</p>
void	<p>Phy_ReportStatusToEnergyModel(Node* node, const int phyIndex, PhyStatusType prevStatus, PhyStatusType newStatus)</p>
	<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 sates 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>
void	<p>Generic_UpdateCurrentLoad(Node* node, const int phyIndex)</p>
	<p>To update the current load of generic energy model.</p>
void	<p>PHY_NotificationOfPacketDrop(Node* node, int phyIndex, int channelIndex, const Message* msg, const string& dropType, double rxPower_mW, double interferencePower_mW, double passloss_dB)</p>
	<p>To Notify the StatsDB module and other modules of the packet dropping event.</p>
void	<p>PHY_NotificationOfSignalReceived(Node* node, int phyIndex, int channelIndex, const Message* msg, double rxPower_mW, double interferencePower_mW, double passloss_dB, int controlSize)</p>
	<p>To Notify the StatsDB module and other modules of the signal received event .</p>
void	<p>PHY_GetSteeringAngle(Node* node, int phyIndex)</p>
	<p>Gets the current steering angle for a directional antenna from PHY models that support this.</p>
double	<p>PHY_GetBandwidth(Node* node, int phyIndex)</p>
	<p>To get the bandwidth for the given PHY model.</p>
double	<p>PHY_GetFrequency(Node* node, int channelIndex)</p>

	<p>To get the frequency for the given signal.</p>
PhyModel	<p>PHY_GetPhyModel(Node* node, int phyIndex)</p>
	<p>To get the PhyModel for the node.</p>
std	<p>PHY_GetPhyModel(Node* node, int phyIndex)</p>
	<p>To get the name of a phy model</p>
BOOL	<p>PHY_isSignalFeatureMatchReceiverPhyModel(Node* node, int phyIndex)</p>
	<p>To check if the signal feature matches the receiver's phyModel.</p>
double	<p>PHY_ComputeInbandPower(double signalPower_mW, double signalFrequency, double signalBandwidth, double rxFrequency, double rxBandwidth)</p>
	<p>To estimate the inband signal power for given signal and receiver parameters.</p>
void	<p>PHY_InterferenceArrivalFromChannel(Node * node, int phyIndex, int channelIndex, PropRxInfo* propRxInfo, double sigPower_mW)</p>
	<p>Called when a interference signal arrives</p>
void	<p>PHY_InterferenceEndFromChannel(Node * node, int phyIndex, int channelIndex, PropRxInfo* propRxInfo, double sigPower_mW)</p>
	<p>Called when a interference signal ends</p>
std	<p>PHY_GetChannelName(Node* node, int channelIndex)</p>
	<p>To get the name of the channel.</p>
Int32	<p>PHY_GetChannelIndexForChannelName(Node* node, std channelName)</p>
	<p>To get the channel index.</p>
BOOL	<p>PHY_ChannelNameExists(Node* node)</p>
	<p>To check whether channelName exist or not.</p>

Constant / Data Structure Detail

Constant	<div>PHY_DEFAULT_NOISE_FACTORPHY_DEFAULT_NOISE_FACTOR 10.0</div> <div>Default noise factor in physical medium</div>
Constant	<div>PHY_DEFAULT_TEMPERATUREPHY_DEFAULT_TEMPERATURE 290.0</div> <div>Default temperature of physical medium.</div>
Constant	<div>PHY_DEFAULT_MIN_PCOM_VALUE 0.0</div> <div>Default minimum pcom value threshold</div>
Constant	<div>PHY_DEFAULT_SYNC_COLLISION_WINDOW 1ms</div> <div>Default minimum pcom value threshold</div>
Enumeration	<div>PhyModel</div> <div>Different phy types supported.</div>
Enumeration	<div>PhyRxModel</div> <div>Different types of packet reception model</div>
Structure	<div>PhyBerEntry</div> <div>SNR/BER curve entry</div>
Structure	<div>PhyBerTable</div> <div>Bit Error Rate table.</div>
Structure	<div>PhyPerEntry</div> <div>SNR/PER curve entry</div>
Structure	<div>PhyPerTable</div> <div>Packet Error Rate table.</div>
Structure	<div>PhySerEntry</div>

	SNR/PER curve entry
Structure	PhySerTable
	Symbol Error Rate table.
Structure	PhySignalMeasurement
	Measurement of the signal of received pkt
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	void PHY_GlobalBerInit (const NodeInput* nodeInput)
Pre-load all the BER files.	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 pointerphyIndex - interface Index Returns: <ul style="list-style-type: none">int - None
PHY_SetSnrBerTableIndex Set index of BER table to be used by PHY.	int PHY_SetSnrBerTableIndex (Node* node, int phyIndex) Parameters: <ul style="list-style-type: none">node - Node pointerphyIndex - interface Index Returns: <ul style="list-style-type: none">int - None
PHY_GlobalPerInit Pre-load all the PER files.	void PHY_GlobalPerInit (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_GetSnrPerTableByName Get a pointer to a specific PER table.	void PHY_GetSnrPerTableByName (char* tableName) Parameters: <ul style="list-style-type: none">tableName - name of the PER file Returns:

	<div>void - None</div>
<div>PHY_GlobalSerInit</div> <div>Pre-load all the SER files.</div>	<div>void PHY_GlobalSerInit (const NodeInput* nodeInput)</div> <div>Parameters:</div> <div><ul style="list-style-type: none">nodeInput - structure containing contents of input file</div> <div>Returns:</div> <div><ul style="list-style-type: none">void - None</div>
<div>PHY_GetSnrSerTableByName</div> <div>Get a pointer to a specific SER table.</div>	<div>void PHY_GetSnrSerTableByName (char* tableName)</div> <div>Parameters:</div> <div><ul style="list-style-type: none">tableName - name of the SER file</div> <div>Returns:</div> <div><ul style="list-style-type: none">void - None</div>
<div>PHY_Init</div> <div>Initialize physical layer</div>	<div>void PHY_Init (Node* node, const NodeInput* nodeInput)</div> <div>Parameters:</div> <div><ul style="list-style-type: none">node - node being initializednodeInput - structure containing contents of input file</div> <div>Returns:</div> <div><ul style="list-style-type: none">void - None</div>
<div>PHY_CreateAPhyForMac</div> <div>Initialization function for the phy layer</div>	<div>void PHY_CreateAPhyForMac (Node* node, const NodeInput* nodeInput, int interfaceIndex, int networkAddress, PhyModel phyModel, int* phyNumber)</div> <div>Parameters:</div> <div><ul style="list-style-type: none">node - node being initializednodeInput - structure containing contents of input fileinterfaceIndex - interface being initialized.networkAddress - address of the interface.phyModel - Which physical model is used.phyNumber - returned value to be used as phyIndex</div> <div>Returns:</div> <div><ul style="list-style-type: none">void - None</div>

PHY_Finalize Called at the end of simulation to collect the results of the simulation of the Phy Layer.	void PHY_Finalize (Node * node) Parameters: <ul style="list-style-type: none">node - node for which results are to be collected Returns: <ul style="list-style-type: none">void - None
PHY_ProcessEvent Models the behaviour of the Phy Layer on receiving the message encapsulated in msgHdr	void PHY_ProcessEvent (Node* node, Message* msg) Parameters: <ul style="list-style-type: none">node - node which received the messagemsg - message received by the layer Returns: <ul style="list-style-type: none">void - None
PHY_GetStatus Retrieves the Phy's current status	PhyStatusType PHY_GetStatus (Node * node, int phyNum) Parameters: <ul style="list-style-type: none">node - node for which stats are to be collectedphyNum - interface for which stats are to be collected Returns: <ul style="list-style-type: none">PhyStatusType - status of interface.
PHY_SetTransmitPower Sets the Radio's transmit power in mW	void PHY_SetTransmitPower (Node * node, int phyIndex, double newTxPower_mW) Parameters: <ul style="list-style-type: none">node - node for which transmit power is to be setphyIndex - interface for which transmit power is to be setnewTxPower_mW - transmit power(mW) Returns: <ul style="list-style-type: none">void - None
PHY_SetRxSNRThreshold Sets the Radio's Rx SNR Threshold	void PHY_SetRxSNRThreshold (Node * node, int phyIndex, double snr) Parameters: <ul style="list-style-type: none">node - node for which transmit power is to be setphyIndex - interface for which transmit power is to be set

	<ul style="list-style-type: none"><code>snr</code> - threshold value to be set Returns: <ul style="list-style-type: none"><code>void</code> - None
PHY_SetDataRate Sets the Radio's Data Rate for both Tx and Rx	void PHY_SetDataRate (Node * node, int phyIndex, Int64 dataRate) Parameters: <ul style="list-style-type: none"><code>node</code> - node for which transmit power is to be set<code>phyIndex</code> - interface for which transmit power is to be set<code>dataRate</code> - dataRate value to be set Returns: <ul style="list-style-type: none"><code>void</code> - None
PHY_SetTxDataRate 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_SetTxDataRate (Node * node, int phyIndex, Int64 dataRate) Parameters: <ul style="list-style-type: none"><code>node</code> - node for which transmit power is to be set<code>phyIndex</code> - interface for which transmit power is to be set<code>dataRate</code> - dataRate value to be set Returns: <ul style="list-style-type: none"><code>void</code> - None
PHY_SetRxDataRate 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, Int64 dataRate) Parameters: <ul style="list-style-type: none"><code>node</code> - node for which transmit power is to be set<code>phyIndex</code> - interface for which transmit power is to be set<code>dataRate</code> - dataRate value to be set Returns: <ul style="list-style-type: none"><code>void</code> - None
PHY_GetTransmitPower Gets the Radio's transmit power in mW.	void PHY_GetTransmitPower (Node * node, int phyIndex, double* txPower_mW) Parameters: <ul style="list-style-type: none"><code>node</code> - Node that is

	<ul style="list-style-type: none">• <code>phyIndex</code> - interface index.• <code>txPower_mW</code> - transmit power(mW) <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
<p>PHY_GetTransmissionDelay</p> <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 PHY_GetTransmissionDelay (Node * node, int phyIndex, 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>size</code> - size of the frame in bytes <p>Returns:</p> <ul style="list-style-type: none">• <code>clocktype</code> - transmission delay.
<p>PHY_GetTransmissionDuration</p> <p>Get transmission duration of a structured signal fragment.</p>	<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
<p>PHY_GetFrameModel</p> <p>Get Physical Model</p>	<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
<p>PHY_GetAntennaModelType</p> <p>Get Antenna Model type</p>	<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

	<ul style="list-style-type: none">• <code>phyNum</code> - interface index <p>Returns:</p> <ul style="list-style-type: none">• <code>PhyModel</code> - Physical Model
<p>PHY_StartTransmittingSignal</p> <p>Starts transmitting a packet.</p>	<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• <code>delayUntilAirborne</code> - delay until airborne <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
<p>PHY_StartTransmittingSignal</p> <p>Starts transmitting a packet. Function is being overloaded</p>	<p>void PHY_StartTransmittingSignal (Node * node, int phyNum, Message* msg, clocktype duration, 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>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
<p>PHY_StartTransmittingSignal</p> <p>Starts transmitting a packet.</p>	<p>void PHY_StartTransmittingSignal (Node * node, int phyNum, Message* msg, int bitSize, 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

	<ul style="list-style-type: none">• <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
<p>PHY_SignalArrivalFromChannel</p> <p>Called when a new signal arrives</p>	<p>void PHY_SignalArrivalFromChannel (Node * node, int phyIndex, int channelIndex, PropRxInfo* propRxInfo)</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 <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
<p>PHY_SignalEndFromChannel</p> <p>Called when the current signal ends</p>	<p>void PHY_SignalEndFromChannel (Node * node, int phyIndex, int channelIndex, PropRxInfo* propRxInfo)</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 <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
<p>PHY_GetTxDataRate</p> <p>Get transmission data rate</p>	<p>Int64 PHY_GetTxDataRate (Node * node, int phyIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>node</code> - node pointer to node• <code>phyIndex</code> - interface index <p>Returns:</p> <ul style="list-style-type: none">• <code>Int64</code> - None

PHY_GetRxDataRate Get reception data rate	Int64 PHY_GetRxDataRate (Node * node, int phyIndex) Parameters: <ul style="list-style-type: none">node - node pointer to nodephyIndex - interface index Returns: <ul style="list-style-type: none">Int64 - None
PHY_GetTxDataRateType Get transmission data rate type	int PHY_GetTxDataRateType (Node * node, int phyIndex) Parameters: <ul style="list-style-type: none">node - node pointer to nodephyIndex - interface index Returns: <ul style="list-style-type: none">int - None
PHY_GetRxDataRateType Get reception data rate type	int PHY_GetRxDataRateType (Node * node, int phyIndex) Parameters: <ul style="list-style-type: none">node - node pointer to nodephyIndex - interface index Returns: <ul style="list-style-type: none">int - None
PHY_SetTxDataRateType Set transmission data rate type	void PHY_SetTxDataRateType (Node * node, int phyIndex, int dataRateType) Parameters: <ul style="list-style-type: none">node - node pointer to nodephyIndex - interface indexdataRateType - rate of data Returns: <ul style="list-style-type: none">void - None
PHY_GetLowestTxDataRateType	void PHY_GetLowestTxDataRateType (Node* node, int phyIndex, int* dataRateType) Parameters:

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

	<ul style="list-style-type: none">dataRateType - rate of data <p>Returns:</p> <ul style="list-style-type: none">void - None
<p>PHY_SetHighestTxDataRateTypeForBC</p> <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 nodephyIndex - interface index <p>Returns:</p> <ul style="list-style-type: none">void - None
<p>PHY_ComputeSINR</p> <p>Compute SINR</p>	<p>double PHY_ComputeSINR (PhyData * phyData, double * signalPower_mW, double* interferencePower_mW, int bandwidth)</p> <p>Parameters:</p> <ul style="list-style-type: none">phyData - PHY layer datasignalPower_mW - Signal powerinterferencePower_mW - Interference powerbandwidth - Bandwidth <p>Returns:</p> <ul style="list-style-type: none">double - Signal to Interference and Noise Ratio
<p>PHY_SignalInterference</p> <p>Compute Power from the desired signal and interference</p>	<p>void PHY_SignalInterference (Node* node, int phyIndex, int channelIndex, Message * msg, double * signalPower_mW, double* interferencePower_mW)</p> <p>Parameters:</p> <ul style="list-style-type: none">node - Node that is beingphyIndex - interface numberchannelIndex - channel indexmsg - message including desired signalsignalPower_mW - power from the desired signalinterferencePower_mW - power from interfering signals <p>Returns:</p>

	<ul style="list-style-type: none">void - None
PHY_BER Get BER	<p>double PHY_BER (PhyData * phyData, int berTableIndex, double sinr)</p> <p>Parameters:</p> <ul style="list-style-type: none">phyData - PHY layer databerTableIndex - index for BER tablessinr - Signal to Interference and Noise Ratio <p>Returns:</p> <ul style="list-style-type: none">double - Bit Error Rate
PHY_SER Get SER	<p>double PHY_SER (PhyData * phyData, int perTableIndex, double sinr)</p> <p>Parameters:</p> <ul style="list-style-type: none">phyData - PHY layer dataperTableIndex - index for SER tablessinr - 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 beingphyIndex - interface numberchannelIndex - channel index <p>Returns:</p> <ul style="list-style-type: none">void - None
PHY_CanListenToChannel Check if it can listen to the channel	<p>BOOL PHY_CanListenToChannel (Node* node, int phyIndex, int channelIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none">node - Node that is beingphyIndex - interface numberchannelIndex - channel index <p>Returns:</p>

	<ul style="list-style-type: none">• <code>BOOL</code> - None
<p>PHY_IsListeningToChannel</p> <p>Check if it is listening to the channel</p>	<p>BOOL PHY_IsListeningToChannel (Node* node, int phyIndex, int channelIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>node</code> - Node that is being• <code>phyIndex</code> - interface number• <code>channelIndex</code> - channel index <p>Returns:</p> <ul style="list-style-type: none">• <code>BOOL</code> - None
<p>PHY_SetTransmissionChannel</p> <p>Set the channel index used for transmission</p>	<p>void PHY_SetTransmissionChannel (Node* node, int phyIndex, int channelIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>node</code> - Node that is being• <code>phyIndex</code> - interface number• <code>channelIndex</code> - channel index <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
<p>PHY_GetTransmissionChannel</p> <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">• <code>node</code> - Node that is being• <code>phyIndex</code> - interface number• <code>channelIndex</code> - channel index <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
<p>PHY_MediumIsIdle</p> <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">• <code>node</code> - Node that is being• <code>phyNum</code> - interface number <p>Returns:</p>

	<ul style="list-style-type: none">• <code>BOOL</code> - None
PHY_MediumIsIdleInDirection Check if the medium is idle if sensed directionally	<p>BOOL PHY_MediumIsIdleInDirection (Node* node, int phyNum, double azimuth)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>node</code> - Node that is being• <code>phyNum</code> - interface number• <code>azimuth</code> - azimuth (in degrees) <p>Returns:</p> <ul style="list-style-type: none">• <code>BOOL</code> - None
PHY_SetSensingDirection Set the sensing direction	<p>void PHY_SetSensingDirection (Node* node, int phyNum, double azimuth)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>node</code> - Node that is being• <code>phyNum</code> - interface number• <code>azimuth</code> - azimuth (in degrees) <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
PHY_StartTransmittingSignalDirectionally Start transmitting a signal directionally	<p>void PHY_StartTransmittingSignalDirectionally (Node* node, int phyNum, Message* msg, BOOL useMacLayerSpecifiedDelay, clocktype delayUntilAirborne, double directionAzimuth)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>node</code> - Node that is• <code>phyNum</code> - interface number• <code>msg</code> - signal to transmit• <code>useMacLayerSpecifiedDelay</code> - use delay specified by MAC• <code>delayUntilAirborne</code> - delay until airborne• <code>directionAzimuth</code> - azimuth to transmit the signal <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
PHY_LockAntennaDirection	<p>void PHY_LockAntennaDirection (Node* node, int phyNum)</p> <p>Parameters:</p>

Lock the direction of antenna	<ul style="list-style-type: none">node - Node that is beingphyNum - interface number Returns: <ul style="list-style-type: none">void - None
PHY_UnlockAntennaDirection Unlock the direction of antenna	void PHY_UnlockAntennaDirection (Node* node, int phyNum) Parameters: <ul style="list-style-type: none">node - Node that is beingphyNum - interface number Returns: <ul style="list-style-type: none">void - None
PHY_GetLastSignalsAngleOfArrival Get the AOA of the last signal	double PHY_GetLastSignalsAngleOfArrival (Node* node, int phyNum) Parameters: <ul style="list-style-type: none">node - Node that is beingphyNum - interface number Returns: <ul style="list-style-type: none">double - AOA
PHY_TerminateCurrentReceive Terminate the current signal reception	void PHY_TerminateCurrentReceive (Node* node, int phyNum, const BOOL terminateOnlyOnReceiveError, BOOL* receiveError, clocktype* endSignalTime) Parameters: <ul style="list-style-type: none">node - Node pointer that thephyNum - interface numberterminateOnlyOnReceiveError - terminate only whenreceiveError - if error happenedendSignalTime - end of signal Returns: <ul style="list-style-type: none">void - None
PHY_PropagationRange	double PHY_PropagationRange (Node* txnode, Node* node, int txInterfaceIndex, int interfaceIndex, int channnelIndex, BOOL printAll) Parameters:

<p>Calculates an estimated radio range for the PHY. Supports only TWO-RAY and FREE-SPACE.</p>	<ul style="list-style-type: none">• <code>txnode</code> - the Tx node of interest• <code>node</code> - the Rx node of interest• <code>txInterfaceIndex</code> - the interface for the TX node• <code>interfaceIndex</code> - the interface for the Rx node• <code>channelIndex</code> - the index of the channel• <code>printAll</code> - if TRUE, prints the range for all data <p>Returns:</p> <ul style="list-style-type: none">• <code>double</code> - the range in meters
<p>ENERGY_Init</p> <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>void ENERGY_Init (Node* node, const int phyIndex, NodeInput* nodeInput)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>node</code> - the node of interest.• <code>phyIndex</code> - the PHY index.• <code>nodeInput</code> - the node input. <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
<p>ENERGY_PrintStats</p> <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">• <code>node</code> - the node of interest.• <code>phyIndex</code> - the PHY index. <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
<p>Phy_ReportStatusToEnergyModel</p> <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 sates 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">• <code>node</code> - the node of interest.• <code>phyIndex</code> - the PHY index.• <code>prevStatus</code> - the the previous status.• <code>newStatus</code> - the the new status.

	<div>Returns:</div> <div><ul style="list-style-type: none"><code>void</code> - None</div>
<div><div>Generic_UpdateCurrentLoad</div><div>To update the current load of generic energy model.</div></div>	<div><div>void Generic_UpdateCurrentLoad (Node* node, const int phyIndex)</div><div>Parameters:</div><div><ul style="list-style-type: none"><code>node</code> - the node of interest.<code>phyIndex</code> - the PHY index.</div><div>Returns:</div><div><ul style="list-style-type: none"><code>void</code> - None</div></div>
<div><div>PHY_NotificationOfPacketDrop</div><div>To Notify the StatsDB module and other modules of the packet dropping event.</div></div>	<div><div>void PHY_NotificationOfPacketDrop (Node* node, int phyIndex, int channelIndex, const Message* msg, const string& dropType, double rxPower_mW, double interferencePower_mW, double passloss_dB)</div><div>Parameters:</div><div><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> - receving power of the signal<code>interferencePower_mW</code> - interference power of the signal<code>passloss_dB</code> - pathloss value of the signal</div><div>Returns:</div><div><ul style="list-style-type: none"><code>void</code> - None</div></div>
<div><div>PHY_NotificationOfSignalReceived</div><div>To Notify the StatsDB module and other modules of the signal received event .</div></div>	<div><div>void PHY_NotificationOfSignalReceived (Node* node, int phyIndex, int channelIndex, const Message* msg, double rxPower_mW, double interferencePower_mW, double passloss_dB, int controlSize)</div><div>Parameters:</div><div><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</div></div>

	<div><div>rxPower_mW - receiving power of the signal</div><div><ul style="list-style-type: none">interferencePower_mW - interference power of the signalpassloss_dB - pathloss value of the signalcontrolSize - size of control header</div><div>Returns:<div><ul style="list-style-type: none">void - None</div></div></div>
<div><div>PHY_GetSteeringAngle</div><div>Gets the current steering angle for a directional antenna from PHY models that support this.</div></div>	<div><div>void PHY_GetSteeringAngle (Node* node, int phyIndex)</div><div>Parameters:<div><ul style="list-style-type: none">node - node being usedphyIndex - physical to be initialized</div></div><div>Returns:<div><ul style="list-style-type: none">void - None</div></div></div>
<div><div>PHY_GetBandwidth</div><div>To get the bandwidth for the given PHY model.</div></div>	<div><div>double PHY_GetBandwidth (Node* node, int phyIndex)</div><div>Parameters:<div><ul style="list-style-type: none">node - The node of interest.phyIndex - The PHY index.</div></div><div>Returns:<div><ul style="list-style-type: none">double - The bandwidth</div></div></div>
<div><div>PHY_GetFrequency</div><div>To get the frequency for the given signal.</div></div>	<div><div>double PHY_GetFrequency (Node* node, int channelIndex)</div><div>Parameters:<div><ul style="list-style-type: none">node - The node of interest.channelIndex - Index of the propagation channel</div></div><div>Returns:<div><ul style="list-style-type: none">double - The frequency</div></div></div>
<div><div>PHY_GetPhyModel</div><div>To get the PhyModel for the node.</div></div>	<div><div>PhyModel PHY_GetPhyModel (Node* node, int phyIndex)</div><div>Parameters:<div><ul style="list-style-type: none">node - The node of interest.phyIndex - The PHY index.</div></div></div>

	<div>Returns:</div> <div><ul style="list-style-type: none">PhyModel - The PhyModel</div>
<div><div>PHY_GetPhyModel</div><div>To get the name of a phy model</div></div>	<div>std PHY_GetPhyModel (Node* node, int phyIndex)</div> <div>Parameters:</div> <div><ul style="list-style-type: none">node - The node of interest.phyIndex - The PHY index.</div> <div>Returns:</div> <div><ul style="list-style-type: none">std - string</div>
<div><div>PHY_isSignalFeatureMatchReceiverPhyModel</div><div>To check if the signal feature matches the receiver's phyModel.</div></div>	<div>BOOL PHY_isSignalFeatureMatchReceiverPhyModel (Node* node, int phyIndex)</div> <div>Parameters:</div> <div><ul style="list-style-type: none">node - The node of interest.phyIndex - The PHY index.</div> <div>Returns:</div> <div><ul style="list-style-type: none">BOOL - if the signal feature matches the receiver's phyModel</div>
<div><div>PHY_ComputeInbandPower</div><div>To estimate the inband signal power for given signal and receiver parameters.</div></div>	<div>double PHY_ComputeInbandPower (double signalPower_mW, double signalFrequency, double signalBandwidth, double rxFrequency, double rxBandwidth)</div> <div>Parameters:</div> <div><ul style="list-style-type: none">signalPower_mW - The signal power in mW.signalFrequency - The signal frequency in Hz.signalBandwidth - The signal bandwidth in HzrxFrequency - The receiver frequency in HzrxBandwidth - The receiver bandwidth in Hz</div> <div>Returns:</div> <div><ul style="list-style-type: none">double - The inband signal power in mW</div>
<div><div>PHY_InterferenceArrivalFromChannel</div><div>Called when a interference signal arrives</div></div>	<div>void PHY_InterferenceArrivalFromChannel (Node * node, int phyIndex, int channelIndex, PropRxInfo* propRxInfo, double sigPower_mW)</div> <div>Parameters:</div> <div><ul style="list-style-type: none">node - node pointer to node</div>

	<ul style="list-style-type: none">• <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
<p>PHY_InterferenceEndFromChannel</p> <p>Called when a interference signal ends</p>	<p><code>void</code> PHY_InterferenceEndFromChannel (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
<p>PHY_GetChannelName</p> <p>To get the name of the channel.</p>	<p><code>std</code> PHY_GetChannelName (Node* node, int channelIndex)</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>std</code> - string
<p>PHY_GetChannelIndexForChannelName</p> <p>To get the channel index.</p>	<p><code>Int32</code> PHY_GetChannelIndexForChannelName (Node* node, std channelName)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>node</code> - The node of interest.• <code>channelName</code> - string <p>Returns:</p> <ul style="list-style-type: none">• <code>Int32</code> - Channel index.

PHY_ChannelNameExists To check whether channelName exist or not.	BOOL PHY_ChannelNameExists (Node* node) Parameters: <ul style="list-style-type: none">node - The node of interest. Returns: <ul style="list-style-type: none">BOOL - TRUE if channelName is valid False if channel name is invalid
--	--



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 © 2001-2013 [SCALABLE Network Technologies, Inc.](#) All rights reserved.



QualNet 7.1 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.

double	<div><div>PROP_PathlossTwoRay(double distance, double wavelength, float txAntennaHeight, float rxAntennaHeight)</div><div>To calculate path loss of a channel.</div></div>
double	<div><div>PROP_PathlossOpar(double distance, double OverlappingDistance, double frequency, ObstructionType obstructiontype)</div><div>Calculates extra path attenuation using opar model.</div></div>
void	<div><div>PROP_CalculatePathloss(Node* node, NodeId txNodeId, NodeId rxNodeId, int channelIndex, double wavelength, float txAntennaHeight, float rxAntennaHeight, PropPathProfile* pathProfile, bool forBinning)</div><div>To calculate path loss of a channel.</div></div>
void	<div><div>PROP_CalculateFading(PropTxInfo* propTxInfo, Node* node2, int channelIndex, clocktype currentTime, float* fading_dB, double* channelReal, double* channelImag)</div><div>To calculate fading between two node.</div></div>
BOOL	<div><div>PROP_CalculateRxPowerAndPropagationDelay(Message* msg, int channelIndex, PropChannel* propChannel, PropTxInfo* propTxInfo, Node* txNode, Node* rxNode, PropPathProfile* pathProfile)</div><div>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.</div></div>
BOOL	<div><div>PROP_CalculateRxPowerAndPropagationDelay(Message* msg, int channelIndex, PropChannel* propChannel, PropTxInfo* propTxInfo, Node* txNode, Node* rxNode, PropPathProfile* pathProfile)</div><div>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.</div></div>
void	<div><div>PROP_MotionObtainfadingStretchingFactor(PropTxInfo* propTxInfo, Node* receiver, int channelIndex)</div><div>Get a stretching factor for fast moving objects.</div></div>
void	<div><div>PROP_UpdatePathProfiles(Node* node)</div><div>UpdatePathProfiles</div></div>
void	<div><div>PROP_ReleaseSignal(Node* node, Message* msg, int phyIndex, int channelIndex, float txPower_dBm, clocktype duration, clocktype delayUntilAirborne)</div><div>Release (transmit) the signal</div></div>
void	<div><div>PROP_SubscribeChannel(Node* node, int phyIndex, int channelIndex)</div></div>

	Start subscribing (listening to) a channel
void	PROP_UnsubscribeChannel (Node* node, int phyIndex, int channelIndex)
	Stop subscription of (listening to) a channel
void	PROP_UnreferenceSignal (Node* node, Message* msg)
	Unreference a signal (internal use)
void	PROP_CalculateInterNodePathLossOnChannel (Node* node, int channelIndex, int* numNodesOnChannel, NodeAddress* nodeIdList, float** pathloss_dB, float** distance)
	Calculate inter-node pathloss, distance values between all the nodes on a given channel
clocktype	PROP_CalculatePropagationDelay (double distance, double propSpeed, PartitionData* partitionData, int channelIndex, int coordinateSystemType, Coordinates* fromPosition, Coordinates* toPosition)
	Calculate the wireless propagation delay for the given distance and propagation speed.
void	PROP_Reset (Node* node, int phyIndex, char* newChannelListenable)
	Reset previous channel remove/add node to propChannel for signal delivery, in propagation_private.
void	PROP_AddNodeToList (Node* node, int channelIndex)
	add node to propChannel nodeList need to make sure that node is not already exists in list before adding.
void	PROP_RemoveNodeFromList (Node* node, int channelIndex)
	remove node from propChannel nodeList need to make sure that all the interface from that node is not listing on that channel before removing.
double	PROP_GetChannelFrequency (Node* node, int channelIndex)
	Get channel frequency from profile for PropChannel.
void	PROP_SetChannelFrequency (Node* node, int channelIndex, double channelFrequency)
	Set channel frequency from profile for PropChannel.
double	PROP_GetChannelWavelength (Node* node, int channelIndex)
	Get channel wavelength from profile for PropChannel.
void	PROP_SetChannelWavelength (Node* node, int channelIndex, double channelWavelength)

	Set channel wavelength from profile for PropChannel.
double	PROP_GetChannelDopplerFrequency (Node* node, int channelIndex)
	Get channel doppler freq from profile for PropChannel.
void	PROP_SetChannelDopplerFrequency (Node* node, int channelIndex, double channelDopplerFrequency)
	Set channel doppler freq from profile for PropChannel.
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.

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	<div>void PROP_GlobalInit (PartitionData* partitionData, NodeInput* nodeInput)</div> <div>Parameters:</div> <ul style="list-style-type: none">partitionData - structure shared among nodesnodeInput - structure containing contents of input file <div>Returns:</div> <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">• <code>partitionData</code> - structure shared among nodes• <code>nodeInput</code> - structure containing contents of input file <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
<p>PROP_Init</p> <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">• <code>node</code> - node being initialized.• <code>channelIndex</code> - channel being initialized.• <code>nodeInput</code> - structure containing contents of input file <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
<p>PROP_ProcessEvent</p> <p>To receive message.</p>	<p>void PROP_ProcessEvent (Node* node, Message* msg)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>node</code> - Node that is• <code>msg</code> - message received by the layer <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
<p>PROP_Finalize</p> <p>To collect various result.</p>	<p>void PROP_Finalize (Node* node)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>node</code> - node for which results are to be collected <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
<p>PROP_PathlossFreeSpace</p> <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">• <code>distance</code> - distance (meters) between two nodes• <code>wavelength</code> - wavelength used for propagation. <p>Returns:</p>

	<ul style="list-style-type: none">double - pathloss in db
<p>PROP_PathlossTwoRay</p> <p>To calculate path loss of a channel.</p>	<p>double PROP_PathlossTwoRay (double distance, double wavelength, float txAntennaHeight, float rxAntennaHeight)</p> <p>Parameters:</p> <ul style="list-style-type: none">distance - distance (meters) between two nodeswavelength - wavelength used for propagation.txAntennaHeight - tranmitting antenna hight.rxAntennaHeight - receiving antenna hight. <p>Returns:</p> <ul style="list-style-type: none">double - pathloss in db
<p>PROP_PathlossOpar</p> <p>Calculates extra path attenuation using opar model.</p>	<p>double PROP_PathlossOpar (double distance, double OverlappingDistance, double frequency, ObstructionType obstructiontype)</p> <p>Parameters:</p> <ul style="list-style-type: none">distance - distance (meters) between two nodesOverlappingDistance - overlapping distancefrequency - frequency used for propagation.obstructiontype - obstruction type <p>Returns:</p> <ul style="list-style-type: none">double - extra path attenuation in db
<p>PROP_CalculatePathloss</p> <p>To calculate path loss of a channel.</p>	<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">node - Node that istxNodeId - including for debuggingrxNodeId - including for debuggingchannelIndex - channel number.wavelength - wavelength used for propagation.txAntennaHeight - tranmitting antenna hight.rxAntennaHeight - 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
<p>PROP_CalculateFading</p> <p>To calculate fading between two node.</p>	<p>void PROP_CalculateFading (PropTxInfo* propTxInfo, Node* node2, int channelIndex, clocktype currentTime, float* fading_dB, double* channelReal, double* channelImag)</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
<p>PROP_CalculateRxPowerAndPropagationDelay</p> <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>BOOL PROP_CalculateRxPowerAndPropagationDelay (Message* msg, int channelIndex, PropChannel* propChannel, PropTxInfo* propTxInfo, Node* txNode, Node* rxNode, PropPathProfile* pathProfile)</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
<p>PROP_CalculateRxPowerAndPropagationDelay</p>	<p>BOOL PROP_CalculateRxPowerAndPropagationDelay (Message* msg, int channelIndex, PropChannel* propChannel,</p>

<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">• <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
<p>PROP_MotionObtainfadingStretchingFactor</p> <p>Get a stretching factor for fast moving objects.</p>	<p>void PROP_MotionObtainfadingStretchingFactor (PropTxInfo* propTxInfo, Node* receiver, int channelIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>propTxInfo</code> - Transmitter information• <code>receiver</code> - Receiver node.• <code>channelIndex</code> - channel number <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
<p>PROP_UpdatePathProfiles</p> <p>UpdatePathProfiles</p>	<p>void PROP_UpdatePathProfiles (Node* node)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>node</code> - Node that is <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
<p>PROP_ReleaseSignal</p> <p>Release (transmit) the signal</p>	<p>void PROP_ReleaseSignal (Node* node, Message* msg, int phyIndex, int channelIndex, float txPower_dBm, clocktype duration, clocktype delayUntilAirborne)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>node</code> - Node that is• <code>msg</code> - Signal to be transmitted

	<ul style="list-style-type: none">• <code>phyIndex</code> - PHY data index• <code>channelIndex</code> - chanel 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
<p>PROP_SubscribeChannel</p> <p>Start subscribing (listening to) a channel</p>	<p><code>void PROP_SubscribeChannel</code> (Node* node, int phyIndex, int channelIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>node</code> - Node that is• <code>phyIndex</code> - interface index• <code>channelIndex</code> - chanel index <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
<p>PROP_UnsubscribeChannel</p> <p>Stop subscription of (listening to) a channel</p>	<p><code>void PROP_UnsubscribeChannel</code> (Node* node, int phyIndex, int channelIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>node</code> - Node that is• <code>phyIndex</code> - interface index• <code>channelIndex</code> - chanel index <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
<p>PROP_UnreferenceSignal</p> <p>Unreference a signal (internal use)</p>	<p><code>void PROP_UnreferenceSignal</code> (Node* node, Message* msg)</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
<p>PROP_CalculateInterNodePathLossOnChannel</p>	<p><code>void PROP_CalculateInterNodePathLossOnChannel</code> (Node* node, int channelIndex, int* numNodesOnChannel,</p>

Calculate inter-node pathloss, distance values between all the nodes on a given channel	<div>NodeAddress* nodeIdList, float** pathloss_dB, float** distance)</div> <div>Parameters:</div> <div><ul style="list-style-type: none">node - any valid nodechannelIndex - selected channel instancenumNodesOnChannel - number of nodes using this channelnodeIdList - list of (numNodesOnChannel) nodeIdspathloss_dB - 2D pathloss array for nodes indistance - 2D array of inter-node distances</div> <div>Returns:</div> <div><ul style="list-style-type: none">void - None</div>
<div>PROP_CalculatePropagationDelay</div> <div>Calculate the wireless propagation delay for the given distance and propagation speed.</div>	<div>clocktype PROP_CalculatePropagationDelay (double distance, double propSpeed, PartitionData* partitionData, int channelIndex, int coordinateSystemType, Coordinates* fromPosition, Coordinates* toPosition)</div> <div>Parameters:</div> <div><ul style="list-style-type: none">distance - Propagation distancepropSpeed - Propagation speedpartitionData - Partition datachannelIndex - Channel index or -1 for p2pcoordinateSystemType - Coordinate system typefromPosition - Source positiontoPosition - Destination position</div> <div>Returns:</div> <div><ul style="list-style-type: none">clocktype - Calculated propagation delay COMMENTS :: + partitionData can be used to get the simulation time or terrain data + channelIndex 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. + fromPosition and toPosition are not used right now. They can be used to calculate location specific delay.</div>
<div>PROP_Reset</div> <div>Reset previous channel remove/add node to propChannel for signal delivery, in propagation_private.</div>	<div>void PROP_Reset (Node* node, int phyIndex, char* newChannelListenable)</div> <div>Parameters:</div> <div><ul style="list-style-type: none">node - Node that is being instantiated inphyIndex - interface indexnewChannelListenable - new channel</div>

	<div>Returns:</div> <div><ul style="list-style-type: none"><code>void</code> - None</div>
<div>PROP_AddNodeToList</div> <div>add node to propChannel nodeList need to make sure that node is not already exists in list before adding.</div>	<div>void PROP_AddNodeToList (Node* node, int channelIndex)</div> <div>Parameters:</div> <div><ul style="list-style-type: none"><code>node</code> - the node<code>channelIndex</code> - channel index</div> <div>Returns:</div> <div><ul style="list-style-type: none"><code>void</code> - None</div>
<div>PROP_RemoveNodeFromList</div> <div>remove node from propChannel nodeList need to make sure that all the interface from that node is not listing on that channel before removing.</div>	<div>void PROP_RemoveNodeFromList (Node* node, int channelIndex)</div> <div>Parameters:</div> <div><ul style="list-style-type: none"><code>node</code> - the node<code>channelIndex</code> - channel index</div> <div>Returns:</div> <div><ul style="list-style-type: none"><code>void</code> - None</div>
<div>PROP_GetChannelFrequency</div> <div>Get channel frequency from profile for PropChannel.</div>	<div>double PROP_GetChannelFrequency (Node* node, int channelIndex)</div> <div>Parameters:</div> <div><ul style="list-style-type: none"><code>node</code> - the node<code>channelIndex</code> - channel index</div> <div>Returns:</div> <div><ul style="list-style-type: none"><code>double</code> - channel frequency</div>
<div>PROP_SetChannelFrequency</div> <div>Set channel frequency from profile for PropChannel.</div>	<div>void PROP_SetChannelFrequency (Node* node, int channelIndex, double channelFrequency)</div> <div>Parameters:</div> <div><ul style="list-style-type: none"><code>node</code> - the node<code>channelIndex</code> - channel index<code>channelFrequency</code> - new channel frequency</div> <div>Returns:</div> <div><ul style="list-style-type: none"><code>void</code> - None</div>

PROP_GetChannelWavelength Get channel wavelength from profile for PropChannel.	<p>double PROP_GetChannelWavelength (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 wavelength
PROP_SetChannelWavelength Set channel wavelength from profile for PropChannel.	<p>void PROP_SetChannelWavelength (Node* node, int channelIndex, double channelWavelength)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>node</code> - the node• <code>channelIndex</code> - channel index• <code>channelWavelength</code> - new channel wavelength <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
PROP_GetChannelDopplerFrequency Get channel doppler freq from profile for PropChannel.	<p>double PROP_GetChannelDopplerFrequency (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 doppler freq
PROP_SetChannelDopplerFrequency Set channel doppler freq from profile for PropChannel.	<p>void PROP_SetChannelDopplerFrequency (Node* node, int channelIndex, double channelDopplerFrequency)</p> <p>Parameters:</p> <ul style="list-style-type: none">• <code>node</code> - the node• <code>channelIndex</code> - channel index• <code>channelDopplerFrequency</code> - new channel doppler freq <p>Returns:</p> <ul style="list-style-type: none">• <code>void</code> - None
PROP_FrequencyOverlap	<p>BOOL PROP_FrequencyOverlap (Node* txNode, Node* rxNode, int txChannelIndex, int rxChannelIndex, int txPhyIndex, int rxPhyIndex)</p>

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 © 2001-2013 [SCALABLE Network Technologies, Inc.](#) All rights reserved.



QualNet 7.1 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

	<p>This enumeration is used by both queues and schedulers to determine the operation of the retrieve functions.</p>
STRUCT	<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>
STRUCT	<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 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>

	<p>This function prototype returns the number of Messages stored in the packetArray.</p>
int	<p>Queue ()</p>
	<p>This function prototype returns the size of the Queue</p>
void	<p>Queue (double serviceTag)</p>
	<p>Set the service tag of the queue</p>
int	<p>Queue (Queue* oldQueue)</p>
	<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>
void	<p>Queue (BOOL suspend)</p>
	<p>This function is proposed to identify and tag misbehaved queue at the interface, so that they can be punished.</p>
void	<p>Queue (int* qDelayVal, int* totalTransmissionVal, const clocktype currentTime, BOOL isResetTotalTransmissionVal)</p>
	<p>This function is proposed for qos information update for Qos Routings like Qospf.</p>
int	<p>Queue ()</p>
	<p>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.</p>
clocktype	<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>
clocktype	<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>
void	<p>Queue (clocktype currentTime)</p>
	<p>This function prototype resets the current period statistics variables, and sets the currentPeriodStartTime to the currentTime.</p>
clocktype	<p>Queue ()</p>

	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) 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.
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) 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	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.
Structure	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.
Structure	QueueAgeInfo
	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.

Function / Macro Detail

Function / Macro	Format
Queue 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.	void Queue (Message* msg, const void* infoField, BOOL* QueueIsFull, const clocktype currentTime, const double serviceTag) Parameters: <ul style="list-style-type: none">msg - Pointer to Message structureinfoField - The infoField parameterQueueIsFull - returns Queue occupancy statuscurrentTime - Current Simulation timeserviceTag - ServiceTag Returns: <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 Set the service tag of the queue	<div>void Queue (double serviceTag)</div> <div>Parameters:</div> <div><ul style="list-style-type: none">serviceTag - the value of the service tag</div> <div>Returns:</div> <div><ul style="list-style-type: none">void - NULL</div>
Queue 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.	<div>int Queue (Queue* oldQueue)</div> <div>Parameters:</div> <div><ul style="list-style-type: none">oldQueue - Old queue pointer</div> <div>Returns:</div> <div><ul style="list-style-type: none">int - Old packetArray</div>
Queue This function is proposed to identify and tag misbehaved queue at the interface, so that they can be punished.	<div>void Queue (BOOL suspend)</div> <div>Parameters:</div> <div><ul style="list-style-type: none">suspend - The queue status</div> <div>Returns:</div> <div><ul style="list-style-type: none">void - Null</div>
Queue This function is proposed for qos information update for Qos Routings like Qospf.	<div>void Queue (int* qDelayVal, int* totalTransmissionVal, const clocktype currentTime, BOOL isResetTotalTransmissionVal)</div> <div>Parameters:</div> <div><ul style="list-style-type: none">qDelayVal - Returning qDelay valuetotalTransmissionVal - Returning totalTransmission valuecurrentTime - Current simulation timeisResetTotalTransmissionVal - Default false</div> <div>Returns:</div> <div><ul style="list-style-type: none">void - Null</div>
Queue This function prototype returns the number of	<div>int Queue ()</div> <div>Parameters:</div> <div>Returns:</div>

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.	<ul style="list-style-type: none">• <code>int</code> - Integer
Queue 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.	clocktype Queue () Parameters: Returns: <ul style="list-style-type: none">• <code>clocktype</code> - Utilize Time.
Queue 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.	clocktype Queue () Parameters: Returns: <ul style="list-style-type: none">• <code>clocktype</code> - Queue Delays.
Queue This function prototype resets the current period statistics variables, and sets the currentPeriodStartTime to the currentTime.	void Queue (clocktype currentTime) Parameters: <ul style="list-style-type: none">• <code>currentTime</code> - Current simulation time. Returns: <ul style="list-style-type: none">• <code>void</code> - Null
Queue This function prototype returns the currentPeriodStartTime.	clocktype Queue () Parameters: Returns: <ul style="list-style-type: none">• <code>clocktype</code> - Current period start time.
Queue 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.	void Queue (Node* node, const char* layer, const int interfaceIndex, const int instanceId, const char* invokingProtocol, const char* splStatStr) Parameters: <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
<p>Queue</p> <p>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 <code>configInfo</code>. Some parameters includes default values, to prevent breaking existing models. [Uses: vide Pseudo code]</p>	<p><code>void Queue</code> (<code>Node* node</code>, <code>const char queueTypeString[]</code>, <code>const int queueSize</code>, <code>const int interfaceIndex</code>, <code>const int queueNumber</code>, <code>const int infoFieldSize</code>, <code>const BOOL enableQueueStat</code>, <code>const BOOL showQueueInGui</code>, <code>const clocktype currentTime</code>, <code>const void* configInfo</code>)</p> <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 <code>infoFieldSize</code> = 0• <code>enableQueueStat</code> - Default <code>enableQueueStat</code> = 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



QualNet 7.1 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_erand (RandomSeed seed) Returns a uniform distribution in [0.0 .. 1.0]
Int32	RANDOM_jrand (RandomSeed seed)

	Returns an integer uniformly distributed between -2^31 and 2^31.
Int32	RANDOM_nrnd (RandomSeed seed)
	Returns an integer uniformly distributed between 0 and 2^31.
void	RANDOM_LoadUserDistributions (NodeInput* nodeInput)
	Loads all user defined distributions.
void	RandomDistribution.init ()
	Initializes the random distribution
void	RandomDistribution.setDistributionUniform (T min, T max)
	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.
void	RandomDistribution.setDistributionUniformInteger (T min, T max)
	This one gives [min, max] for integers.
void	RandomDistribution.setDistributionExponential (T mean)
	Sets the distribution to exponential with the given mean.
void	RandomDistribution.setDistributionGaussian (T sigma)
	Sets the distribution to Gaussian with the given sigma
void	RandomDistribution.setDistributionGaussianInt (T sigma)
	Sets the distribution to Gaussian with the given sigma
void	RandomDistribution.setDistributionPareto (T val1, T val2, double alpha)
	Sets the distribution to the truncated Pareto distribution
void	RandomDistribution.setDistributionPareto4 (T val1, T val2, T val3, double alpha)
	Sets the distribution to the truncated Pareto distribution
void	RandomDistribution.setDistributionGeneralPareto (T val1, T val2, double alpha)

	Sets the distribution to general Pareto distribution
void	RandomDistribution.setDistributionParetoUntruncated (double alpha)
	Sets the distribution to the truncated Pareto distribution
void	RandomDistribution.setDistributionDeterministic (T val)
	The distribution will always return val.
void	RandomDistribution.setDistributionNull ()
	The distribution will return 0. This is used for initialization.
int	RandomDistribution.setDistribution (char* inputString, char* printStr, RandomDataType dataType)
	Sets the distribution by parsing string input.
T	RandomDistribution.getRandomNumber (RandomSeed seed, Node* node)
	These two functions return the next random number from the defined distribution.
void	RandomDistribution.setSeed (UInt32 globalSeed, UInt32 nodeId, UInt32 protocolId, UInt32 instanceId)
	Calls RANDOM_SetSeed on the member seed.
void	RandomDistribution.setSeed (RandomSeed seed)
	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	ArbitraryDistribution
	Stores a user defined distribution.

Function / Macro Detail

Function / Macro	Format
RANDOM_SetSeed 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.	void RANDOM_SetSeed (RandomSeed seed, UInt32 globalSeed, UInt32 nodeId, UInt32 protocolId, UInt32 instanceId) Parameters: <ul style="list-style-type: none">• seed - the seed to be set.• globalSeed - the scenario's global seed, i.e. SEED in the• nodeId - the node's ID• protocolId - the protocol number, as defined in the layer• instanceId - the instance of this protocol, often the Returns: <ul style="list-style-type: none">• void - None
RANDOM_erand Returns a uniform distribution in [0.0 .. 1.0]	double RANDOM_erand (RandomSeed seed) Parameters: <ul style="list-style-type: none">• seed - the seed for this random stream. Returns: <ul style="list-style-type: none">• double - a random number
RANDOM_jrand Returns an integer uniformly distributed between -2^31 and 2^31.	Int32 RANDOM_jrand (RandomSeed seed) Parameters: <ul style="list-style-type: none">• seed - the seed for this random stream. Returns: <ul style="list-style-type: none">• Int32 - a random number

RANDOM_nrand Returns an integer uniformly distributed between 0 and 2^31.	Int32 RANDOM_nrand (RandomSeed seed) Parameters: <ul style="list-style-type: none">seed - the seed for this random stream. Returns: <ul style="list-style-type: none">Int32 - a random number
RANDOM_LoadUserDistributions Loads all user defined distributions.	void RANDOM_LoadUserDistributions (NodeInput* nodeInput) Parameters: <ul style="list-style-type: none">nodeInput - the .config file Returns: <ul style="list-style-type: none">void - None
RandomDistribution.init Initializes the random distribution	void RandomDistribution.init () Parameters: Returns: <ul style="list-style-type: none">void - None
RandomDistribution.setDistributionUniform 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.	void RandomDistribution.setDistributionUniform (T min, T max) Parameters: <ul style="list-style-type: none">min - the low end of the rangemax - the high end of the range Returns: <ul style="list-style-type: none">void - None
RandomDistribution.setDistributionUniformInteger This one gives [min, max] for integers.	void RandomDistribution.setDistributionUniformInteger (T min, T max) Parameters: <ul style="list-style-type: none">min - the low end of the rangemax - the high end of the range Returns: <ul style="list-style-type: none">void - None
RandomDistribution.setDistributionExponential	void RandomDistribution.setDistributionExponential (T mean) Parameters:

Sets the distribution to exponential with the given mean.	<ul style="list-style-type: none">mean - the mean value of the distribution Returns: <ul style="list-style-type: none">void - None
RandomDistribution.setDistributionGaussian Sets the distribution to Gaussian with the given sigma	void RandomDistribution.setDistributionGaussian (T sigma) Parameters: <ul style="list-style-type: none">sigma - the sigma value Returns: <ul style="list-style-type: none">void - None
RandomDistribution.setDistributionGaussianInt Sets the distribution to Gaussian with the given sigma	void RandomDistribution.setDistributionGaussianInt (T sigma) Parameters: <ul style="list-style-type: none">sigma - the sigma value Returns: <ul style="list-style-type: none">void - None
RandomDistribution.setDistributionPareto Sets the distribution to the truncated Pareto distribution	void RandomDistribution.setDistributionPareto (T val1, T val2, double alpha) Parameters: <ul style="list-style-type: none">val1 - the low end of the rangeval2 - the high end of the rangealpha - the alpha value Returns: <ul style="list-style-type: none">void - None
RandomDistribution.setDistributionPareto4 Sets the distribution to the truncated Pareto distribution	void RandomDistribution.setDistributionPareto4 (T val1, T val2, T val3, double alpha) Parameters: <ul style="list-style-type: none">val1 - the minimum value of Pareto distributionval2 - the low end of the rangeval3 - the high end of the rangealpha - the alpha value Returns:

	<ul style="list-style-type: none">void - None
RandomDistribution.setDistributionGeneralPareto Sets the distribution to general Pareto distribution	void RandomDistribution.setDistributionGeneralPareto (T val1, T val2, double alpha) Parameters: <ul style="list-style-type: none">val1 - the low end of the rangeval2 - the high end of the rangealpha - the alpha value Returns: <ul style="list-style-type: none">void - None
RandomDistribution.setDistributionParetoUntruncated Sets the distribution to the truncated Pareto distribution	void RandomDistribution.setDistributionParetoUntruncated (double alpha) Parameters: <ul style="list-style-type: none">alpha - the alpha value Returns: <ul style="list-style-type: none">void - None
RandomDistribution.setDistributionDeterministic The distribution will always return val.	void RandomDistribution.setDistributionDeterministic (T val) Parameters: <ul style="list-style-type: none">val - the value to return Returns: <ul style="list-style-type: none">void - None
RandomDistribution.setDistributionNull The distribution will return 0. This is used for initialization.	void RandomDistribution.setDistributionNull () Parameters: Returns: <ul style="list-style-type: none">void - None
RandomDistribution.setDistribution Sets the distribution by parsing string input.	int RandomDistribution.setDistribution (char* inputString, char* printStr, RandomDataType dataType) Parameters: <ul style="list-style-type: none">inputString - the input string, typically from a lineprintStr - usually the name of the callingdataType - the data type of the template class is Returns:

	<ul style="list-style-type: none">• <code>int</code> - returns the number of tokens read from the input string
<p>RandomDistribution.getRandomNumber</p> <p>These two functions return the next random number from the defined distribution.</p>	<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
<p>RandomDistribution.setSeed</p> <p>Calls RANDOM_SetSeed on the member seed.</p>	<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
<p>RandomDistribution.setSeed</p> <p>Copies the parameter seed into the member seed.</p>	<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



QualNet 7.1 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)

	<p>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.</p>
int	<p>Scheduler(const int priority)</p>
	<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>
void	<p>Scheduler(int queueIndex, int* qDelayVal, int* totalTransmissionVal, const clocktype currentTime, BOOL isResetTotalTransmissionVal)</p>
	<p>This function enable Qos monitoring for all queues that the scheduler controls.</p>
void	<p>Scheduler(int priority, int packetSize, const clocktype currentTime)</p>
	<p>This function enable data collection for performance study of schedulers.</p>
void	<p>Scheduler(Node* node, const char* layer, const int interfaceIndex, const int instanceId, const char* invokingProtocol, const char* splStatStr)</p>
	<p>This function invokes queue finalization.</p>
void	<p>SCHEDULER_Setup(Scheduler** scheduler, const char schedulerTypeString[], BOOL enableSchedulerStat, const char* graphDataStr)</p>
	<p>This function runs the generic and then algorithm-specific scheduler initialization routine.</p>
int	<p>GenericPacketClassifier(Scheduler* scheduler, int pktPriority)</p>
	<p>Classify a packet for a specific queue</p>

Constant / Data Structure Detail

Constant	<p>DEFAULT_QUEUE_COUNT 3</p> <p>Default number of queue per interface</p>
Structure	<p>QueueData</p> <p>This structure contains pointers to queue structures, default function behaviors, and statistics for the scheduler</p>

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>void Scheduler (int queueIndex, int* qDelayVal, int* totalTransmissionVal, const clocktype currentTime, BOOL isResetTotalTransmissionVal)</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>void Scheduler (int priority, int packetSize, const clocktype currentTime)</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>void Scheduler (Node* node, const char* layer, const int interfaceIndex, const int instanceId, const char* invokingProtocol, const char* splStatStr)</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">splStatStr - Special string for stat print Returns: <ul style="list-style-type: none">void - Null
SCHEDULER_Setup This function runs the generic and then algorithm-specific scheduler initialization routine.	void SCHEDULER_Setup (Scheduler** scheduler, const char schedulerTypeString[], BOOL enableSchedulerStat, const char* graphDataStr) Parameters: <ul style="list-style-type: none">scheduler - Pointer of pointer to Scheduler classschedulerTypeString[] - Scheduler Type stringenableSchedulerStat - Scheduler Statistics is set YES or NOgraphDataStr - Scheduler's graph statistics is set or not Returns: <ul style="list-style-type: none">void - Null
GenericPacketClassifier Classify a packet for a specific queue	int GenericPacketClassifier (Scheduler* scheduler, int pktPriority) Parameters: <ul style="list-style-type: none">scheduler - Pointer to a Scheduler class.pktPriority - Incoming packet's priority Returns: <ul style="list-style-type: none">int - 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 © 2001-2013 [SCALABLE Network Technologies, Inc.](#) All rights reserved.



QualNet 7.1 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
double	MsTmWinTotalSum (MsTmWin* pWin, clocktype theTime)
	computes the total data sum
double	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
<p>MsTmWinNewData</p> <p>updates time sliding window with the given new data</p>	<p>void MsTmWinNewData (MsTmWin* pWin, double data, clocktype theTime)</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
<p>MsTmWinWinSize</p> <p>returns the window size</p>	<p>clocktype MsTmWinWinSize (MsTmWin* pWin, clocktype theTime)</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
<p>MsTmWinSum</p> <p>computes the data sum of the window</p>	<p>double MsTmWinSum (MsTmWin* pWin, clocktype theTime)</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
<p>MsTmWinAvg</p> <p>computes the data average of the window</p>	<p>double MsTmWinAvg (MsTmWin* pWin, clocktype theTime)</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
<p>MsTmWinTotalSum</p> <p>computes the total data sum</p>	<p>double MsTmWinTotalSum (<code>MsTmWin* pWin</code>, <code>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
<p>MsTmWinTotalAvg</p> <p>computes the total data average</p>	<p>double MsTmWinTotalAvg (<code>MsTmWin* pWin</code>, <code>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 © 2001-2013 [SCALABLE Network Technologies, Inc.](#) All rights reserved.

QualNet 7.1 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.
void	TRACE_DisableTraceXML (Node* node, TraceProtocolType protocol, char* protocolName, BOOL writeMap)
	Disable XML trace for a particular protocol.
void	TRACE_WriteToBufferXML (Node* node, char* buf)
	Write trace information to a buffer, which will then be printed to a file.
void	TRACE_WriteTraceHeader (FILE* fp)
	Write trace header information to the partition's trace file
void	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 nodenodeInput - access to configuration file Returns: <ul style="list-style-type: none">void - NULL

TRACE_IsTraceAll Determine if TRACE-ALL is enabled from configuration file.	BOOL TRACE_IsTraceAll (Node* node) Parameters: <ul style="list-style-type: none">node - this node Returns: <ul style="list-style-type: none">BOOL - TRUE if TRACE-ALL is enabled, FALSE otherwise.
TRACE_PrintTrace Print trace information to file. To be used with Tracer.	void TRACE_PrintTrace (Node* node, Message* message, TraceLayerType layerType, PacketDirection pktDirection, ActionData* actionData) Parameters: <ul style="list-style-type: none">node - this nodemessage - Packet to print trace info from.layerType - Layer that is calling this function.pktDirection - If the packet is coming out ofactionData - more details about the packet action Returns: <ul style="list-style-type: none">void - NULL
TRACE_PrintTrace 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) Parameters: <ul style="list-style-type: none">node - this nodemessage - Packet to print trace info from.layerType - Layer that is calling this function.pktDirection - If the packet is coming out ofactionData - more details about the packet actionnetType - The network type. Returns: <ul style="list-style-type: none">void - NULL
TRACE_EnableTraceXML	void TRACE_EnableTraceXML (Node* node, TraceProtocolType protocol, char* protocolName, TracePrintXMLFn xmlPrintFn, BOOL writeMap) Parameters:

Enable XML trace for a particular protocol.	<ul style="list-style-type: none">node - this nodeprotocol - protocol to enable trace forprotocolName - name of protocolxmlPrintFn - callback functionwriteMap - flag to print protocol ID map Returns: <ul style="list-style-type: none">void - NULL
TRACE_EnableTraceXML Enable XML trace for a particular protocol.	void TRACE_EnableTraceXML (Node* node, TraceProtocolType protocol, char* protocolName, TracePrintXMLFn xmlPrintFn, BOOL writeMap) Parameters: <ul style="list-style-type: none">node - this nodeprotocol - protocol to enable trace forprotocolName - name of protocolxmlPrintFn - callback functionwriteMap - flag to print protocol ID map Returns: <ul style="list-style-type: none">void - NULL
TRACE_DisableTraceXML Disable XML trace for a particular protocol.	void TRACE_DisableTraceXML (Node* node, TraceProtocolType protocol, char* protocolName, BOOL writeMap) Parameters: <ul style="list-style-type: none">node - this nodeprotocol - protocol to enable trace forprotocolName - name of protocolwriteMap - flag to print protocol ID map Returns: <ul style="list-style-type: none">void - NULL
TRACE_WriteToBufferXML Write trace information to a buffer, which	void TRACE_WriteToBufferXML (Node* node, char* buf) Parameters: <ul style="list-style-type: none">node - This node.

will then be printed to a file.	<ul style="list-style-type: none">• buf - Content to print to trace file. Returns: <ul style="list-style-type: none">• void - NULL
TRACE_WriteTraceHeader Write trace header information to the partition's trace file	void TRACE_WriteTraceHeader (FILE* fp) Parameters: <ul style="list-style-type: none">• fp - pointer to the trace file. Returns: <ul style="list-style-type: none">• void - NULL
TRACE_WriteXMLTraceTail Write trace tail information to the partition's trace file	void TRACE_WriteXMLTraceTail (FILE* fp) Parameters: <ul style="list-style-type: none">• fp - pointer to the trace file. Returns: <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 © 2001-2013 [SCALABLE Network Technologies, Inc.](#) All rights reserved.



QualNet 7.1 API Reference

TRANSPORT LAYER

This file describes data structures and functions used by the Tansport 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 © 2001-2013 [SCALABLE Network Technologies, Inc.](#) All rights reserved.

QualNet 7.1 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 CECREASE DISSATISFACTION The step value that the user dissatisfaction degree is decreased each time.
CONSTANT	USER_DEFAULT_STATUS_START_TIME Defines the default user status start 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	USER_HandleCallUpdate (Node* node, UserApplicationStatus appStatus) Reaction to the status change of an application session
void	USER_HandleUserLayerEvent (Node* node, Message* msg) Handle messages and events for user layer
void	USER_SetTrafficPattern (Node* node) Set a user's traffic pattern based on its profile.
void	USER_SetApplicationArrival (Node* node) Schedule an application arrival time.

Constant / Data Structure Detail

Constant	USER_PHONE_STARTUP_DELAY 5S Delay from a cellphone is powered on until it can start working.
Constant	USER_INCREASE_DISSATISFACTION 0.1 The step value that the user dissatisfaction degree is increased each time.
Constant	USER_CECREASE_DISSATISFACTION -0.1 The step value that the user dissatisfaction degree is decreased each time.
Constant	USER_DEFAULT_STATUS_START_TIME 10S Defines the default user status start time
Enumeration	UserApplicationStatus

	Status of an user application session.
Structure	UserAppInfo Data structure stores information of one user application session.
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.

	<div>Returns:<ul style="list-style-type: none">void - NULL</div>
<div><div>USER_SetApplicationArrival</div><div>Schedule an application arrival time.</div></div>	<div><div>void USER_SetApplicationArrival (Node* node)</div><div>Parameters:<ul style="list-style-type: none">node - Pointer to node.</div><div>Returns:<ul style="list-style-type: none">void - NULL</div></div>



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 © 2001-2013 [SCALABLE Network Technologies, Inc.](#) All rights reserved.



QualNet 7.1 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	WallClock (void) This method returns true if the WallClock is currently in the paused state.
double	WallClock () Return the real time multiple
void	WallClock (void) 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.
void	WallClock (void) Allows pausing of the WallClock
double	WallClock (void) Get the amount of time, in seconds, spent paused.

Function / Macro Detail

Function / Macro	Format
WallClock This method returns true if the WallClock is	BOOL WallClock (void) Parameters: <ul style="list-style-type: none">void - None

currently in the paused state.	<div>Returns:<ul style="list-style-type: none">• <code>BOOL</code> - TRUE or FALSE</div>
<div>WallClock</div> <div>Return the real time multiple</div>	<div>double WallClock ()</div> <div>Parameters:</div> <div>Returns:<ul style="list-style-type: none">• <code>double</code> - None</div>
<div>WallClock</div> <div>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.</div>	<div>void WallClock (void)</div> <div>Parameters:<ul style="list-style-type: none">• <code>void</code> - None</div> <div>Returns:<ul style="list-style-type: none">• <code>void</code> - None</div>
<div>WallClock</div> <div>Allows pausing of the WallClock</div>	<div>void WallClock (void)</div> <div>Parameters:<ul style="list-style-type: none">• <code>void</code> - None</div> <div>Returns:<ul style="list-style-type: none">• <code>void</code> - None</div>
<div>WallClock</div> <div>Get the amount of time, in seconds, spent paused.</div>	<div>double WallClock (void)</div> <div>Parameters:<ul style="list-style-type: none">• <code>void</code> - None</div> <div>Returns:<ul style="list-style-type: none">• <code>double</code> - The amount of time paused, in seconds.</div>



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 © 2001-2013 [SCALABLE Network Technologies, Inc.](#) All rights reserved.