



# EXata 5.1

## 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](http://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](http://SCALABLE-NETWORKS.COM)**

# EXata 5.1 API Reference

## EXata API

### [3D\\_MATH](#)

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

### [ANTENNA](#)

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

### [ANTENNA\\_GLOBAL](#)

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

### [API](#)

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

### [APP\\_UTIL](#)

This file describes Application Layer utility functions.

### [APPLICATION\\_LAYER](#)

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

### [BUFFER](#)

This file describes data structures and functions to implement buffers.

### [CIRCULAR-BUFFER](#)

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

### [CLOCK](#)

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

### [COORDINATES](#)

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

### [ERROR](#)

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

### [EXTERNAL](#)

This file defines the generic interface to external modules.

### [EXTERNAL\\_SOCKET](#)

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

### [EXTERNAL UTILITIES](#)

This file describes utilities for external interfaces.

### [FILEIO](#)

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

### [GUI](#)

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

### [IP](#)

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

[IPv6](#)

This file describes data structures and parameters used in network layer are defined here.

[LIST](#)

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

[MAC LAYER](#)

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

[MAIN](#)

This file contains some common definitions.

[MAPPING](#)

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

[MEMORY](#)

This file describes the memory management data structures and functions.

[MESSAGE](#)

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

[MOBILITY](#)

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

[MUTEX](#)

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

[NETWORK LAYER](#)

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

[NODE](#)

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

[PARALLEL](#)

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

[PARTITION](#)

This file contains declarations of some functions for partition threads.

[PHYSICAL LAYER](#)

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

[PROPAGATION](#)

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

[QUEUES](#)

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

[RANDOM NUMBERS](#)

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

[SCHEDULERS](#)

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

[SLIDING-WINDOW](#)

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

[TRACE](#)

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

[TRANSPORT LAYER](#)

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

**USER**

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

**WALLCLOCK**

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



# EXata 5.1 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	<a href="#">Vector3</a>
	<p>This is used to hold 3D points and vectors. This will eventually be added upon to create a robust class with operator overloading. For now we just need an x, y, z.</p>
STRUCT	<a href="#">Triangle3</a>
	<p>This struture will hold information for one triangle.</p>

### Function / Macro Summary

Return Type	Summary
Vector3	<p><a href="#">MATH_CrossProduct</a>(Vector3 vector1, Vector3 vector2)</p> <p>Returns a perpendicular vector from 2 given vectors by taking the cross product.</p>
Vector3	<p><a href="#">MATH_Vector</a>(Vector3 point1, Vector3 point2)</p> <p>Returns a vector between 2 points</p>
double	<p><a href="#">MATH_Magnitude</a>(Vector3 vector)</p> <p>Returns the magnitude of a normal (or any other vector)</p>
Vector3	<p><a href="#">MATH_Normalize</a>(Vector3 vector)</p> <p>Returns a normalized vector (of exactly length 1)</p>

Vector3	<a href="#">MATH_Normal</a> (Vector3[] triangle)
	Returns the direction the polygon is facing
double	<a href="#">MATH_PlaneDistance</a> (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	<a href="#">MATH_IntersectedPlane</a> (Vector3[] polygon, Vector3[] line, Vector3& normal, double& originDistance)
	Takes a triangle (plane) and line and returns true if they intersected
double	<a href="#">MATH_DotProduct</a> (Vector3 vector1, Vector3 vector2)
	Returns the dot product between 2 vectors.
double	<a href="#">MATH_AngleBetweenVectors</a> (Vector3 vector1, Vector3 vector2)
	This returns the angle between 2 vectors
Vector3	<a href="#">MATH_IntersectionPoint</a> (Vector3 normal, Vector3[] line, double distance)
	Returns an intersection point of a polygon and a line (assuming intersects the plane)
BOOL	<a href="#">MATH_InsidePolygon</a> (Vector3 intersection, Vector3[] polygon, int verticeCount)
	Returns true if the intersection point is inside of the polygon
BOOL	<a href="#">MATH_IntersectedPolygon</a> (Vector3[] polygon, Vector3[] line, int verticeCount)
	Tests collision between a line and polygon
double	<a href="#">MATH_Distance</a> (Vector3 point1, Vector3 point2)
	Returns the distance between 2 3D points
BOOL	<a href="#">MATH_LineIntersects</a> (Vector3[] line1, Vector3[] line2)
	Checks whether two lines intersect each other or not.
Vector3	<a href="#">MATH_ReturnLineToLineIntersectionPoint</a> (Vector3[] line1, Vector3[] line2)
	Returns the point of intersection between two lines.

BOOL	<a href="#">MATH_IsPointOnLine</a> (Vector3 point, Vector3[] line)
void	<a href="#">MATH_ConvertXYToLatLong</a> (double x1, double y1, double latitude, double longitude)  Converts given cartesian coordinates to Latitide and Longitude

**Constant / Data Structure Detail**

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

	<p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>Vector3</code> - a vector between the two points</li> </ul>
<b>MATH_Magnitude</b>	<p><code>double MATH_Magnitude (Vector3 vector)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>vector</code> - a vector</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>double</code> - the magnitude of the vector</li> </ul>
<b>MATH_Normalize</b>	<p><code>Vector3 MATH_Normalize (Vector3 vector)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>vector</code> - a vector</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>Vector3</code> - a normalized vector</li> </ul>
<b>MATH_Normal</b>	<p><code>Vector3 MATH_Normal (Vector3[] triangle)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>triangle</code> - an array of vectors representing a polygon</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>Vector3</code> - the direction vector</li> </ul>
<b>MATH_PlaneDistance</b>	<p><code>double MATH_PlaneDistance (Vector3 vector, Vector3 point)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>vector</code> - a vector</li> <li>• <code>point</code> - a point</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>double</code> - the plane's distance from the origin (0,0,0)</li> </ul>
<b>MATH_IntersectedPlane</b>	<p><code>BOOL MATH_IntersectedPlane (Vector3[] polygon, Vector3[] line, Vector3&amp; normal, double&amp; originDistance)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>polygon</code> - a polygon</li> <li>• <code>line</code> - a line</li> <li>• <code>normal</code> - a normalized vector</li> </ul>

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

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

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



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

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

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



# EXata 5.1 API Reference

## ANTENNA

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

### Constant / Data Structure Summary

Type	Name
CONSTANT	<a href="#">ANTENNA_DEFAULT_HEIGHT</a> Default height of the antenna
CONSTANT	<a href="#">ANTENNA_DEFAULT_GAIN_dBi</a> Default gain of the antenna
CONSTANT	<a href="#">ANTENNA_DEFAULT_EFFICIENCY</a> Default efficiency of the antenna
CONSTANT	<a href="#">ANTENNA_DEFAULT_MISMATCH_LOSS_db</a> Default mismatch loss of the antenna
CONSTANT	<a href="#">ANTENNA_DEFAULT_CONNECTION_LOSS_db</a> Default connection loss of the antenna
CONSTANT	<a href="#">ANTENNA_DEFAULT_CABLE_LOSS_db</a> Default cable loss of the antenna
CONSTANT	<a href="#">ANTENNA_LOWEST_GAIN_dBi</a> Default minimum gain of the antenna
CONSTANT	<a href="#">ANTENNA_DEFAULT_PATTERN</a>

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

## Function / Macro Summary

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

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

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

	<p>Set antenna for best gain using the azimuth.</p>
void	<code><a href="#">ANTENNA_SetSteeringAngle</a>(Node* node, int phyIndex, Orientation* angle)</code>
	<p>Get steering angle of the antenna.</p>
void	<code><a href="#">ANTENNA_GetSteeringAngle</a>(Node* node, int phyIndex, Orientation angle)</code>
	<p>Set the steering angle of the antenna</p>
void	<code><a href="#">ANTENNA_SetSteeringAngle</a>(Node* node, int phyIndex, Orientation* angle)</code>
	<p>Read in the ASCII pattern .</p>
void	<code><a href="#">ANTENNA_ReturnAsciiPatternFile</a>(Node* node, int phyIndex, const NodeInput* antennaModelInput)</code>
	<p>Read in the NSMA pattern .</p>
void	<code><a href="#">ANTENNA_ReturnNsmaPatternFile</a>(Node* node, int phyIndex, const NodeInput* antennaModelInput, AntennaPatterns* antennaPatterns)</code>
	<p>Read in the NSMA pattern .</p>
void	<code><a href="#">ANTENNA_ReturnTraditionalPatternFile</a>(Node* node, int phyIndex, const NodeInput* antennaModelInput)</code>
	<p>Used to read Qualnet Traditional pattern file</p>
NodeInput *	<code><a href="#">ANTENNA_MakeAntennaModelInput</a>(Node* node, char* buf)</code>
	<p>Reads the antenna configuration parameters into the NodeInput structure.</p>

## Constant / Data Structure Detail

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

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

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

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

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

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

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

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

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



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

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

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



# EXata 5.1 API Reference

## ANTENNA\_GLOBAL

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

### Constant / Data Structure Summary

Type	Name
CONSTANT	<a href="#">MAX_ANTENNA_MODELS</a>  Maximum number of models to allow.
CONSTANT	<a href="#">MAX_ANTENNA_PATTERNS</a>  Maximum number of antenna patterns to allow.
ENUMERATION	<a href="#">AntennaModelType</a>  Different types of antenna models supported.
ENUMERATION	<a href="#">AntennaPatternType</a>  Different types of antenna pattern types supported.
ENUMERATION	<a href="#">NSMAPatternVersion</a>  Different types of NSMA pattern versions supported.
ENUMERATION	<a href="#">AntennaGainUnit</a>  Different types of antenna gain units supported.
ENUMERATION	<a href="#">AntennaPatternUnit</a>  Different types of antenna pattern units supported.
STRUCT	<a href="#">struct_antenna_pattern_element</a>

	Structure for antenna pattern elements <a href="#">struct_antenna_pattern</a>
STRUCT	Structure for antenna pattern <a href="#">struct_antenna_Global_model</a>

**Function / Macro Summary**

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

	Return the model based on the name.
AntennaPattern*	<a href="#">ANTENNA_GlobalAntennaPatternGet</a> (PartitionData* partitionData, const char* antennaPatternName)
void	Return the antenna pattern based on the name. <a href="#">ANTENNA_GeneratePatternName</a> (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	<pre>struct_antenna_pattern_element</pre> <p>Structure for antenna pattern elements</p>
Structure	<pre>struct_antenna_pattern</pre> <p>Structure for antenna pattern</p>
Structure	<pre>struct_antenna_Global_model</pre> <p>Structure for antenna model</p>

## Function / Macro Detail

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

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

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



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

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

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



# EXata 5.1 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	<a href="#">MESSAGE/EVENT</a>  Event/message types exchanged in the simulation
ENUMERATION	<a href="#">TransportType</a>  Transport type to check reliable, unreliable or TADIL network for Link16 or Link11
ENUMERATION	<a href="#">DestinationType</a>  Interface IP address type
STRUCT	<a href="#">PhyBatteryPower</a>  Used by App layer and Phy layer to exchange battery power
STRUCT	<a href="#">PacketNetworkToApp</a>  Network to application layer packet structure
STRUCT	<a href="#">NetworkToTransportInfo</a>  Network To Transport layer Information structure
STRUCT	<a href="#">PacketTransportNetwork</a>  Transport to network layer packet structure
STRUCT	<a href="#">TcpTimerPacket</a>

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

	Application attempts to establish a connection <a href="#">AppToTcpSend</a>
STRUCT	Application wants to send some data over the connection <a href="#">AppToTcpClose</a>
STRUCT	Application wants to release the connection <a href="#">AppToTcpConnSetup</a>
STRUCT	Application sets up connection at the local end Needed for NS TCP to fake connection setup <a href="#">AppQosToNetworkSend</a>
STRUCT	Application uses this structure in its info field to perform the initialization of a new QoS connection with its QoS requirements. <a href="#">NetworkToAppQosConnectionStatus</a>
	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	<a href="#">CHANNEL_Initialize</a> (Node* node, const NodeInput* nodeInput)  Initialization function for channel
void	<a href="#">PHY_Init</a> (Node* node, const NodeInput* nodeInput)  Initialization function for physical layer
void	<a href="#">MAC_Initialize</a> (Node* node, const NodeInput* nodeInput)  Initialization function for the MAC layer
void	<a href="#">NETWORK_PreInit</a> (Node* node, const NodeInput* nodeInput)  Pre-Initialization function for Network layer
void	<a href="#">NETWORK_Initialize</a> (Node* node, const NodeInput* nodeInput)

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

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

	Processes the message/event received by the node thus simulating the USER layer behaviour <a href="#"><b>ATMLAYER2_ProcessEvent</b></a> (Node* node, Message* msg)
void	Processes the message/event of ATM_LAYER2 layer received by the node thus simulating the ATM_LAYER2 layer behaviour <a href="#"><b>ADAPTATION_ProcessEvent</b></a> (Node* node, Message* msg)
void	Processes the message/event received by the node thus simulating the ADAPTATION layer behaviour <a href="#"><b>MAC_RunTimeStat</b></a> (Node* node)
void	To print runtime statistics for the MAC layer <a href="#"><b>NETWORK_RunTimeStat</b></a> (Node* node)
void	To print runtime statistics for the NETWORK layer <a href="#"><b>TRANSPORT_RunTimeStat</b></a> (Node* node)
void	To print runtime statistics for the TRANSPORT layer <a href="#"><b>APP_RunTimeStat</b></a> (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

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

**Function / Macro Detail**

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

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

	<p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>ATMLAYER2_Initialize</b>	<p>void <b>ATMLAYER2_Initialize</b> (Node* node, const NodeInput* nodeInput)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - node being initialized</li> <li>• nodeInput - structure containing input file details</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>ADAPTATION_Initialize</b>	<p>void <b>ADAPTATION_Initialize</b> (Node* node, const NodeInput* nodeInput)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - node being initialized</li> <li>• nodeInput - structure containing input file details</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>CHANNEL_Finalize</b>	<p>void <b>CHANNEL_Finalize</b> (Node * node)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Node for which data is collected</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>PHY_Finalize</b>	<p>void <b>PHY_Finalize</b> (Node * node)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Node for which finalization function is called</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>MAC_Finalize</b>	<p>void <b>MAC_Finalize</b> (Node * node)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Node for which finalization function is called</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>

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

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

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

<p>node thus simulating the ADAPTATION layer behaviour</p>	<ul style="list-style-type: none"> <li>• <code>msg</code> - Received message structure</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<p><b>MAC_RunTimeStat</b></p> <p>To print runtime statistics for the MAC layer</p>	<p><code>void MAC_RunTimeStat (Node* node)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - node for which statistics to be printed</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<p><b>NETWORK_RunTimeStat</b></p> <p>To print runtime statistics for the NETWORK layer</p>	<p><code>void NETWORK_RunTimeStat (Node* node)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - node for which statistics to be printed</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<p><b>TRANSPORT_RunTimeStat</b></p> <p>To print runtime statistics for the TRANSPORT layer</p>	<p><code>void TRANSPORT_RunTimeStat (Node* node)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - node for which statistics to be printed</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<p><b>APP_RunTimeStat</b></p> <p>To print runtime statistics for the APPLICATION layer</p>	<p><code>void APP_RunTimeStat (Node* node)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - node for which statistics to be printed</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>



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

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

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



# EXata 5.1 API Reference

## APP\_UTIL

This file describes Application Layer utility functions.

### Function / Macro Summary

Return Type	Summary
MACRO	<p><a href="#">APP_GetTimerType(x)</a></p> <p>Get the timerType for a received App Layer Timer.</p>
AppInfo*	<p><a href="#">APP_RegisterNewApp(Node* node, AppType appType, void * dataPtr)</a></p> <p>Insert a new application into the list of apps on this node.</p>
void	<p><a href="#">APP_SetTimer(Node* node, AppType appType, int connId, short sourcePort, int timerType, clocktype delay)</a></p> <p>Set a new App Layer Timer and send to self after delay.</p>
Message *	<p><a href="#">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 mapUniqueId)</a></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>
Message*	<p><a href="#">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)</a></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>
void	<p><a href="#">APP_TcpServerListen_(Node * node, AppType appType, NodeAddress serverAddr, short serverPort)</a></p> <p>Listen on a server port.</p>
void	<p><a href="#">APP_TcpServerListen_(Node * node, AppType appType, Address serverAddr, short serverPort)</a></p>

	(Overloaded for IPv6) Listen on a server port.
void	<a href="#">APP_TcpCloseConnection</a> (Node * node, int connId)  Close the connection.
void	<a href="#">APP_InitMulticastGroupMembershipIfAny</a> (Node * node, const NodeInput nodeInput)  Start process of joining multicast group if need to do so.
void	<a href="#">APP_CheckMulticastByParsingSourceAndDestString</a> (Node * node, const char * inputString, const char * sourceString, NodeAddress * sourceNodeId, NodeAddress * sourceAddr, const char * destString, NodeAddress * destNodeId, NodeAddress * destAddr, BOOL * isDestMulticast)  Application input parsing API. Parses the source and destination strings. At the same time validates those strings for multicast address.
void	<a href="#">APP_ParsingSourceAndDestString</a> (Node * node, const char * inputString, const char * sourceString, NodeAddress * sourceNodeId, NodeAddress * sourceAddr, const char * destString, NodeAddress * destNodeId, NodeAddress * destAddr, DestinationType * destType)  API to parse the input source and destination strings read from the *.app file. At the same time checks and fills the destination type parameter.
void	<a href="#">APP_ParsingSourceAndDestString</a> (Node * node, const char * inputString, const char * sourceString, NodeId * sourceNodeId, Address * sourceAddr, const char * destString, NodeId * destNodeId, Address * destAddr, DestinationType * destType)  API to parse the input source and destination strings read from the *.app file. At the same time checks and fills the destination type parameter.
AppInfo*	<a href="#">APP_RegisterNewApp</a> (Node* node, AppType appType, void * dataPtr, short myPort)  Also inserts the port number being used for this app in the port table.
BOOL	<a href="#">APP_IsFreePort</a> (Node* node, short portNumber)  there is an application running at the node that uses an AppType that has been assigned the same value as this port number. This is done since applications such as CBR use the value of AppType as destination port.
short	<a href="#">APP_GetFreePort</a> (Node* node)
void	<a href="#">APP_InsertInPortTable</a> (Node* node, AppType appType, short myPort)
unsigned short	<a href="#">APP_GetProtocolType</a> (Node* node, Message* msg)
BOOL	<a href="#">APP_AssignTos</a> (char array tosString, char array tosValString, unsigned * tosVal)

	Application input parsing API. Parses the tos string and tos value strings. At the same time validates those strings for proper ranges.  <b>void APP_UnregisterApp(Node* node, void * dataPtr, bool freeData)</b>
void	Remove an application from list of apps on this node.  <b>APP_UnregisterApp(Node* node, AppType appType, void * dataPtr, short myPort)</b>
BOOL	Also Remove the port number being used for this app in the port table.  <b>APP_IsFreePort(Node* node, short portNumber)</b>  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	<b>APP_RemoveFromPortTable(Node* node, short myPort)</b>
SequenceNumber	<b>APP_ReportStatsDbReceiveEvent(Node* node, Message* msg, SequenceNumber** seqCache, Int64 seqNo, clocktype delay, clocktype jitter, int size, int numRcvd, AppMsgStatus msgStatus)</b>  Report receive event to StatsDB app event table This function will check duplicate and out of order msgs

## Function / Macro Detail

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

<p>Set a new App Layer Timer and send to self after delay.</p>	<ul style="list-style-type: none"> <li>• node - node that is issuing the Timer.</li> <li>• appType - application type</li> <li>• connId - if applicable, the TCP connectionId for this timer</li> <li>• sourcePort - the source port of the application setting</li> <li>• timerType - an integer value that can be used to</li> <li>• delay - send the timer to self after this delay.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<p><b>APP_UdpSendNewHeaderVirtualDataWithPriority</b></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 * <b>APP_UdpSendNewHeaderVirtualDataWithPriority</b> (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"> <li>• node - node that is sending the data.</li> <li>• sourceAddr - the source sending the data.</li> <li>• sourcePort - the application source port.</li> <li>• destAddr - the destination node Id data</li> <li>• destinationPort - the destination port</li> <li>• header - header of the payload.</li> <li>• headerSize - size of the header.</li> <li>• payloadSize - size of the data in bytes.</li> <li>• priority - priority of data.</li> <li>• delay - send the data after this delay.</li> <li>• traceProtocol - specify the type of application</li> <li>• isMdpEnabled - status of MDP layer.</li> <li>• mdpUniqueId - unique id for MPD session.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• Message * - None</li> </ul>
<p><b>APP_UdpSendNewHeaderVirtualDataWithPriority</b></p>	<p>Message* <b>APP_UdpSendNewHeaderVirtualDataWithPriority</b> (Node * node, Address sourceAddr, short sourcePort, Address destAddr, short destinationPort, char* header, int headerSize, int payloadSize, TosType priority,</p>

	<p>(Overloaded for IPv6) Allocate header + virtual data with specified priority and send to UDP. Data is sent to a non-default destination port (port number may not have same value as the AppType).</p>	<p><code>clocktype endTime, UInt32 itemSize, D_Clocktype interval, clocktype delay, TraceProtocolType traceProtocol)</code></p> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• <code>node</code> - node that is sending the data.</li> <li>• <code>sourceAddr</code> - the source sending the data.</li> <li>• <code>sourcePort</code> - the application source port.</li> <li>• <code>destAddr</code> - the destination node Id data</li> <li>• <code>destinationPort</code> - the destination port</li> <li>• <code>header</code> - header of the payload.</li> <li>• <code>headerSize</code> - size of the header.</li> <li>• <code>payloadSize</code> - size of the data in bytes.</li> <li>• <code>priority</code> - priority of data.</li> <li>• <code>endTime</code> - zigbeeApp end time.</li> <li>• <code>itemSize</code> - zigbeeApp item size.</li> <li>• <code>interval</code> - zigbeeApp interval</li> <li>• <code>delay</code> - send the data after this delay.</li> <li>• <code>traceProtocol</code> - specify the type of application</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>• <code>Message*</code> - The created message</li> </ul>
<b>APP_TcpServerListen.</b>	Listen on a server port.	<p><code>void APP_TcpServerListen. (Node * node, AppType appType, NodeAddress serverAddr, short serverPort)</code></p> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• <code>node</code> - Node pointer that the protocol is</li> <li>• <code>appType</code> - which application initiates this request</li> <li>• <code>serverAddr</code> - server address</li> <li>• <code>serverPort</code> - server port number</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>APP_TcpServerListen.</b>		<p><code>void APP_TcpServerListen. (Node * node, AppType appType, Address serverAddr, short serverPort)</code></p> <p><b>Parameters:</b></p>

<p>(Overloaded for IPv6) Listen on a server port.</p>	<ul style="list-style-type: none"> <li>• <code>node</code> - Node pointer that the protocol is</li> <li>• <code>appType</code> - which application initiates this request</li> <li>• <code>serverAddr</code> - server address</li> <li>• <code>serverPort</code> - server port number</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<p><b>APP_TcpCloseConnection</b></p> <p>Close the connection.</p>	<p><code>void APP_TcpCloseConnection (Node * node, int connId)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Node pointer that the protocol is</li> <li>• <code>connId</code> - connection id.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<p><b>APP_InitMulticastGroupMembershipIfAny</b></p> <p>Start process of joining multicast group if need to do so.</p>	<p><code>void APP_InitMulticastGroupMembershipIfAny (Node * node, const NodeInput nodeInput)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - node - node that is joining a group.</li> <li>• <code>nodeInput</code> - used to access configuration file.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<p><b>APP_CheckMulticastByParsingSourceAndDestString</b></p> <p>Application input parsing API. Parses the source and destination strings. At the same time validates those strings for multicast address.</p>	<p><code>void APP_CheckMulticastByParsingSourceAndDestString (Node * node, const char * inputString, const char * sourceString, NodeAddress * sourceNodeId, NodeAddress * sourceAddr, const char * destString, NodeAddress * destNodeId, NodeAddress * destAddr, BOOL * isDestMulticast)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - A pointer to Node.</li> <li>• <code>inputString</code> - The input string.</li> <li>• <code>sourceString</code> - The source string.</li> <li>• <code>sourceNodeId</code> - A pointer to NodeAddress.</li> <li>• <code>sourceAddr</code> - A pointer to NodeAddress.</li> <li>• <code>destString</code> - The destination string.</li> <li>• <code>destNodeId</code> - A pointer to NodeAddress.</li> </ul>

	<ul style="list-style-type: none"> <li>• destAddr - A pointer to NodeAddress.</li> <li>• isDestMulticast - Pointer to multicast checking flag.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>APP_ParsingSourceAndDestString</b>	<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 <b>APP_ParsingSourceAndDestString</b> (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"> <li>• node - A pointer to Node.</li> <li>• inputString - The input string.</li> <li>• sourceString - The source string.</li> <li>• sourceNodeId - A pointer to NodeAddress.</li> <li>• sourceAddr - A pointer to NodeAddress.</li> <li>• destString - The destination string.</li> <li>• destNodeId - A pointer to NodeAddress.</li> <li>• destAddr - A pointer to NodeAddress.</li> <li>• destType - A pointer to Destinationtype.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>APP_ParsingSourceAndDestString</b>	<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 <b>APP_ParsingSourceAndDestString</b> (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"> <li>• node - A pointer to Node.</li> <li>• inputString - The input string.</li> <li>• sourceString - The source string.</li> <li>• sourceNodeId - A pointer to NodeAddress.</li> <li>• sourceAddr - A pointer to NodeAddress.</li> <li>• destString - The destination string.</li> </ul>

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

	<ul style="list-style-type: none"> <li>• <code>myPort</code> - port number to check</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>APP_GetProtocolType</b>	<p>unsigned short <b>APP_GetProtocolType</b> (Node* node, Message* msg)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - node that received the message</li> <li>• <code>msg</code> - pointer to the message received</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>unsigned short</code> - protocol which will receive the message</li> </ul>
<b>APP_AssignTos</b>	<p>BOOL <b>APP_AssignTos</b> (char array tosString, char array tosValString, unsigned * tosVal)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>tosString</code> - The tos string.</li> <li>• <code>tosValString</code> - The tos value string.</li> <li>• <code>tosVal</code> - A pointer to equivalent 8-bit TOS value.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>BOOL</code> - None</li> </ul>
<b>APP_UnregisterApp</b>	<p>void <b>APP_UnregisterApp</b> (Node* node, void * dataPtr, bool freeData)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - node that is unregistering the application.</li> <li>• <code>dataPtr</code> - pointer to the data space for this app.</li> <li>• <code>freeData</code> - if true, free (via <code>MEM_free</code>) the dataPtr</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>APP_UnregisterApp</b>	<p>void <b>APP_UnregisterApp</b> (Node* node, AppType appType, void * dataPtr, short myPort)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - node that is registering the application.</li> <li>• <code>appType</code> - application type</li> </ul>

	<ul style="list-style-type: none"> <li>• <code>dataPtr</code> - pointer to the data space for this app</li> <li>• <code>myPort</code> - port number to be inserted in the port table</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>APP_IsFreePort</b>	<p>BOOL <b>APP_IsFreePort</b> (Node* node, short portNumber)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - node that is checking it's port table.</li> <li>• <code>portNumber</code> - port number to check.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>BOOL</code> - indicates if the port is free.</li> </ul>
<b>APP_RemoveFromPortTable</b>	<p>void <b>APP_RemoveFromPortTable</b> (Node* node, short myPort)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - node that needs to be remove from port table</li> <li>• <code>myPort</code> - port number to check</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>APP_ReportStatsDbReceiveEvent</b>	<p>SequenceNumber <b>APP_ReportStatsDbReceiveEvent</b> (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"> <li>• <code>node</code> - Pointer to a node who receives the msg</li> <li>• <code>msg</code> - The received message or fragment</li> <li>• <code>seqCache</code> - Pointer to the sequence number cache</li> <li>• <code>seqNo</code> - Sequence number of the message or fragment</li> <li>• <code>delay</code> - Delay of the message/fragment</li> <li>• <code>jitter</code> - Smoothed jitter of the received message</li> <li>• <code>size</code> - Size of msg/fragment to be report to db</li> <li>• <code>numRcvd</code> - # of msgs/frags received so far</li> <li>• <code>msgStatus</code> - This is for performance optimization. If</li> </ul>

**Returns:**

- SequenceNumber - Status or out of order or new



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

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

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



# EXata 5.1 API Reference

## APPLICATION LAYER

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

### Constant / Data Structure Summary

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

	Store Link16/IP gateway forwarding table <a href="#">AppInfo</a>
STRUCT	Information relevant to specific app layer protocol <a href="#">PortInfo</a>
	Store port related information <a href="#">AppMultimedia</a>
STRUCT	Store multimedia signalling related information <a href="#">AppData</a>
STRUCT	Details of application data structure in node structure <a href="#">AppTimer</a>
	Timer structure used by applications

**Function / Macro Summary**

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

**Constant / Data Structure Detail**

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

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

**Function / Macro Detail**

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

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



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

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

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



# EXata 5.1 API Reference

## BUFFER

This file describes data structures and functions to implement buffers.

### Constant / Data Structure Summary

Type	Name
STRUCT	<a href="#">DataBuffer</a> structure for the data buffer
STRUCT	<a href="#">ReassemblyBuffer</a> Format for the Reassembly buffer
STRUCT	<a href="#">PacketBuffer</a> structure for the packet buffer

### Function / Macro Summary

Return Type	Summary
MACRO	<a href="#">BUFFER_GetCurrentSize(x)</a> Returns the current size of the buffer
MACRO	<a href="#">BUFFER_GetMaxSize(x)</a> Returns maximum allowable size of the buffer
MACRO	<a href="#">BUFFER_GetData(x)</a> Returns a pointer to the data in the buffer
MACRO	<a href="#">BUFFER_GetAnticipatedSize(x)</a>

	Returns the initial size of the buffer
MACRO	<a href="#">BUFFER_SetCurrentSize(x,y)</a>
	Sets current size of the buffer
MACRO	<a href="#">BUFFER_GetFreeSpace(x)</a>
	Get free space available in the buffer
MACRO	<a href="#">BUFFER_ReturnTop(x)</a>
	Returns top of the buffer
MACRO	<a href="#">BUFFER_NumberOfBlocks(x)</a>
	Returns the no. of blocks in the buffer
void	<a href="#">BUFFER_InitializeDataBuffer(DataBuffer* buffer, int size)</a>
	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	<a href="#">BUFFER_AddSpaceToDataBuffer(DataBuffer* buffer, int size)</a>
	Adding memory space to the buffer
void	<a href="#">BUFFER_ClearDataFromDataBuffer(DataBuffer* buffer, char * startLocation, int size, BOOL destroy)</a>
	clear data from the buffer(already used portion of buffer Not any unused portion unless u clear till end)
void	<a href="#">BUFFER_DestroyDataBuffer(DataBuffer* buffer)</a>
	To Destroy a buffer
void	<a href="#">BUFFER_AddDataToDataBuffer(DataBuffer* buffer, char * data, int size)</a>
	Add data to databuffer
void	<a href="#">BUFFER_RemoveDataFromDataBuffer(DataBuffer* buffer, char* startLocation, int size)</a>
	To remove data from the data buffer
void	<a href="#">InitializeReassemblyBuffer(ReassemblyBuffer* buffer, int size)</a>

	Initialize Reassembly buffer <code>BUFFER_AddDataToAssemblyBuffer(ReassemblyBuffer* buffer, char* data, int size, BOOL allowOverflow)</code>
void	Appending data to the reassembly buffer <code>BUFFER_ClearAssemblyBuffer(ReassemblyBuffer* buffer, int max, BOOL setSize)</code>
void	clear the buffer <code>BUFFER_SetAnticipatedSizeForAssemblyBuffer(ReassemblyBuffer* buffer, int size)</code>
void	To set the anticipated size of the assemblyBuffer <code>BUFFER_AllocatePacketBuffer(int initialSize, int anticipatedHeaderMax, BOOL allowOverflow, char ** dataPtr)</code>
PacketBuffer *	To allocate packet buffer <code>BUFFER_AllocatePacketBufferWithInitialHeader(int initialSize, int initialHeaderSize, int anticipatedHeaderMax, BOOL allowOverflow, char ** dataPtr, char ** headerPtr)</code>
PacketBuffer *	To allocate buffer with Intial header <code>BUFFER_AddHeaderToPacketBuffer(PacketBuffer* buffer, int headerSize, char** headerPtr)</code>
void	To add header to buffer <code>BUFFER_RemoveHeaderFromPacketBuffer(PacketBuffer* buffer, int headerSize, char** dataPtr)</code>
void	To remove header from packet buffer <code>BUFFER_ClearPacketBufferData(PacketBuffer* buffer)</code>
void	To clear data from current buffer <code>BUFFER_FreePacketBuffer(PacketBuffer* buffer)</code>
void	Free the packet buffer passed as argument <code>BUFFER_ConcatenatePacketBuffer(const PacketBuffer* source, PacketBuffer* dest)</code>
	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
<b>BUFFER_GetCurrentSize(x)</b>	Returns the current size of the buffer
<b>BUFFER_GetMaxSize(x)</b>	Returns maximum allowable size of the buffer
<b>BUFFER_GetData(x)</b>	Returns a pointer to the data in the buffer
<b>BUFFER_GetAnticipatedSize(x)</b>	Returns the intial size of the buffer
<b>BUFFER_SetCurrentSize(x,y)</b>	Sets current size of the buffer
<b>BUFFER_GetFreeSpace(x)</b>	Get free space available in the buffer
<b>BUFFER_ReturnTop(x)</b>	Returns top of the buffer
<b>BUFFER_NumberOfBlocks(X)</b>	Returns the no. of blocks in the buffer

<b>BUFFER_InitializeDataBuffer</b>	<p>void <b>BUFFER_InitializeDataBuffer</b> (DataBuffer* buffer, int size)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <b>buffer</b> - buffer to be initialized</li> <li>• <b>size</b> - initial size of the buffer</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <b>void</b> - None</li> </ul>
<b>BUFFER_AddSpaceToDataBuffer</b>	<p>void <b>BUFFER_AddSpaceToDataBuffer</b> (DataBuffer* buffer, int size)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <b>buffer</b> - buffer to which to add space</li> <li>• <b>size</b> - size to be added</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <b>void</b> - None</li> </ul>
<b>BUFFER_ClearDataFromDataBuffer</b>	<p>void <b>BUFFER_ClearDataFromDataBuffer</b> (DataBuffer* buffer, char * startLocation, int size, BOOL destroy)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <b>buffer</b> - buffer from which data is cleared</li> <li>• <b>startLocation</b> - starting location</li> <li>• <b>size</b> - initial size of the buffer</li> <li>• <b>destroy</b> - destroy or not</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <b>void</b> - None</li> </ul>
<b>BUFFER_DestroyDataBuffer</b>	<p>void <b>BUFFER_DestroyDataBuffer</b> (DataBuffer* buffer)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <b>buffer</b> - buffer to be destroyed</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <b>void</b> - None</li> </ul>
<b>BUFFER_AddDataToDataBuffer</b>	<p>void <b>BUFFER_AddDataToDataBuffer</b> (DataBuffer* buffer, char * data, int size)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <b>buffer</b> - buffer to which data is added</li> </ul>

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

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

	<p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>PacketBuffer *</code> - Pointer to packetbuffer</li> </ul>
<b>BUFFER_AddHeaderToPacketBuffer</b>	<p>void <b>BUFFER_AddHeaderToPacketBuffer</b> (<code>PacketBuffer*</code> buffer, <code>int</code> headerSize, <code>char**</code> headerPtr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>buffer</code> - Pointer to PacketBuffer</li> <li>• <code>headerSize</code> - size of header</li> <li>• <code>headerPtr</code> - Pointer to an array of strings</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
To add header to buffer	
<b>BUFFER_RemoveHeaderFromPacketBuffer</b>	<p>void <b>BUFFER_RemoveHeaderFromPacketBuffer</b> (<code>PacketBuffer*</code> buffer, <code>int</code> headerSize, <code>char**</code> dataPtr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>buffer</code> - Pointer to PacketBuffer</li> <li>• <code>headerSize</code> - size of header</li> <li>• <code>dataPtr</code> - Pointer to an strings array</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
To remove header from packet buffer	
<b>BUFFER_ClearPacketBufferData</b>	<p>void <b>BUFFER_ClearPacketBufferData</b> (<code>PacketBuffer*</code> buffer)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>buffer</code> - Pointer to PacketBuffer</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
To clear data from current buffer	
<b>BUFFER_FreePacketBuffer</b>	<p>void <b>BUFFER_FreePacketBuffer</b> (<code>PacketBuffer*</code> buffer)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>buffer</code> - Pointer to PacketBuffer</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
Free the packet buffer passed as argument	
<b>BUFFER_ConcatenatePacketBuffer</b>	<p>void <b>BUFFER_ConcatenatePacketBuffer</b> (<code>const PacketBuffer*</code> source, <code>PacketBuffer*</code> dest)</p>

Add useful contents of source buffer as header to to the destination buffer

Parameters:

- `source` - Pointer to PacketBuffer
- `dest` - Pointer to PacketBuffer

Returns:

- `void` - None



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

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

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



# EXata 5.1 API Reference

## CIRCULAR-BUFFER

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

### Constant / Data Structure Summary

Type	Name
CONSTANT	<a href="#">CIR_BUF_SIZE</a> Default Circular Buffer Size
ENUMERATION	Type of Circular Buffer Operation
ENUMERATION	Type of Wrap operation

### Function / Macro Summary

Return Type	Summary
bool	<a href="#">CircularBuffer.incPos</a> (Int32 increment, Int32 operation)  increment read/write position based on operation
None	<a href="#">CircularBuffer.CircularBuffer</a> ()  Constructor
None	<a href="#">CircularBuffer.CircularBuffer</a> (Int32 queueSize)  Constructor
None	<a href="#">CircularBuffer.CircularBuffer</a> (unsigned short index)  Constructor

## CIRCULAR-BUFFER

None	<a href="#"><code>CircularBuffer.CircularBuffer</code></a> (unsigned short index, Int32 queueSize)
	Constructor
Node	<a href="#"><code>CircularBuffer.-CircularBuffer</code></a> ()
	Destructor
bool	<a href="#"><code>CircularBuffer.create</code></a> (Int32 queueSize)
	Memory allocation for Circular Buffer
void	<a href="#"><code>CircularBuffer.release</code></a> (void )
	To free the allocated memory
void	<a href="#"><code>CircularBuffer.reset</code></a> (void )
	reset position and wrap values
bool	<a href="#"><code>CircularBuffer.getCount</code></a> (Int32& count, Int32 operation)
	gets the number of bytes to read
Int32	<a href="#"><code>CircularBuffer.lengthToEnd</code></a> (Int32 operation)
	get the circular buffer's allocated size
bool	<a href="#"><code>CircularBuffer.readWithCount</code></a> (unsigned char* data, Int32& length)
	Read data from Buffer and pass the length of data read
bool	<a href="#"><code>CircularBuffer.readFromBuffer</code></a> (unsigned char* data, Int32 length, bool noIncrement)
	Reading the required no. of bytes from the circular buffer
bool	<a href="#"><code>CircularBuffer.write</code></a> (unsigned char* data, Int32 length)
	Write to the circular buffer
bool	<a href="#"><code>CircularBuffer.read</code></a> (unsigned char* buffer)
	To Read data from Buffer
Int32	<a href="#"><code>CircularBuffer.getIndex</code></a> (Int32 operation)

	get the circular buffer's allocated size <a href="#"><code>CircularBuffer.getBufSize(void none)</code></a>
Int32	get the circular buffer's allocated size <a href="#"><code>CircularBuffer.getContentSize(void none)</code></a>
void	get the size of the queue's contents. This is the maximum amount of data that may be read. <a href="#"><code>CircularBuffer.dumpToStdout(void none)</code></a>  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	<a href="#"><code>CircularBuffer.getIndex(void none)</code></a>  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
<a href="#"><code>CircularBuffer.incPos</code></a>	bool <a href="#"><code>CircularBuffer.incPos(Int32 increment, Int32 operation)</code></a>  Parameters: <ul style="list-style-type: none"><li>• increment - How much will be incremented</li></ul> increment read/write position based on

operation	<ul style="list-style-type: none"> <li>• <code>operation</code> - Type of Operation (Read or Write )</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>bool</code> - Successful or not</li> </ul>
<b>CircularBuffer.CircularBuffer</b>	<p>None <b>CircularBuffer.CircularBuffer ()</b></p> <p>Parameters:</p>
Constructor	<p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>None</code> - None</li> </ul>
<b>CircularBuffer.CircularBuffer</b>	<p>None <b>CircularBuffer.CircularBuffer (Int32 queueSize)</b></p> <p>Parameters:</p>
Constructor	<ul style="list-style-type: none"> <li>• <code>queueSize</code> - Size of the Queue</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>None</code> - None</li> </ul>
<b>CircularBuffer.CircularBuffer</b>	<p>None <b>CircularBuffer.CircularBuffer (unsigned short index)</b></p> <p>Parameters:</p>
Constructor	<ul style="list-style-type: none"> <li>• <code>index</code> - Circular Buffer Index</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>None</code> - None</li> </ul>
<b>CircularBuffer.CircularBuffer</b>	<p>None <b>CircularBuffer.CircularBuffer (unsigned short index, Int32 queueSize)</b></p> <p>Parameters:</p>
Constructor	<ul style="list-style-type: none"> <li>• <code>index</code> - Circular Buffer Index</li> <li>• <code>queueSize</code> - Size of the queue</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>None</code> - None</li> </ul>
<b>CircularBuffer.~CircularBuffer</b>	<p>Node <b>CircularBuffer.~CircularBuffer ()</b></p> <p>Parameters:</p>
Destructor	<p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>Node</code> - None</li> </ul>

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

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

<b>CircularBuffer.getBufSize</b>	<p>Int32 <b>CircularBuffer.getBufSize</b> (void none)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• none - None</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• Int32 - circular buffer's allocated size</li> </ul>
<b>CircularBuffer.getContentsSize</b>	<p>Int32 <b>CircularBuffer.getContentsSize</b> (void none)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• none - None</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• Int32 - amount of data in buffer</li> </ul>
<b>CircularBuffer.dumpToStdout</b>	<p>void <b>CircularBuffer.dumpToStdout</b> (void none)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• none - None</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>CircularBuffer.getIndex</b>	<p>unsigned short <b>CircularBuffer.getIndex</b> (void none)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• none - None</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• unsigned short - unique index</li> </ul>



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

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





# EXata 5.1 API Reference

## CLOCK

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

### Constant / Data Structure Summary

Type	Name
CONSTANT	<a href="#">CLOCKTYPE_MAX</a>
	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	<a href="#">NANO_SECOND</a>
	Defined as basic unit of clocktype
CONSTANT	<a href="#">MICRO_SECOND</a>
	Defined as 1000 times the basic unit of clocktype
CONSTANT	<a href="#">MILLI_SECOND</a>
	unit of time equal to 1000 times MICRO_SECOND
CONSTANT	<a href="#">SECOND</a>
	simulation unit of time = 1000 times MILLI_SECOND
CONSTANT	<a href="#">MINUTE</a>
	unit of simulation time = 60 times SECOND
CONSTANT	<a href="#">HOUR</a>
	unit of simulation time = 60 times MINUTE
CONSTANT	<a href="#">DAY</a>

	unit of simulation time = 24 times HOUR
CONSTANT	<a href="#">PROCESS IMMEDIATELY</a>  Used to prioritize a process

**Function / Macro Summary**

Return Type	Summary
MACRO	<a href="#">ctoa</a>  like sprintf, prints a clocktype to a string
MACRO	<a href="#">atoc</a>  like atoi or atof, converts a string to a clocktype
MACRO	<a href="#">getSimStartTime(node)</a>  To get the simulation start time of a node
clocktype	<a href="#">TIME_ConvertToClock</a> (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	<a href="#">TIME_PrintClockInSecond</a> (clocktype clock, char * stringInSecond)  Print a clocktype value in second.The result is copied to string in Seconds
void	<a href="#">TIME_PrintClockInSecond</a> (clocktype clock, char * stringInSecond, Node * node)  Print a clocktype value in second.The result is copied to string in Seconds
void	<a href="#">TIME_PrintClockInSecond</a> (clocktype clock, char * stringInSecond, PartitionData * partition)  Print a clocktype value in second.The result is copied to string in Seconds
clocktype	<a href="#">TIME_ReturnMaxSimClock</a> (Node* node)  Return the maximum simulation clock

clocktype	<a href="#">TIME_ReturnStartSimClock</a> (Node* node)
Return the simulation start clock	

**Constant / Data Structure Detail**

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

Constant	<b>PROCESS_IMMEDIATELY 0</b>
Used to prioritize a process	

**Function / Macro Detail**

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

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



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

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

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



# EXata 5.1 API Reference

## COORDINATES

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

### Constant / Data Structure Summary

Type	Name
CONSTANT	<a href="#">PI</a> Defines the value of constant PI
CONSTANT	<a href="#">ANGLE_RESOLUTION</a> Defines ANGLE_RESOLUTION
CONSTANT	<a href="#">IN_RADIAN</a> Defines the constant IN_RADIAN
CONSTANT	<a href="#">EARTH_RADIUS</a> Defines the above constant EARTH_RADIUS
ENUMERATION	<a href="#">EarthRepresentationType</a> Defines the type of Earth that is represented Replaces coordinate_system_type
ENUMERATION	<a href="#">CoordinateRepresentationType</a> Defines the coordinate system that a coordinate is given in reference to
ENUMERATION	<a href="#">coordinate_system_type</a> Defines the type of coordinate system
STRUCT	<a href="#">cartesian_coordinates</a>

## COORDINATES

	Defines the three cartesian coordinates <a href="#"><u>latlonalt_coordinates</u></a>
STRUCT	Defines the three latlonalt coordinates <a href="#"><u>common_coordinates</u></a>
STRUCT	Defines the three common coordinates <a href="#"><u>Coordinates</u></a>
STRUCT	Defines coordinates <a href="#"><u>Orientation</u></a>
	Defines the orientation structure

## Function / Macro Summary

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

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

## COORDINATES

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

## Constant / Data Structure Detail

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

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

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

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

<p>Convert coordinate from JGIS to GEODETIC</p>	<ul style="list-style-type: none"> <li>• source - coordinate in JGIS</li> <li>• target - new coordinate in GEODETIC</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• static void - None</li> </ul>
<p><b>COORD_JGISToUnreferencedCartesian</b></p> <p>Convert coordinate from JGIS to UNREFERENCED_CARTESIAN</p>	<p>static void <b>COORD_JGISToUnreferencedCartesian</b> (const Coordinates* const source, Coordinates* const target)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• source - coordinate in JGIS</li> <li>• target - new coordinate in UNREFERENCED_CARTESIAN</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• static void - None</li> </ul>
<p><b>COORD_ConvertToCoordinates</b></p> <p>Read the string in "buf" and provide the corresponding coordinates for the string.</p>	<p>static void <b>COORD_ConvertToCoordinates</b> (char* buf, Coordinates* coordinates)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• buf - input string to be converted to coordinates</li> <li>• coordinates - Pointer to the coordinates</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• static void - None</li> </ul>
<p><b>COORD_MapCoordinateSystemToType</b></p> <p>Set coordinates type field (CoordinateRepresentationType) based on the user-provided coordinate system (coordinate_system_type)</p>	<p>static void <b>COORD_MapCoordinateSystemToType</b> (int coordinateSystem, Coordinates* coordinates)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• coordinateSystem - enum value indicating coordinate system</li> <li>• coordinates - Pointer to the coordinates</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• static void - None</li> </ul>
<p><b>COORD_NormalizeLongitude</b></p> <p>Correct the longitude value to between -180 and 180. This function assumes the coordinate system is LLA.</p>	<p>static void <b>COORD_NormalizeLongitude</b> (Coordinates* coordinates)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• coordinates - Pointer to the coordinates</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• static void - None</li> </ul>

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

## Returns:

- void - None



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

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

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



# EXata 5.1 API Reference

## ERROR

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

### Constant / Data Structure Summary

Type	Name
CONSTANT	<a href="#">ERROR_ASSERTION</a>  Defines the ERROR_ASSERTION constant
CONSTANT	<a href="#">ERROR_ERROR</a>  Defines the ERROR_ERROR constant
CONSTANT	<a href="#">ERROR_WARNING</a>  Defines the ERROR_WARNING constant

### Function / Macro Summary

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

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

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



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

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





# EXata 5.1 API Reference

## EXTERNAL

This file defines the generic interface to external modules.

### Constant / Data Structure Summary

Type	Name
CONSTANT	<a href="#">EXTERNAL_MAX_TIME</a>  The maximum possible time
CONSTANT	<a href="#">EXTERNAL_NUM_CPU_TIMING_INTERVAL_GUESSES</a>  The number of guesses to make for the cpu timing interval
CONSTANT	<a href="#">EXTERNAL_MAPPING_TABLE_SIZE</a>  The size of an interface's mapping hash table
CONSTANT	<a href="#">EXTERNAL_NUM_FUNCTIONS</a>  The number of functions an interface may implement
CONSTANT	<a href="#">EXTERNAL_RT_INDICATOR_INTERVAL</a>  The report interval of the realtime indication
CONSTANT	<a href="#">EXTERNAL_RT_INDICATOR_THRESHOLD</a>  red flag if the difference between realtime and the sim time is bigger than thread
ENUMERATION	<a href="#">ExternalInterfaceType</a>  Enumeration of different types of external interfaces
STRUCT	<a href="#">EXTERNAL_ThreadedMessage</a>

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

## Function / Macro Summary

Return Type	Summary
EXTERNAL_Interface *	<p><a href="#">EXTERNAL_RegisterExternalInterface</a>(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><a href="#">EXTERNAL_RegisterExternalInterface</a>(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	<a href="#">EXTERNAL_RegisterFunction</a> (EXTERNAL_Interface* iface, EXTERNAL_FunctionType type, EXTERNAL function)

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

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

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

	simulation horizon function will only be called if the current time (now) is >= that interface's current horizon.
void	<p><a href="#"><code>EXTERNAL_CallReceiveFunctions</code></a>(EXTERNAL_InterfaceList* list)</p> <p>This function will call all receive function that were not started in a thread</p>
void	<p><a href="#"><code>EXTERNAL_CallFinalizeFunctions</code></a>(EXTERNAL_InterfaceList* list)</p> <p>This function will call all finalize functions</p>
void	<p><a href="#"><code>EXTERNAL_InitializeExternalInterfaces</code></a>(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><a href="#"><code>EXTERNAL_PostInitialize</code></a>(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><a href="#"><code>EXTERNAL_GetExternalMessages</code></a>(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><a href="#"><code>EXTERNAL_Finalize</code></a>(partitionData* partitionData)</p> <p>Shuts down interfaces to external simulators</p>
void	<p><a href="#"><code>EXTERNAL_SetActive</code></a>(partitionData* partitionData)</p> <p>Sets isActive parameter based on interface registration</p>
void	<p><a href="#"><code>EXTERNAL_DeactivateInterface</code></a>(EXTERNAL_Interface* ifaceToDeactivate)</p> <p>Remove the indicated interface for the list of currently activated interfaces.</p>
void	<p><a href="#"><code>EXTERNAL_ProcessEvent</code></a>(Node* node, Message* msg)</p> <p>Process events meant for external code.</p>
clocktype	<p><a href="#"><code>GetNextInternalEventTime</code></a>(PartitionData* partitionData)</p> <p>Get the next internal event on the given partition. This includes both regular events and mobility events.</p>
void	<p><a href="#"><code>EXTERNAL_SendRtssNotification</code></a>(Node* node)</p>

To send Rtss notification over all active external interfaces that support Rtss

## Constant / Data Structure Detail

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

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

## Function / Macro Detail

Function / Macro	Format
<b>EXTERNAL_RegisterExternalInterface</b>	<p><b>EXTERNAL_Interface * EXTERNAL_RegisterExternalInterface</b> (EXTERNAL_InterfaceList* list, char* name, EXTERNAL_PerformanceParameters params)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>list - The list of external interfaces</li> <li>name - The name of the external interface.</li> <li>params - The performance parameters</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li><b>EXTERNAL_Interface * -</b> A pointer to the newly registered external interface</li> </ul>
<b>EXTERNAL_RegisterExternalInterface</b>	<b>EXTERNAL_Interface * EXTERNAL_RegisterExternalInterface</b> (EXTERNAL_InterfaceList* list, char* name, EXTERNAL_PerformanceParameters params, ExternalInterfaceType type)

<p>This function is an overloaded variation. for registering a new external interface with QualNet</p>	<p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• <code>list</code> - The list of external interfaces</li> <li>• <code>name</code> - The name of the external interface.</li> <li>• <code>params</code> - The performance parameters</li> <li>• <code>type</code> - PartitionData's interfaceTable will be</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>• <code>EXTERNAL_Interface *</code> - A pointer to the newly registered external interface</li> </ul>
<p><b>EXTERNAL_RegisterFunction</b></p> <p>Register a new function for an interface.</p>	<p><code>void EXTERNAL_RegisterFunction (EXTERNAL_Interface* iface, EXTERNAL_FunctionType type, EXTERNAL function)</code></p> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• <code>iface</code> - The external interface</li> <li>• <code>type</code> - the type of function</li> <li>• <code>function</code> - Function pointer to be called</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<p><b>EXTERNAL_SetTimeManagementRealTime</b></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>	<p><code>void EXTERNAL_SetTimeManagementRealTime (EXTERNAL_Interface* iface, clocktype lookahead)</code></p> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• <code>iface</code> - The external interface</li> <li>• <code>lookahead</code> - How far into the future the simulation is</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<p><b>EXTERNAL_ChangeRealTimeLookahead</b></p> <p>Modifies the lookahead value. Must be called after <code>EXTERNAL_SetTimeManagementRealTime()</code>. May be called during the simulation.</p>	<p><code>void EXTERNAL_ChangeRealTimeLookahead (EXTERNAL_Interface* iface, clocktype lookahead)</code></p> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• <code>iface</code> - The external interface</li> <li>• <code>lookahead</code> - The new lookahead value</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<p><b>EXTERNAL_InitializeWarmupParams</b></p>	<p><code>void EXTERNAL_InitializeWarmupParams (EXTERNAL_Interface* iface, NodeInput* nodeInput)</code></p>

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

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

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

	<ul style="list-style-type: none"> <li>• <code>forwardSize</code> - The size of the data to forward</li> <li>• <code>FwdReceiverOpt</code> - Whether to store the</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>EXTERNAL_RemoteForwardData</b>	<p>Send data back to the external source with no time stamp. This function is similar to <code>EXTERNAL_ForwardData</code>, except that this function can forward the message to an external interface on a different partition.</p> <p><code>void EXTERNAL_RemoteForwardData (EXTERNAL_Interface* iface, Node* node, void* forwardData, int forwardSize, int partitionId)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>iface</code> - The external interface</li> <li>• <code>node</code> - The node that is forwarding the data</li> <li>• <code>forwardData</code> - The data to forward</li> <li>• <code>forwardSize</code> - The size of the data to forward</li> <li>• <code>partitionId</code> - The partition Id to forward the message to</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>EXTERNAL_ForwardDataTimeStamped</b>	<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><code>void EXTERNAL_ForwardDataTimeStamped (EXTERNAL_Interface* iface, Node* node, Message* message, clocktype timestamp)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>iface</code> - The external interface</li> <li>• <code>node</code> - The node that is forwarding the data</li> <li>• <code>message</code> - The message</li> <li>• <code>timestamp</code> - The time stamp. This value is in external</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>EXTERNAL_UserFunctionRegistration</b>	<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><code>void EXTERNAL_UserFunctionRegistration (EXTERNAL_InterfaceList * list, NodeInput* nodeInput)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>list</code> - The list of external interfaces</li> <li>• <code>nodeInput</code> - The configuration file</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>

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

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

<p>This function will start the receive/forward threads for all threaded interfaces. Called after EXTERNAL_CallInitializeNodesFunctions.</p>	<ul style="list-style-type: none"> <li>• <code>list</code> - The list of external interfaces</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<p><b>EXTERNAL_CalculateMinSimulationHorizon</b></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 <math>\geq</math> that interface's current horizon.</p>	<p>clocktype <b>EXTERNAL_CalculateMinSimulationHorizon</b> (EXTERNAL_InterfaceList* <code>list</code>, clocktype <code>now</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>list</code> - The list of external interfaces</li> <li>• <code>now</code> - The current time</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>clocktype</code> - The minimum Simulation Horizon, or EXTERNAL_MAX_TIME if no horizon.</li> </ul>
<p><b>EXTERNAL_CallReceiveFunctions</b></p> <p>This function will call all receive function that were not started in a thread</p>	<p>void <b>EXTERNAL_CallReceiveFunctions</b> (EXTERNAL_InterfaceList* <code>list</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>list</code> - The list of external interfaces</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<p><b>EXTERNAL_CallFinalizeFunctions</b></p> <p>This function will call all finalize functions</p>	<p>void <b>EXTERNAL_CallFinalizeFunctions</b> (EXTERNAL_InterfaceList* <code>list</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>list</code> - The list of external interfaces</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<p><b>EXTERNAL_InitializeExternalInterfaces</b></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 <b>EXTERNAL_InitializeExternalInterfaces</b> (partitionData* <code>partitionData</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>partitionData</code> - pointer to data for this partition</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<p><b>EXTERNAL_PostInitialize</b></p> <p>Function used to initialize a generic interface to an external source of messages, e.g. an HLA</p>	<p>void <b>EXTERNAL_PostInitialize</b> (partitionData* <code>partitionData</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>partitionData</code> - pointer to data for this partition</li> </ul> <p>Returns:</p>

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

	<ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>GetNextInternalEventTime</b>	<p>clocktype <b>GetNextInternalEventTime</b> (PartitionData* partitionData)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• partitionData - Pointer to the partition</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• clocktype - The next internal event</li> </ul>
<b>EXTERNAL_SendRtssNotification</b>	<p>void <b>EXTERNAL_SendRtssNotification</b> (Node* node)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Node pointer.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - NULL</li> </ul>



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

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

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



# EXata 5.1 API Reference

## EXTERNAL\_SOCKET

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

### Constant / Data Structure Summary

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

### Function / Macro Summary

Return Type	Summary
void	<a href="#">EXTERNAL_VarArrayInit</a> (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.

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

EXTERNAL_SocketErrorType	<a href="#">EXTERNAL_SocketInitListen</a> (EXTERNAL_Socket* listenSocket, int port)
	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.
EXTERNAL_SocketErrorType	<a href="#">EXTERNAL_SocketAccept</a> (EXTERNAL_Socket* listenSocket, EXTERNAL_Socket* connectSocket)
	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.
EXTERNAL_SocketErrorType	<a href="#">EXTERNAL_SocketDataAvailable</a> (EXTERNAL_Socket* s, bool* available)
	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.
EXTERNAL_SocketErrorType	<a href="#">EXTERNAL_SocketConnect</a> (EXTERNAL_Socket* socket, char* address, int port, int maxAttempts)
	Connect to a listening socket. The socket is set to non-blocking mode.
EXTERNAL_SocketErrorType	<a href="#">EXTERNAL_SocketSend</a> (EXTERNAL_Socket* socket, char* data, unsigned int size, bool block)
	Send data on a connected socket. Since the socket is non-blocking, it is possible that the send would result in a block: If the "block" parameter is FALSE, then EXTERNAL_DataNotSent is returned, and no data is sent.
EXTERNAL_SocketErrorType	<a href="#">EXTERNAL_SocketSend</a> (EXTERNAL_Socket* socket, EXTERNAL_VarArray* data, bool block)
	This is a wrapper for the above overloaded function.
EXTERNAL_SocketErrorType	<a href="#">EXTERNAL_SocketRecv</a> (EXTERNAL_Socket* socket, char* data, unsigned int size, unsigned int* size, bool block)
	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.
EXTERNAL_SocketErrorType	<a href="#">EXTERNAL_SocketRecv</a> (EXTERNAL_Socket* socket, char* data, unsigned int size, unsigned int* size, int* ip, int* port, bool block)
	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.
EXTERNAL_SocketErrorType	<a href="#">EXTERNAL_SocketRecv</a> (EXTERNAL_Socket* socket, std data)
	Receive data on a connected socket. Continues reading until a '\n' character is found. This function always blocks.
EXTERNAL_SocketErrorType	<a href="#">EXTERNAL_SocketClose</a> (EXTERNAL_Socket* socket)

Close a socket. Must be called for each socket that is listening or connected.

### Constant / Data Structure Detail

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

### Function / Macro Detail

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

<b>EXTERNAL_VarArrayAccomodateSize</b>	<p>This function will increase the maximum size of the VarArray so that it can contain at least "size" bytes.</p> <p><b>void EXTERNAL_VarArrayAccomodateSize (EXTERNAL_VarArray* array, unsigned int size)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <b>array</b> - Pointer to the VarArray</li> <li>• <b>size</b> - The new minimum size of the VarArray</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <b>void</b> - None</li> </ul>
<b>EXTERNAL_VarArrayAppendData</b>	<p>This function will add data to the end of the VarArray. The size of the VarArray will be increased if necessary.</p> <p><b>void EXTERNAL_VarArrayAppendData (EXTERNAL_VarArray* array, char* data, unsigned int size)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <b>array</b> - Pointer to the VarArray</li> <li>• <b>data</b> - Pointer to the data to add</li> <li>• <b>size</b> - The size of the data to add</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <b>void</b> - None</li> </ul>
<b>EXTERNAL_VarArrayConcatString</b>	<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><b>void EXTERNAL_VarArrayConcatString (EXTERNAL_VarArray* array, char* string)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <b>array</b> - Pointer to the VarArray</li> <li>• <b>string</b> - The string</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <b>void</b> - None</li> </ul>
<b>EXTERNAL_VarArrayFree</b>	<p>This function will free all memory allocated to the VarArray</p> <p><b>void EXTERNAL_VarArrayFree (EXTERNAL_VarArray* array)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <b>array</b> - Pointer to the VarArray</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <b>void</b> - None</li> </ul>
<b>EXTERNAL_hton</b>	<p>Convert data from host byte order to network</p> <p><b>void EXTERNAL_hton (void* ptr, unsigned size)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <b>ptr</b> - Pointer to the data</li> </ul>

byte order	<ul style="list-style-type: none"> <li>• <code>size</code> - Size of the data</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>EXTERNAL_ntoh</b>	<p>Convert data from network byte order to host byte order</p> <p><code>void EXTERNAL_ntoh (void* ptr, unsigned size)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>ptr</code> - Pointer to the data</li> <li>• <code>size</code> - Size of the data</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>EXTERNAL_swapBitfield</b>	<p><code>void EXTERNAL_swapBitfield (void* ptr, unsigned size)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>ptr</code> - Pointer to the data</li> <li>• <code>size</code> - Size of the data (in bytes)</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>EXTERNAL_SocketInit</b>	<p>Initialize a socket. Must be called before all other socket API calls on the individual socket.</p> <p><code>void EXTERNAL_SocketInit (EXTERNAL_Socket* socket)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>socket</code> - Pointer to the socket</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>EXTERNAL_SocketInitUDP</b>	<p>Initialize a UDP socket. Must be called before all other socket API calls on the individual socket.</p> <p><code>EXTERNAL_SocketErrorType EXTERNAL_SocketInitUDP (EXTERNAL_Socket* socket)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>socket</code> - Pointer to the socket</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>EXTERNAL_SocketErrorType</code> - Error type</li> </ul>
<b>EXTERNAL_SocketValid</b>	<p><code>bool EXTERNAL_SocketValid (EXTERNAL_Socket* socket)</code></p> <p>Parameters:</p>

	<p>Check if a socket connection is valid and no errors have occurred.</p> <ul style="list-style-type: none"> <li>• <code>socket</code> - Pointer to the socket</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>bool</code> - true if valid, FALSE if closed or errors</li> </ul>
<b>EXTERNAL_SocketListen</b>	<p>Listen and accept a connections on a socket. This function is a wrapper for EXTERNAL_SocketInitListen and EXTERNAL_SocketAccept.</p> <p>EXTERNAL_SocketErrorType <b>EXTERNAL_SocketListen</b> (EXTERNAL_Socket* listenSocket, int port, EXTERNAL_Socket* connectSocket)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>listenSocket</code> - Pointer to the socket to listen on</li> <li>• <code>port</code> - The port to listen on</li> <li>• <code>connectSocket</code> - Pointer to the socket that will</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>EXTERNAL_SocketErrorType</code> - EXTERNAL_NoSocketError if successful, different error if not successful which could be due to a number of reasons.</li> </ul>
<b>EXTERNAL_SocketInitListen</b>	<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 <b>EXTERNAL_SocketInitListen</b> (EXTERNAL_Socket* listenSocket, int port)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>listenSocket</code> - Pointer to the socket to listen on</li> <li>• <code>port</code> - The port to listen on</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>EXTERNAL_SocketErrorType</code> - EXTERNAL_NoSocketError if successful, different error if not successful which could be due to a number of reasons.</li> </ul>
<b>EXTERNAL_SocketAccept</b>	<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 <b>EXTERNAL_SocketAccept</b> (EXTERNAL_Socket* listenSocket, EXTERNAL_Socket* connectSocket)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>listenSocket</code> - Pointer to the socket to listen on.</li> <li>• <code>connectSocket</code> - Pointer to the newly created socket</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>EXTERNAL_SocketErrorType</code> - EXTERNAL_NoSocketError if successful, different error if not successful which could be due to a number of reasons.</li> </ul>
<b>EXTERNAL_SocketDataAvailable</b>	<p>Test if a socket has readable data. For a</p> <p>EXTERNAL_SocketErrorType <b>EXTERNAL_SocketDataAvailable</b> (EXTERNAL_Socket* s, bool* available)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>s</code> - Pointer to the socket</li> </ul>

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

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

found. This function always blocks.	<ul style="list-style-type: none"> <li>• <code>data</code> - <code>string*</code></li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>EXTERNAL_SocketErrorType</code> - <code>EXTERNAL_NoSocketError</code> if successful, different error if not successful which could be due to a number of reasons.</li> </ul>
<b>EXTERNAL_SocketClose</b>  Close a socket. Must be called for each socket that is listening or connected.	<code>EXTERNAL_SocketErrorType EXTERNAL_SocketClose (EXTERNAL_Socket* socket)</code>  Parameters: <ul style="list-style-type: none"> <li>• <code>socket</code> - Pointer to the socket</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>EXTERNAL_SocketErrorType</code> - <code>EXTERNAL_NoSocketError</code> if successful, different error if not successful which could be due to a number of reasons.</li> </ul>



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

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

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



# EXata 5.1 API Reference

## EXTERNAL\_UTILITIES

This file describes utilities for external interfaces.

### Constant / Data Structure Summary

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

	A duration of simulation time <a href="#">EXTERNAL_TableOverflow</a>
STRUCT	A overflow record. <a href="#">EXTERNAL_Table</a>
STRUCT	A table. Generally used for storing external packet data, but can be used for anything. <a href="#">EXTERNAL_NetworkLayerPacket</a>
	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	<a href="#">EXTERNAL_TreeInitialize</a> (EXTERNAL_Tree* tree, BOOL useStore, int maxStore)  To initialize the splaytree
void	<a href="#">SCHED_SplayTreeInsert</a> (EXTERNAL_Tree* tree, EXTERNAL_TreeNode* treeNode)  To insert a node into the Splaytree
void	<a href="#">EXTERNAL_TreePeekMin</a> (EXTERNAL_Tree* tree)  To look up a node in the Splaytree
void	<a href="#">EXTERNAL_InitializeTable</a> (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	<a href="#">EXTERNAL_FinalizeTable</a> (EXTERNAL_Table* table)  This function will finalize the table
EXTERNAL_TableRecord*	<a href="#">EXTERNAL_GetUnusedRecord</a> (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*	<a href="#">EXTERNAL_GetEarliestRecord</a> (EXTERNAL_Table* table)

	This function will retrieve the earliest record in the table or NULL if the table is empty.
BOOL	<a href="#"><code>EXTERNAL_GetEarliestRecord</code></a> (EXTERNAL_Table* table, char* data)
EXTERNAL_TableRecord*	This function will check if a data pointer is still in the table. <a href="#"><code>EXTERNAL_FreeRecord</code></a> (EXTERNAL_Table* table)
void	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. <a href="#"><code>EXTERNAL_SendDataAppLayerUDP</code></a> (EXTERNAL_Interface* iface, NodeAddress from, NodeAddress to, char* data, int dataSize, clocktype timestamp, AppType app, TraceProtocolType trace, TosType priority)
void	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. <a href="#"><code>EXTERNAL_SendDataAppLayerUDP</code></a> (EXTERNAL_Interface* iface, NodeAddress from, NodeAddress to, char* header, int headerSize, int virtualDataSize, clocktype timestamp, AppType app, TraceProtocolType trace, TosType priority)
void	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. <a href="#"><code>EXTERNAL_SendDataAppLayerTCP</code></a> (EXTERNAL_Interface* iface, NodeAddress from, NodeAddress to, char* data, int dataSize, clocktype timestamp)
void	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. <a href="#"><code>EXTERNAL_SendDataNetworkLayer</code></a> (EXTERNAL_Interface* iface, NodeAddress from, NodeAddress srcAddr, NodeAddress destAddr, TosType tos, unsigned char protocol, unsigned int ttl, char* payload, int payloadSize, clocktype timestamp)
void	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. <a href="#"><code>EXTERNAL_SendDataNetworkLayerOnInterface</code></a> (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)
void	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. <a href="#"><code>EXTERNAL_SendVirtualDataNetworkLayer</code></a> (EXTERNAL_Interface* iface, NodeAddress from, NodeAddress srcAddr, NodeAddress destAddr, TosType tos, unsigned char protocol, unsigned int ttl, char* payload, int dataSize, int virtualSize, clocktype timestamp)

	Sends data originating from network layer. No provisions are made for handling this data once it enters the QualNet network. This is the responsibility of the external interface or protocols the data is sent to.
void	<a href="#">EXTERNAL_SendIpv6DataNetworkLayer</a> (EXTERNAL_Interface* iface, Address from, Address srcAddr, Address destAddr, TosType tos, unsigned char protocol, unsigned int hlim, char* payload, int payloadSize, clocktype timestamp)
	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.
void	<a href="#">EXTERNAL_SendDataNetworkLayer</a> (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)
	Sends data originating from network layer. No provisions are made for handling this data once it enters the QualNet network. This is the responsibility of the external interface or protocols the data is sent to.
void	<a href="#">EXTERNAL_SendNetworkLayerPacket</a> (Node* node, Message* msg)
	Sends the packet from EXTERNAL_SendDataNetworkLayer after some delay. This function should never be called directly.
void	<a href="#">EXTERNAL_CreateMapping</a> (EXTERNAL_Interface* iface, char* key, int keySize, char* value, int valueSize)
	Creates a mapping between a key and a value. The key may be any value and any length, such as an IP address, a MAC address, or a generic string. The value may be anything and is the responsibility of the user. Memory will be allocated for the key and the value.
int	<a href="#">EXTERNAL_ResolveMapping</a> (EXTERNAL_Interface* iface, char* key, int keySize, char** value, int* valueSize)
	Resolves a mapping created by EXTERNAL_CreateMapping. If it exists it is placed in the value and valueSize parameters and returns 0. The returned value will point to the memory block allocated by EXTERNAL_CreateMapping. If it does not exist it returns non-zero and the value and valueSize parameters are invalid.
int	<a href="#">EXTERNAL_DeleteMapping</a> (EXTERNAL_Interface* iface, char* key, int keySize)
	Deletes a mapping created by EXTERNAL_CreateMapping.
void	<a href="#">EXTERNAL_ActivateNode</a> (EXTERNAL_Interface* iface, Node* node)
	Activate a node so that it can begin processing events.
void	<a href="#">EXTERNAL_DectivateNode</a> (EXTERNAL_Interface* iface, Node* node)
	Deactivate a node so that it stops processing events.
void	<a href="#">EXTERNAL_PHY_SetTxPower</a> (Node* node, int phyIndex, double newTxPower)
	Just like PHY_SetTxPower (), but able to handle setting transmission power when node is owned by a remote partition. Change to

	TxPower will be scheduled as "best-effort" for remote nodes. The range of coordinate values depends on the terrain data.  <a href="#">EXTERNAL_PHY_GetTxPower</a> (Node* node, int phyIndex, double * txPowerPtr)
void	Just like PHY_GetTxPower (), but able to handle getting transmission power when node is owned by a remote partition.
void	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.  <a href="#">EXTERNAL_ChangeNodePosition</a> (EXTERNAL_Interface* iface, Node* node, double c1, double c2, double c3)
void	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.  <a href="#">EXTERNAL_ChangeNodeOrientation</a> (EXTERNAL_Interface* iface, Node* node, float azimuth, float elevation)
void	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.  <a href="#">EXTERNAL_ChangeNodePositionAndOrientation</a> (EXTERNAL_Interface* iface, Node* node, double c1, double c2, double c3, float azimuth, float elevation)
void	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.  <a href="#">EXTERNAL_ChangeNodePositionOrientationAndSpeedAtTime</a> (EXTERNAL_Interface* iface, Node* node, clocktype mobilityEventTime, double c1, double c2, double c3, float azimuth, float elevation, double speed)
void	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.  <a href="#">EXTERNAL_ChangeNodePositionOrientationAndVelocityAtTime</a> (EXTERNAL_Interface* iface, Node* node, clocktype mobilityEventTime, double c1, double c2, double c3, float azimuth, float elevation, double speed, double c1Speed, double c2Speed, double c3Speed)
void	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.  <a href="#">EXTERNAL_ChangeNodePositionOrientationAndVelocityAtTime</a> (EXTERNAL_Interface* iface, Node* node, clocktype mobilityEventTime, double c1, double c2, double c3, float azimuth, float elevation, double c1Speed, double c2Speed, double c3Speed)
void	<a href="#">EXTERNAL_ChangeNodeVelocityAtTime</a> (EXTERNAL_Interface* iface, Node* node, clocktype mobilityEventTime, double speed, double c1Speed, double c2Speed, double c3Speed)

	<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><a href="#"><b>EXTERNAL_ChangeNodeVelocityAtTime</b></a>(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><a href="#"><b>EXTERNAL_ConfigStringPresent</b></a>(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><a href="#"><b>EXTERNAL_ConfigStringIsYes</b></a>(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><a href="#"><b>EXTERNAL_MESSAGE_RemoteSend</b></a>(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 you run your simulation. Further notes about scheduling. If your external event won't result in additional qualnet events, except those that will be scheduled after safe time, then you can use LOOSELY. If, your event is going to schedule additional qualnet event though, then you must use EXTERNAL_SCHEDULE_SAFE (so that the event is delayed to the next safe time). If you violate safe time you will get assertion failures for safe time of signal receive time.</p>
void	<p><a href="#"><b>EXTERNAL_SetSimulationEndTime</b></a>(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><a href="#"><b>EXTERNAL_QueryRealTime</b></a>()</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><a href="#"><b>EXTERNAL_QueryRealTime</b></a>()</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	<a href="#">EXTERNAL_QueryCPUTime</a> (EXTERNAL_Interface* iface)
	This function will return the amount of Cpu time used by QualNet. The first call to this function will be 0, and timing will begin from that point.
void	<a href="#">EXTERNAL_Sleep</a> (clocktype amount)
	This function will sleep for a minimum amount of time as indicated by the amount parameter. Depending on which platform it is called on the amount of time spent sleeping could be greater.
void	<a href="#">EXTERNAL_AddHdr</a> (Node* node, Message* sendMessage, int payloadSize, UInt8* src, TosType tos, unsigned char protocol, unsigned ttl)
	This function will create qualnet in6_addr and add ipv6 header to message

## Constant / Data Structure Detail

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

	Info field used for sending a UDP forward app
Structure	<b>EXTERNAL_TableRecord</b>  A record in the table. Contains a pointer value and a timestamp, as well as information for maintaining a linked list.
Structure	<b>EXTERNAL_SimulationDurationInfo</b>  A duration of simulation time
Structure	<b>EXTERNAL_TableOverflow</b>  A overflow record.
Structure	<b>EXTERNAL_Table</b>  A table. Generally used for storing external packet data, but can be used for anything.
Structure	<b>EXTERNAL_NetworkLayerPacket</b>  A packet that will be sent at the network layer. Created by <b>EXTERNAL_SendDataNetworkLayer</b> , sent by <b>EXTERNAL_SendNetworkLayerPacket</b>

**Function / Macro Detail**

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

	<ul style="list-style-type: none"> <li>• <code>treeNode</code> - Pointer to the splayNode</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>EXTERNAL_TreePeekMin</b>	<p>To look up a node in the Splaytree</p> <p><code>void EXTERNAL_TreePeekMin (EXTERNAL_Tree* tree)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>tree</code> - Pointer to the splaytree</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>EXTERNAL_InitializeTable</b>	<p>This function will initialize the table. The size parameter represents the number of records that will be allocated in one block.</p> <p><code>void EXTERNAL_InitializeTable (EXTERNAL_Table* table, int size)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>table</code> - The table</li> <li>• <code>size</code> - The size of the table</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>EXTERNAL_FinalizeTable</b>	<p>This function will finalize the table</p> <p><code>void EXTERNAL_FinalizeTable (EXTERNAL_Table* table)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>table</code> - The table</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>EXTERNAL_GetUnusedRecord</b>	<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><code>EXTERNAL_TableRecord* EXTERNAL_GetUnusedRecord (EXTERNAL_Table* table)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>table</code> - The table</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>EXTERNAL_TableRecord*</code> - The retrieved record</li> </ul>
<b>EXTERNAL_GetEarliestRecord</b>	<p>This function will retrieve the earliest record in the table or NULL if the table is empty.</p> <p><code>EXTERNAL_TableRecord* EXTERNAL_GetEarliestRecord (EXTERNAL_Table* table)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>table</code> - The table</li> </ul>

	<p>Returns:</p> <ul style="list-style-type: none"> <li>• EXTERNAL_TableRecord* - The retrieved record</li> </ul>
<b>EXTERNAL_GetEarliestRecord</b>	<p>BOOL <b>EXTERNAL_GetEarliestRecord</b> (EXTERNAL_Table* table, char* data)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• table - The table</li> <li>• data - The data to check for</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• BOOL - TRUE if it is in the table, FALSE if not</li> </ul>
<b>EXTERNAL_FreeRecord</b>	<p>EXTERNAL_TableRecord* <b>EXTERNAL_FreeRecord</b> (EXTERNAL_Table* table)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• table - The table</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• EXTERNAL_TableRecord* - The retrieved record</li> </ul>
<b>EXTERNAL_SendDataAppLayerUDP</b>	<p>void <b>EXTERNAL_SendDataAppLayerUDP</b> (EXTERNAL_Interface* iface, NodeAddress from, NodeAddress to, char* data, int dataSize, clocktype timestamp, AppType app, TraceProtocolType trace, TosType priority)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• iface - The external interface</li> <li>• from - The address of the sending node</li> <li>• to - The address of the receiving node</li> <li>• data - The data that is to be sent. This may be NULL if there</li> <li>• dataSize - The size of the data</li> <li>• timestamp - The time to send this message. Pass 0 to send</li> <li>• app - The application to send to, defaults to APP_FORWARD</li> <li>• trace - The trace protocol, defaults to TRACE_FORWARD</li> <li>• priority - The priority to send this message at</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>EXTERNAL_SendDataAppLayerUDP</b>	void <b>EXTERNAL_SendDataAppLayerUDP</b> (EXTERNAL_Interface* iface, NodeAddress from,

	<p>Sends virtual data originating from the app layer using UDP. When the packet reaches its destination it will call the forward function of the external interface, if it exists.</p> <p>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"> <li>• <code>iface</code> - The external interface</li> <li>• <code>from</code> - The address of the sending node</li> <li>• <code>to</code> - The address of the receiving node</li> <li>• <code>header</code> - The header that is to be sent.</li> <li>• <code>headerSize</code> - The size of the header</li> <li>• <code>virtualDataSize</code> - The size of the virtual data</li> <li>• <code>timestamp</code> - The time to send this message. Pass 0 to send</li> <li>• <code>app</code> - The application to send to, defaults to APP_FORWARD</li> <li>• <code>trace</code> - The trace protocol, defaults to TRACE_FORWARD</li> <li>• <code>priority</code> - The priority to send this message at. defaults to IPTOS_PREC_ROUTINE</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>EXTERNAL_SendDataAppLayerTCP</b>	<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 <b>EXTERNAL_SendDataAppLayerTCP</b> (EXTERNAL_Interface* iface, NodeAddress from, NodeAddress to, char* data, int dataSize, clocktype timestamp)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>iface</code> - The external interface</li> <li>• <code>from</code> - The address of the sending node</li> <li>• <code>to</code> - The address of the receiving node</li> <li>• <code>data</code> - The data that is to be sent. This may be NULL if there</li> <li>• <code>dataSize</code> - The size of the data</li> <li>• <code>timestamp</code> - The time to send this message. Pass 0 to send</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>EXTERNAL_SendDataNetworkLayer</b>	<p>Sends data originating from network layer. No provisions are made</p> <p>void <b>EXTERNAL_SendDataNetworkLayer</b> (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"> <li><code>payloadSize</code> - The size of the data</li> <li><code>interfaceIndex</code> - The interface index</li> <li><code>timestamp</code> - The time to send this packet. Pass 0 to send</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li><code>void</code> - None</li> </ul>
<b>EXTERNAL_SendVirtualDataNetworkLayer</b>	<p>Sends data originating from network layer. No provisions are made for handling this data once it enters the QualNet network. This is the responsibility of the external interface or protocols the data is sent to.</p> <p><code>void EXTERNAL_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)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li><code>iface</code> - The external interface</li> <li><code>from</code> - The address of the node that will send the</li> <li><code>srcAddr</code> - The IP address of the node originally</li> <li><code>destAddr</code> - The address of the receiving node</li> <li><code>tos</code> - The Type of Service field in the IP header</li> <li><code>protocol</code> - The protocol field in the IP header</li> <li><code>ttl</code> - The Time to Live field in the IP header</li> <li><code>payload</code> - The data that is to be sent. This should include</li> <li><code>dataSize</code> - The size of the data</li> <li><code>virtualSize</code> - The size of the virtual data</li> <li><code>timestamp</code> - The time to send this packet. Pass 0 to send</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li><code>void</code> - None</li> </ul>
<b>EXTERNAL_SendIpv6DataNetworkLayer</b>	<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><code>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)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li><code>iface</code> - The external interface</li> <li><code>from</code> - The address of the node that will send the</li> <li><code>srcAddr</code> - The IP address of the node originally</li> <li><code>destAddr</code> - The address of the receiving node</li> </ul>

	<ul style="list-style-type: none"> <li><code>tos</code> - The Type of Service field in the IPv6 header</li> <li><code>protocol</code> - The protocol field in the IPv6 header</li> <li><code>hlim</code> - The hop limit field in the IPv6 header</li> <li><code>payload</code> - The data that is to be sent. This should include</li> <li><code>payloadSize</code> - The size of the data</li> <li><code>timestamp</code> - The time to send this packet. Pass 0 to send</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li><code>void</code> - None</li> </ul>
<b>EXTERNAL_SendDataNetworkLayer</b>	<p>Sends data originating from network layer. No provisions are made for handling this data once it enters the QualNet network. This is the responsibility of the external interface or protocols the data is sent to.</p> <p><code>void EXTERNAL_SendDataNetworkLayer (EXTERNAL_Interface* iface, NodeAddress from, NodeAddress srcAddr, NodeAddress destAddr, 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)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li><code>iface</code> - The external interface</li> <li><code>from</code> - The address of the node that will send the</li> <li><code>srcAddr</code> - The IP address of the node originally</li> <li><code>destAddr</code> - The address of the receiving node</li> <li><code>identification</code> - The identification field in the IP</li> <li><code>dontFragment</code> - Whether to set the dont fragment bit in the IP</li> <li><code>moreFragments</code> - Whether to set the more fragments bit in the IP</li> <li><code>fragmentOffset</code> - The fragment offset field in the IP</li> <li><code>tos</code> - The Type of Service field in the IP header</li> <li><code>protocol</code> - The protocol field in the IP header</li> <li><code>ttl</code> - The Time to Live field in the IP header</li> <li><code>payload</code> - The data that is to be sent. This should include</li> <li><code>payloadSize</code> - The size of the data</li> <li><code>ipHeaderLength</code> - length of the IP Header including options if any</li> <li><code>ipOptions</code> - pointer to the IP Option.</li> <li><code>timestamp</code> - The time to send this packet. Pass 0 to send</li> </ul>

	<p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>EXTERNAL_SendNetworkLayerPacket</b>	<p>Sends the packet from EXTERNAL_SendDataNetworkLayer after some delay. This function should never be called directly.</p> <p><code>void EXTERNAL_SendNetworkLayerPacket (Node* node, Message* msg)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - The node sending the packet</li> <li>• <code>msg</code> - The message</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>EXTERNAL_CreateMapping</b>	<p>Creates a mapping between a key and a value. The key may be any value and any length, such as an IP address, a MAC address, or a generic string. The value may be anything and is the responsibility of the user. Memory will be allocated for the key and the value.</p> <p><code>void EXTERNAL_CreateMapping (EXTERNAL_Interface* iface, char* key, int keySize, char* value, int valueSize)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>iface</code> - The external interface</li> <li>• <code>key</code> - The address of the key</li> <li>• <code>keySize</code> - The size of the key in bytes</li> <li>• <code>value</code> - The address of what the value maps to</li> <li>• <code>valueSize</code> - The size of the value in bytes</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>EXTERNAL_ResolveMapping</b>	<p>Resolves a mapping created by EXTERNAL_CreateMapping. If it exists it is placed in the value and valueSize parameters and returns 0. The returned value will point to the memory block allocated by EXTERNAL_CreateMapping. If it does not exist it returns non-zero and the value and valueSize parameters are invalid.</p> <p><code>int EXTERNAL_ResolveMapping (EXTERNAL_Interface* iface, char* key, int keySize, char** value, int* valueSize)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>iface</code> - The external interface</li> <li>• <code>key</code> - Pointer to the key</li> <li>• <code>keySize</code> - The size of the key in bytes</li> <li>• <code>value</code> - Pointer to the value (output)</li> <li>• <code>valueSize</code> - The size of the key in bytes (output)</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>int</code> - 0 if the mapping resolved, non-zero if it did not</li> </ul>
<b>EXTERNAL_DeleteMapping</b>	<code>int EXTERNAL_DeleteMapping (EXTERNAL_Interface* iface, char* key, int keySize)</code>

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

	<ul style="list-style-type: none"> <li>• node - The node (can be either a local node or remote)</li> <li>• phyIndex - The physical index</li> <li>• txPowerPtr - (OUT) value of transmission power will be</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>EXTERNAL_ChangeNodePosition</b> <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><b>void EXTERNAL_ChangeNodePosition (EXTERNAL_Interface* iface, Node* node, double c1, double c2, double c3)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• iface - The external interface</li> <li>• node - The node</li> <li>• c1 - The first coordinate</li> <li>• c2 - The second coordinate</li> <li>• c3 - The third coordinate</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>EXTERNAL_ChangeNodeOrientation</b> <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><b>void EXTERNAL_ChangeNodeOrientation (EXTERNAL_Interface* iface, Node* node, float azimuth, float elevation)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• iface - The external interface</li> <li>• node - The node</li> <li>• azimuth - The azimuth, <math>0 \leqslant \text{azimuth} \leqslant 359</math></li> <li>• elevation - The elevation, <math>-180 \leqslant \text{elevation} \leqslant 180</math></li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>EXTERNAL_ChangeNodePositionAndOrientation</b> <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><b>void EXTERNAL_ChangeNodePositionAndOrientation (EXTERNAL_Interface* iface, Node* node, double c1, double c2, double c3, float azimuth, float elevation)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• iface - The external interface</li> <li>• node - The node</li> </ul>

	<ul style="list-style-type: none"> <li>• <code>c1</code> - The first coordinate</li> <li>• <code>c2</code> - The second coordinate</li> <li>• <code>c3</code> - The third coordinate</li> <li>• <code>azimuth</code> - The azimuth, <math>0 \leq \text{azimuth} \leq 359</math></li> <li>• <code>elevation</code> - The elevation, <math>-180 \leq \text{elevation} \leq 180</math></li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>EXTERNAL_ChangeNodePositionOrientationAndSpeedAtTime</b>	<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><code>void EXTERNAL_ChangeNodePositionOrientationAndSpeedAtTime (EXTERNAL_Interface* iface, Node* node, clocktype mobilityEventTime, double c1, double c2, double c3, float azimuth, float elevation, double speed)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>iface</code> - The external interface</li> <li>• <code>node</code> - The node</li> <li>• <code>mobilityEventTime</code> - The absolute simulation time (not delay)</li> <li>• <code>c1</code> - The first coordinate</li> <li>• <code>c2</code> - The second coordinate</li> <li>• <code>c3</code> - The third coordinate</li> <li>• <code>azimuth</code> - The azimuth, <math>0 \leq \text{azimuth} \leq 359</math></li> <li>• <code>elevation</code> - The elevation, <math>-180 \leq \text{elevation} \leq 180</math></li> <li>• <code>speed</code> - The speed in m/s</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>EXTERNAL_ChangeNodePositionOrientationAndVelocityAtTime</b>	<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><code>void EXTERNAL_ChangeNodePositionOrientationAndVelocityAtTime (EXTERNAL_Interface* iface, Node* node, clocktype mobilityEventTime, double c1, double c2, double c3, float azimuth, float elevation, double speed, double c1Speed, double c2Speed, double c3Speed)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>iface</code> - The external interface</li> <li>• <code>node</code> - The node</li> <li>• <code>mobilityEventTime</code> - The absolute simulation time (not delay)</li> <li>• <code>c1</code> - The first coordinate</li> </ul>

	<ul style="list-style-type: none"> <li>• <code>c2</code> - The second coordinate</li> <li>• <code>c3</code> - The third coordinate</li> <li>• <code>azimuth</code> - The azimuth, <math>0 \leq \text{azimuth} \leq 359</math></li> <li>• <code>elevation</code> - The elevation, <math>-180 \leq \text{elevation} \leq 180</math></li> <li>• <code>speed</code> - The speed in m/s</li> <li>• <code>c1Speed</code> - The rate of change of the first coordinate in the</li> <li>• <code>c2Speed</code> - The rate of change of the second coordinate in the</li> <li>• <code>c3Speed</code> - The rate of change of the third coordinate in the</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>EXTERNAL_ChangeNodePositionOrientationAndVelocityAtTime</b>	<p>void <b>EXTERNAL_ChangeNodePositionOrientationAndVelocityAtTime</b> (EXTERNAL_Interface* <code>iface</code>, Node* <code>node</code>, clocktype <code>mobilityEventTime</code>, double <code>c1</code>, double <code>c2</code>, double <code>c3</code>, float <code>azimuth</code>, float <code>elevation</code>, double <code>c1Speed</code>, double <code>c2Speed</code>, double <code>c3Speed</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>iface</code> - The external interface</li> <li>• <code>node</code> - The node</li> <li>• <code>mobilityEventTime</code> - The absolute simulation time (not delay)</li> <li>• <code>c1</code> - The first coordinate</li> <li>• <code>c2</code> - The second coordinate</li> <li>• <code>c3</code> - The third coordinate</li> <li>• <code>azimuth</code> - The azimuth, <math>0 \leq \text{azimuth} \leq 359</math></li> <li>• <code>elevation</code> - The elevation, <math>-180 \leq \text{elevation} \leq 180</math></li> <li>• <code>c1Speed</code> - The rate of change of the first coordinate in the</li> <li>• <code>c2Speed</code> - The rate of change of the second coordinate in the</li> <li>• <code>c3Speed</code> - The rate of change of the third coordinate in the</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>EXTERNAL_ChangeNodeVelocityAtTime</b>	<p>void <b>EXTERNAL_ChangeNodeVelocityAtTime</b> (EXTERNAL_Interface* <code>iface</code>, Node* <code>node</code>, clocktype <code>mobilityEventTime</code>, double <code>speed</code>, double <code>c1Speed</code>, double <code>c2Speed</code>, double <code>c3Speed</code>)</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><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• <code>iface</code> - The external interface</li> <li>• <code>node</code> - The node</li> <li>• <code>mobilityEventTime</code> - The absolute simulation time (not delay)</li> <li>• <code>speed</code> - The speed in m/s</li> <li>• <code>c1Speed</code> - The rate of change of the first coordinate in the</li> <li>• <code>c2Speed</code> - The rate of change of the second coordinate in the</li> <li>• <code>c3Speed</code> - The rate of change of the third coordinate in the</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<p><b>EXTERNAL_ChangeNodeVelocityAtTime</b></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><b>void EXTERNAL_ChangeNodeVelocityAtTime</b> (<code>EXTERNAL_Interface* iface</code>, <code>Node* node</code>, <code>clocktype mobilityEventTime</code>, <code>double c1Speed</code>, <code>double c2Speed</code>, <code>double c3Speed</code>)</p> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• <code>iface</code> - The external interface</li> <li>• <code>node</code> - The node</li> <li>• <code>mobilityEventTime</code> - The absolute simulation time (not delay)</li> <li>• <code>c1Speed</code> - The rate of change of the first coordinate in the</li> <li>• <code>c2Speed</code> - The rate of change of the second coordinate in the</li> <li>• <code>c3Speed</code> - The rate of change of the third coordinate in the</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<p><b>EXTERNAL_ConfigStringPresent</b></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><b>BOOL EXTERNAL_ConfigStringPresent</b> (<code>NodeInput* nodeInput</code>, <code>char* string</code>)</p> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• <code>nodeInput</code> - The configuration file</li> <li>• <code>string</code> - The string to check for</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>• <code>BOOL</code> - TRUE if the string is present, FALSE otherwise</li> </ul>
	<p><b>BOOL EXTERNAL_ConfigStringIsYes</b> (<code>NodeInput* nodeInput</code>, <code>char* 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><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• nodeInput - The configuration file</li> <li>• string - The string to check for</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>• BOOL - TRUE if the string is YES, FALSE otherwise</li> </ul>
<b>EXTERNAL_MESSAGE_RemoteSend</b>	<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 you run your simulation. Further notes about scheduling. If your external event won't result in additional qualnet events, except those that will be scheduled after safe time, then you can use LOOSELY. If, your event is going to schedule additional qualnet event though, then you must use EXTERNAL_SCHEDULE_SAFE (so that the event is delayed to the next safe time). If you violate safe time you will get assertion failures for safe time of signal receive time.</p> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• iface - Your external interface</li> <li>• destinationPartitionId - The partitionId that you want to send to</li> <li>• msg - The external message to send</li> <li>• delay - When the message should be scheduled on the remote partition.</li> <li>• scheduling - Whether this event can be executed lossely</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>EXTERNAL_SetSimulationEndTime</b>	<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> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• partitionData - pointer to data for this partition</li> <li>• endTime - The simulation time to end at.</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>EXTERNAL_QueryRealTime</b>	<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><b>Parameters:</b></p> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>• clocktype - The real time, not adjusted for simulation pauses.</li> </ul>

<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"> <li>• <code>clocktype</code> - The real time, adjusted for simulation pauses.</li> </ul>
<p><b>EXTERNAL_QueryCPUTime</b></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 EXTERNAL_QueryCPUTime (EXTERNAL_Interface* iface)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>iface</code> - The external interface</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>clocktype</code> - The CPU time</li> </ul>
<p><b>EXTERNAL_Sleep</b></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 EXTERNAL_Sleep (clocktype amount)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>amount</code> - The amount of time to sleep</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<p><b>EXTERNAL_AddHdr</b></p> <p>This function will create qualnet <code>in6_addr</code> and add ipv6 header to message</p>	<p><code>void EXTERNAL_AddHdr (Node* node, Message* sendMessage, int payloadSize, UInt8* src, TosType tos, unsigned char protocol, unsigned ttl)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Node pointer</li> <li>• <code>sendMessage</code> - Message on which ipv6 header needs to be added</li> <li>• <code>payloadSize</code> - Size of payload in packet</li> <li>• <code>src</code> - IPv6 Source address</li> <li>• <code>tos</code> - Packet Priority</li> <li>• <code>protocol</code> - Protocol after ipv6 header</li> <li>• <code>ttl</code> - Hop limit</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>



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

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

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



# EXata 5.1 API Reference

## FILEIO

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

### Constant / Data Structure Summary

Type	Name
CONSTANT	<a href="#">ANY_NODEID</a>
	Optional macro values to use when calling IO_Read...() APIs. Defines any node id.
CONSTANT	<a href="#">ANY_ADDRESS</a>
	Optional macro values to use when calling IO_Read...() APIs. Defines any node address.
CONSTANT	<a href="#">ANY_INSTANCE</a>
	Optional macro values to use when calling IO_Read...() APIs. Defines any instance.
CONSTANT	<a href="#">MAX_INPUT_FILE_LINE_LENGTH</a>
	Maximum input file line length. Evaluates (6 * MAX_STRING_LENGTH)
CONSTANT	<a href="#">MAX_ADDRESS_STRING_LENGTH</a>
	Maximum length of address string.
CONSTANT	<a href="#">MAX_NUM_CACHED_FILES</a>
	Max number of -FILE references in an input file. (Restriction is only for immediate children)
CONSTANT	<a href="#">MATCH_GLOBAL</a>
	Defines the matching at global level
CONSTANT	<a href="#">MATCH_NODE_ID</a>

	Defines the matching by node id.
CONSTANT	<a href="#">MATCH_NETWORK</a>
	Defines the matching by network.
CONSTANT	<a href="#">MATCH_INTERFACE</a>
	Defines the matching by interface.
CONSTANT	<a href="#">INPUT_ALLOCATION_UNIT</a>
	Defines input allocation unit.
STRUCT	<a href="#">NodeInput</a>
	Definition of node input structure. typedef to NodeInput in include/main.h.

## Function / Macro Summary

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

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

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

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

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

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

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

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

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

	<p>Print out the relevant stat in "buf", along with the node id and the layer type generating this stat. Overloaded API for Ipv6 compatibility.</p>
void	<p><a href="#">IO_ParseNodeIdHostOrNetworkAddress</a>(const char s[], in6_addr* ipAddress, BOOL* isIpAddress, NodeId* nodeId)</p>
	<p>Parses a string for a nodeId, host address, or network address. Overloaded API for Ipv6 compatibility.</p>
void	<p><a href="#">IO_ParseNodeIdHostOrNetworkAddress</a>(const char s[], ATM addr* atmAddress, BOOL* isAtmAddr, NodeId* nodeId)</p>
	<p>Parses a string for a nodeId, host address, or network address. Overloaded API for Ipv6 compatibility.</p>
void	<p><a href="#">IO_ParseNodeIdOrHostAddress</a>(const char s[], in6_addr* outputNodeAddress, BOOL* isNodeId)</p>
	<p>Parses a string for a nodeId or host address.</p>
void	<p><a href="#">IO_ParseNetworkAddress</a>(const char s[], unsigned int* tla, unsigned int* nla, unsigned int* sla)</p>
	<p>Parses a string for a network address. Overloaded API for Ipv6 compatibility.</p>
void	<p><a href="#">IO_AppParseSourceAndDestStrings</a>(Node* node, const char* inputString, const char* sourceString, NodeId* sourceNodeId, Address* sourceAddr, const char* destString, NodeId* destNodeId, Address* destAddr)</p>
	<p>Application input parsing API. Parses the source and destination strings. Overloaded for Ipv6 compatibility.</p>
void	<p><a href="#">IO_AppParseSourceString</a>(Node* node, const char* inputString, const char* sourceString, NodeAddress* sourceNodeId, Address* sourceAddr, NetworkType networkType)</p>
	<p>Application input parsing API. Parses the source string. Overloaded for Ipv6 compatibility.</p>
void	<p><a href="#">IO_AppParseDestString</a>(Node* node, const char* inputString, const char* destString, NodeAddress* destNodeId, Address* destAddr, NetworkType networkType)</p>
	<p>Application input parsing API. Parses the destination string. Overloaded for Ipv6 compatibility.</p>
BOOL	<p><a href="#">QualifierMatches</a>(const NodeId nodeId, const in6_addr interfaceAddress, const char* qualifier, int* matchType)</p>
	<p>This is an auxiliary API used by the IO_Read...() set of APIs. Overloaded for Ipv6 compatibility</p>
BOOL	<p><a href="#">QualifierMatches</a>(const NodeId nodeId, const ATMAddress* interfaceAddress, const char* qualifier, int* matchType)</p>
	<p>This is an auxiliary API used by the IO_Read...() set of APIs. Overloaded for Ipv6 compatibility</p>
void	<p><a href="#">IO_ConvertIpv6StringToAddress()</a>(char* interfaceAddr, in6_addr* ipAddress)</p>
	<p>Convert IPv6 address string to in6_addr structure. API for Ipv6 compatibility.</p>
void	<p><a href="#">IO_ConvertIpAddressToString</a>(in6_addr* ipAddress, char* interfaceAddr)</p>

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

	Convert generic address to appropriate network type address string format. <a href="#"><code>IO_ConvertAtmAddressToString</code></a> (AtmAddress addr, char* addrStr)
void	Convert Atm address to address string format. <a href="#"><code>IO_InsertIntValue</code></a> (const char s[], const unsigned int val, unsigned int u_atmVal)
	Insert integer value for specific string in case of ATM
int	<a href="#"><code>IO_ReadCachedFileIndex</code></a> (NodeAddress nodeId, NodeAddress interfaceAddress, unsigned int nodeInput)
	Return Cached file index for the given parameter name
void	<a href="#"><code>IO_ReadString</code></a> (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	<a href="#"><code>IO_ReadInt64</code></a> (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	<a href="#"><code>IO_ReadTime</code></a> (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	<a href="#"><code>IO_ReadInt</code></a> (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	<a href="#"><code>IO_ReadDouble</code></a> (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	<a href="#"><code>IO_ReadCachedFile</code></a> (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	<a href="#"><code>IO_ReadLine</code></a> (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	<a href="#"><code>IO_ReadStringInstance</code></a> (Node* node, const NodeId nodeId, int interfaceIndex, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, char* parameterValue)
	This API is used to retrieve string parameter values from input files for a specific instance.
void	<a href="#"><code>IO_ReadDoubleInstance</code></a> (Node* node, const NodeId nodeId, int interfaceIndex, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, double* parameterValue)
	This API is used to retrieve double parameter values from input files for a specific instance.
void	<a href="#"><code>IO_ReadIntInstance</code></a> (Node* node, const NodeId nodeId, int interfaceIndex, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, int* parameterValue)
	This API is used to retrieve int parameter values from input files for a specific instance.
void	<a href="#"><code>IO_ReadTimeInstance</code></a> (Node* node, const NodeId nodeId, int interfaceIndex, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, clocktype* parameterValue)
	This API is used to retrieve clocktype parameter values from input files for a specific instance.
void	<a href="#"><code>IO_ReadCachedFileInstance</code></a> (Node* node, const NodeId nodeId, int interfaceIndex, const NodeInput* nodeInput, const char* parameterName, const int parameterInstanceNumber, const BOOL fallbackIfNoInstanceMatch, BOOL* wasFound, NodeInput* parameterValue)
	This API is used to retrieve cached file parameter values from input files for a specific instance.
void	<a href="#"><code>IO_ReadStringUsingIpAddress</code></a> (Node* node, int interfaceIndex, const NodeInput* nodeInput, const char* parameterName, BOOL* wasFound, char* parameterValue)
	This API is used to retrieve a string parameter value from input files using the ip-address.
void	<a href="#"><code>IO_ReadString</code></a> (const NodeAddress nodeId, NodeAddress ipv4Address, in6_addr* ipv6Address, const NodeInput* nodeInput, const char* parameterName, BOOL* wasFound, char* parameterValue)
	This API is used to retrieve a string parameter value from input files.
void	<a href="#"><code>IO_ReadString</code></a> (const NodeAddress nodeId, int interfaceIndex, const NodeAddress ipv4SubnetAddress, const in6_addr* ipv6SubnetAddress, const NodeInput* nodeInput, const char* parameterName, char* parameterValue, BOOL& wasFound, int& matchType)
	This API is used to retrieve a string parameter value from input files.
void	<a href="#"><code>IO_ReadChannelMask</code></a> (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)

	This API is used to retrieve the value channel mask for a specific instance.
void	<code><a href="#">IO_ReadBool</a>(Node* node, const NodeId nodeId, int interfaceIndex, const NodeInput* nodeInput, const char* parameterName, BOOL* wasFound, BOOL* parameterValue)</code>
void	<p>This API is used to retrieve a boolean parameter value from input files.</p> <p><code><a href="#">IO_ReadFloat</a>(Node* node, const NodeAddress nodeId, int interfaceIndex, const NodeInput* nodeInput, const char* parameterName, BOOL* wasFound, float* parameterValue)</code></p> <p>This API is used to retrieve a float parameter value from input files.</p>

## Constant / Data Structure Detail

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

	Defines the matching at global level
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
<b>IO_ConvertIpAddressToString</b>  Parses IPv4 address into a dotted-decimal string.	void <b>IO_ConvertIpAddressToString</b> (NodeAddress ipAddress, char* addressString)  Parameters: <ul style="list-style-type: none"><li>• ipAddress - IPv4 address to be converted into</li><li>• addressString - Storage for string.</li></ul> Returns: <ul style="list-style-type: none"><li>• void - None</li></ul>
<b>IO_FindStringPos</b>  Returns the index of the first subString found in s.	int <b>IO_FindStringPos</b> (const char s[], const char subString[])  Parameters: <ul style="list-style-type: none"><li>• s[] - Source string.</li><li>• subString[] - Substring to search for.</li></ul>

	<p>Returns:</p> <ul style="list-style-type: none"> <li>• int - Index of the first subString found in s. -1, if not found.</li> </ul>
<b>IO_GetToken</b>	<p>char* <b>IO_GetToken</b> (char* dst, const char* src, char ** next)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• dst - Buffer to copy token too. If passed in as</li> <li>• src - Source string.</li> <li>• next - Storage for pointer to remainder of string.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• char* - dst, if string was found and dst was passed in as non-NULL. Pointer to token in src, if string was found and dst was passed in as NULL. NULL, otherwise.</li> </ul>
<b>IO_GetDelimitedToken</b>	<p>char* <b>IO_GetDelimitedToken</b> (char* dst, const char* src, const char* delim, char** next)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• dst - Buffer to copy token too. If passed in as</li> <li>• src - Source string.</li> <li>• delim - Delimiter string.</li> <li>• next - Storage for pointer to remainder of string.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• char* - dst, if string was found and dst was passed in as non-NULL. Pointer to token in src, if string was found and dst was passed in as NULL. NULL, otherwise.</li> </ul>
<b>IO_Right</b>	<p>const char* <b>IO_Right</b> (const char * s, unsigned count)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• s - String.</li> <li>• count - Number of characters on the right side.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• const char* - 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.</li> </ul>
<b>IO_Chop</b>	<p>char* <b>IO_Chop</b> (const char* s)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• s - String.</li> </ul> <p>Removes the last character of string.</p>

	<p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>char* - s</code>. If the string has a <code>strlen()</code> of zero, then the string is returned unmodified. A "character" is just a byte of char type that's not NULL. So, '\n' counts as a character.</li> </ul>
<b>IO_TrimNsbpSpaces</b>	<p><code>void IO_TrimNsbpSpaces (char* s)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>s - String.</code></li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void - None</code></li> </ul>
<b>IO_TrimLeft</b>	<p><code>void IO_TrimLeft (char* s)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>s - String.</code></li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void - None</code></li> </ul>
<b>IO_TrimRight</b>	<p><code>void IO_TrimRight (char* s)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>s - String.</code></li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void - None</code></li> </ul>
<b>IO_CompressWhiteSpace</b>	<p><code>void IO_CompressWhiteSpace (char* s)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>s - String.</code></li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void - None</code></li> </ul>
<b>IO_IsStringNonNegativeInteger</b>	<p><code>BOOL IO_IsStringNonNegativeInteger (const char* s)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>s - String.</code></li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>BOOL</code></li> </ul>

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

<b>IO_FindCaseInsensitiveStringPos</b>	<p>Finds the case insensitive sub string position in a string.</p> <p><b>int IO_FindCaseInsensitiveStringPos (const char s[], const char subString[])</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>s[]</code> - String.</li> <li>• <code>subString[]</code> - Sub string</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>int</code> - Returns the position of case insensitive sub string if found. -1, otherwise.</li> </ul>
<b>IO_FindCaseInsensitiveStringPos</b>	<p>Finds the case insensitive sub string position in a string.</p> <p><b>int IO_FindCaseInsensitiveStringPos (const char s[], const char subString[])</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>s[]</code> - String.</li> <li>• <code>subString[]</code> - Sub string</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>int</code> - Returns the position of case insensitive sub string if found. -1, otherwise.</li> </ul>
<b>IO_SkipToken.</b>	<p>skip the first n tokens.</p> <p><b>IO_SkipToken. (char* token, char* tokenSep, char* skip)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>token</code> - pointer to the input string,</li> <li>• <code>tokenSep</code> - pointer to the token separators,</li> <li>• <code>skip</code> - number of skips.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• -</li> </ul>
<b>IO_CreateNodeInput</b>	<p>Allocates a NodeInput datastructure that can then be passed to IO_ReadNodeInput Called for each file variable in the config file.</p> <p><b>NodeInput * IO_CreateNodeInput (NodeInput* nodeInput, const char* filename)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>nodeInput</code> - Pointer to node input.</li> <li>• <code>filename</code> - Path to input file.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>NodeInput * - None</code></li> </ul>
<b>IO_InitializeNodeInput</b>	<p><b>void IO_InitializeNodeInput (NodeInput* nodeInput)</b></p> <p>Parameters:</p>

Initializes a NodeInput structure	<ul style="list-style-type: none"> <li>• <code>nodeInput</code> - A pointer to NodeInput structure.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>IO_ReadNodeInput</b>  Reads an input file into a NodeInput struct. Calls IO_ReadFileParameters to first read in -FILE paramters. Then calls IO_ReadFileParameters to read the rest of the parameters.	<p><code>void IO_ReadNodeInput (NodeInput* nodeInput, const char* filename)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>nodeInput</code> - Pointer to node input.</li> <li>• <code>filename</code> - Path to input file.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>IO_ReadNodeInputEx</b>  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.	<p><code>void IO_ReadNodeInputEx (NodeInput* nodeInput, const char* filename, const char* includeComment)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>nodeInput</code> - Pointer to node input.</li> <li>• <code>filename</code> - Path to input file.</li> <li>• <code>includeComment</code> - When this flag is true it</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>IO_ConvertFile</b>  Converts the contents of an old configuration file to the latest version.	<p><code>BOOL IO_ConvertFile (NodeInput* nodeInput, NodeInput* nodeOutput, char* version)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>nodeInput</code> - A pointer to node input.</li> <li>• <code>nodeOutput</code> - A pointer to node input. Goes through</li> <li>• <code>version</code> - Not used.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>BOOL</code> - 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.</li> </ul>
<b>IO_ReadLine</b>	<p><code>void IO_ReadLine (const NodeAddress nodeId, const NodeAddress interfaceAddress, const NodeInput* nodeInput, const char* index, BOOL* wasFound, char* readVal)</code></p> <p>Parameters:</p>

<p>This API is used to retrieve a whole line from input files.</p>	<ul style="list-style-type: none"> <li>• nodeId - nodeId. Can be ANY_NODEID.</li> <li>• interfaceAddress - IP address of interface.</li> <li>• nodeInput - Pointer to node input.</li> <li>• index - Parameter name.</li> <li>• wasFound - Storage for success of search.</li> <li>• readVal - Storage for parameter value.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<p><b>IO_ReadString</b></p> <p>This API is used to retrieve a string parameter value from input files.</p>	<p>void <b>IO_ReadString</b> (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"> <li>• nodeId - nodeId. Can be ANY_NODEID.</li> <li>• interfaceAddress - IP address of interface.</li> <li>• nodeInput - Pointer to node input.</li> <li>• index - Parameter name.</li> <li>• wasFound - Storage for success of search.</li> <li>• readVal - Storage for parameter value.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<p><b>IO_ReadBool</b></p> <p>This API is used to retrieve a boolean parameter value from input files.</p>	<p>void <b>IO_ReadBool</b> (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"> <li>• nodeId - nodeId. Can be ANY_NODEID.</li> <li>• interfaceAddress - IP address of interface.</li> <li>• nodeInput - Pointer to node input.</li> <li>• index - Parameter name.</li> <li>• wasFound - Storage for success of search.</li> <li>• readVal - Storage for parameter value.</li> </ul>

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

	<ul style="list-style-type: none"> <li>• nodeInput - Pointer to node input.</li> <li>• index - Parameter name.</li> <li>• wasFound - Storage for success of search.</li> <li>• readVal - Storage for parameter value.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>IO_ReadTime</b>	<p>This API is used to retrieve time parameter value from input files.</p> <p>void <b>IO_ReadTime</b> (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"> <li>• nodeId - nodeId. Can be ANY_NODEID.</li> <li>• interfaceAddress - IP address of interface.</li> <li>• nodeInput - Pointer to node input.</li> <li>• index - Parameter name.</li> <li>• wasFound - Storage for success of search.</li> <li>• readVal - Storage for parameter value.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>IO_ReadCachedFile</b>	<p>This API is used to retrieve cached file parameter value from input files.</p> <p>void <b>IO_ReadCachedFile</b> (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"> <li>• nodeId - nodeId. Can be ANY_NODEID.</li> <li>• interfaceAddress - IP address of interface.</li> <li>• nodeInput - Pointer to node input.</li> <li>• index - Parameter name.</li> <li>• wasFound - Storage for success of search.</li> <li>• parameterValue - Storage for parameter value.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>IO_ReadStringInstance</b>	void <b>IO_ReadStringInstance</b> (const NodeAddress nodeId, const NodeAddress interfaceAddress, const NodeInput* nodeInput,

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

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

	<ul style="list-style-type: none"> <li>• <code>interfaceAddress</code> - IP address of interface.</li> <li>• <code>nodeInput</code> - Pointer to node input.</li> <li>• <code>parameterName</code> - Parameter name.</li> <li>• <code>parameterInstanceNumber</code> - Instance number.</li> <li>• <code>fallbackIfNoInstanceMatch</code> - Selects parameter without</li> <li>• <code>wasFound</code> - Storage for success of search.</li> <li>• <code>parameterValue</code> - Storage for parameter value.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>IO_ReadTimeInstance</b>	<p>This API is used to retrieve time parameter values from input files for a specific instance.</p> <pre>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)</pre> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>nodeId</code> - nodeId. Can be ANY_NODEID.</li> <li>• <code>interfaceAddress</code> - IP address of interface.</li> <li>• <code>nodeInput</code> - Pointer to node input.</li> <li>• <code>parameterName</code> - Parameter name.</li> <li>• <code>parameterInstanceNumber</code> - Instance number.</li> <li>• <code>fallbackIfNoInstanceMatch</code> - Selects parameter without</li> <li>• <code>wasFound</code> - Storage for success of search.</li> <li>• <code>parameterValue</code> - Storage for parameter value.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>IO_ReadCachedFileInstance</b>	<p>This API is used to retrieve file parameter values from input files for a specific instance.</p> <pre>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)</pre> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>nodeId</code> - nodeId. Can be ANY_NODEID.</li> <li>• <code>interfaceAddress</code> - IP address of interface.</li> </ul>

	<p><code>nodeInput</code> - Pointer to node input.</p> <ul style="list-style-type: none"> <li>• <code>parameterName</code> - Parameter name.</li> <li>• <code>parameterInstanceNumber</code> - Instance number.</li> <li>• <code>fallbackIfNoInstanceMatch</code> - Selects parameter without instance if no match found.</li> <li>• <code>wasFound</code> - Storage for success of search.</li> <li>• <code>parameterValue</code> - Storage for parameter value.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>IO_ParseNodeIdHostOrNetworkAddress</b>	<p>Parses a string for a nodeId, host address, or network address.</p> <p><code>void IO_ParseNodeIdHostOrNetworkAddress (const char s[], NodeAddress* outputNodeAddress, int* numHostBits, BOOL* isNodeId)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>s[]</code> - String to parse.</li> <li>• <code>outputNodeAddress</code> - Storage for nodeId or IP address.</li> <li>• <code>numHostBits</code> - Storage for number of host bits</li> <li>• <code>isNodeId</code> - Storage for whether the string is a nodeId.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>IO_ParseNodeIdOrHostAddress</b>	<p>Parses a string for a nodeId or host address.</p> <p><code>void IO_ParseNodeIdOrHostAddress (const char s[], NodeAddress* outputNodeAddress, BOOL* isNodeId)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>s[]</code> - String to parse.</li> <li>• <code>outputNodeAddress</code> - Storage for nodeId or IP address.</li> <li>• <code>isNodeId</code> - Storage for whether the string is a nodeId.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>IO_ParseNetworkAddress</b>	<p>Parses a string for a network address.</p> <p><code>void IO_ParseNetworkAddress (const char s[], NodeAddress* outputNodeAddress, int* numHostBits)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>s[]</code> - String to parse.</li> <li>• <code>outputNodeAddress</code> - Storage for network address.</li> </ul>

	<ul style="list-style-type: none"> <li>• <code>numHostBits</code> - Storage for number of host bits</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>IO_FreeNodeInput</b>	<p><code>void IO_FreeNodeInput (NodeInput* nodeInput)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>nodeInput</code> - Pointer to node input.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>IO_PrintStat</b>	<p>Print out the relevant stat in "buf", along with the node id and the layer type generating this stat.</p> <p><code>void IO_PrintStat (Node* node, const char* layer, const char* protocol, NodeAddress interfaceAddress, int instanceId, const char* buf)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - The node generating the stat.</li> <li>• <code>layer</code> - The layer generating the stat.</li> <li>• <code>protocol</code> - The protocol generating the stat.</li> <li>• <code>interfaceAddress</code> - Interface address.</li> <li>• <code>instanceId</code> - Instance id.</li> <li>• <code>buf</code> - String which has the statistic to</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>IO_AppParseSourceAndDestStrings</b>	<p>Application input parsing API. Parses the source and destination strings.</p> <p><code>void IO_AppParseSourceAndDestStrings (Node* node, const char* inputString, const char* sourceString, NodeAddress* sourceNodeId, NodeAddress* sourceAddr, const char* destString, NodeAddress* destNodeId, NodeAddress* destAddr)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - A pointer to Node.</li> <li>• <code>inputString</code> - The input string.</li> <li>• <code>sourceString</code> - The source string.</li> <li>• <code>sourceNodeId</code> - A pointer to NodeAddress.</li> <li>• <code>sourceAddr</code> - A pointer to NodeAddress.</li> <li>• <code>destString</code> - Const char pointer.</li> </ul>

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

	<ul style="list-style-type: none"> <li>• destString - Const char pointer.</li> <li>• destNodeId - A pointer to NodeAddress.</li> <li>• destAddr - A pointer to NodeAddress.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>IO_AppForbidSameSourceAndDest</b>	<p>Application input checking API. Checks for the same source and destination node id. Calls abort() for same source and destination.</p> <p><b>void IO_AppForbidSameSourceAndDest</b> (const char* inputString, NodeAddress sourceNodeId, NodeAddress destNodeId)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• inputString - The input string.</li> <li>• sourceNodeId - Source node id, read from the</li> <li>• destNodeId - Destination node id, read from the</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>QualifierMatches</b>	<p>This is an auxiliary API used by the IO_Read...() set of APIs.</p> <p><b>BOOL QualifierMatches</b> (const NodeAddress nodeId, const NodeAddress interfaceAddress, const char* qualifier, int* matchType)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• nodeId - nodeId to select for.</li> <li>• interfaceAddress - IP address to select for.</li> <li>• qualifier - String containing the</li> <li>• matchType - Stores the type of the match,</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• BOOL - Returns TRUE if match found. FALSE, otherwise.</li> </ul>
<b>IO_ReadBool</b>	<p>This API is used to retrieve boolean parameter values from input files. Overloaded API for Ipv6 compatibility.</p> <p><b>void IO_ReadBool</b> (const NodeAddress nodeId, const in6_addr* interfaceAddress, const NodeInput* nodeInput, const char* parameterName, BOOL* wasFound, BOOL* parameterValue)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• nodeId - nodeId. Can be ANY_NODEID.</li> <li>• interfaceAddress - IPv6 address of interface.</li> <li>• nodeInput - Pointer to node input.</li> <li>• parameterName - Parameter name.</li> <li>• wasFound - Storage for success of seach.</li> </ul>

	<ul style="list-style-type: none"> <li>• <code>parameterValue</code> - Storage for parameter value.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>IO_ReadBool</b>	<p>None <b>IO_ReadBool ()</b></p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>None</code> - None</li> </ul>
<b>IO_ReadBool</b>	<p>void <b>IO_ReadBool</b> (const NodeAddress nodeId, const Address* address, const NodeInput* nodeInput, const char* parameterName, BOOL* wasFound, BOOL* parameterValue)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>nodeId</code> - nodeId. Can be ANY_NODEID.</li> <li>• <code>address</code> - Address of interface.</li> <li>• <code>nodeInput</code> - Pointer to node input.</li> <li>• <code>parameterName</code> - Parameter name.</li> <li>• <code>wasFound</code> - Storage for success of search.</li> <li>• <code>parameterValue</code> - Storage for parameter value.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>IO_ReadString</b>	<p>void <b>IO_ReadString</b> (const NodeId nodeId, const in6_addr* interfaceAddress, const NodeInput* nodeInput, const char* index, BOOL* wasFound, char* readVal)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>nodeId</code> - nodeId. Can be ANY_NODEID.</li> <li>• <code>interfaceAddress</code> - IP address of interface.</li> <li>• <code>nodeInput</code> - Pointer to node input.</li> <li>• <code>index</code> - Parameter name.</li> <li>• <code>wasFound</code> - Storage for success of search.</li> <li>• <code>readVal</code> - Storage for parameter value.</li> </ul> <p>Returns:</p>

	<p><b>void - None</b></p>
<b>IO_ReadString</b>	<p>Reads string value for specified ATM address.</p> <p><b>void IO_ReadString (const NodeId nodeId, const AtmAddress* interfaceAddress, const NodeInput * nodeInput, const char * parameterName, BOOL * wasFound, char * parameterValue)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>nodeId</code> - NodeId for which parameter has</li> <li>• <code>interfaceAddress</code> - ATM Interface address</li> <li>• <code>nodeInput</code> - pointer to configuration inputs</li> <li>• <code>parameterName</code> - Parameter to be read</li> <li>• <code>wasFound</code> - Parameter found or not</li> <li>• <code>parameterValue</code> - Parameter's value if found.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void - None</code></li> </ul>
<b>IO_ReadString</b>	<p>This API is used to retrieve a string parameter value from input files. Overloaded API for Ipv6 compatibility.</p> <p><b>void IO_ReadString (const NodeId nodeId, const Address* address, const NodeInput* nodeInput, const char* index, BOOL* wasFound, char* readVal)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>nodeId</code> - nodeId. Can be ANY_NODEID.</li> <li>• <code>address</code> - IP address of interface.</li> <li>• <code>nodeInput</code> - Pointer to node input.</li> <li>• <code>index</code> - Parameter name.</li> <li>• <code>wasFound</code> - Storage for success of seach.</li> <li>• <code>readVal</code> - Storage for parameter value.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void - None</code></li> </ul>
<b>IO_ReadInt</b>	<p>This API is used to retrieve an integer parameter value from input files. Overloaded API for Ipv6 compatibility.</p> <p><b>void IO_ReadInt (const NodeId nodeId, const in6_addr* interfaceAddress, const NodeInput* nodeInput, const char* index, BOOL* wasFound, int* readVal)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>nodeId</code> - nodeId. Can be ANY_NODEID.</li> <li>• <code>interfaceAddress</code> - IPv6 address of interface.</li> <li>• <code>nodeInput</code> - Pointer to node input.</li> </ul>

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

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

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

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

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

	<p><code>wasFound</code> - Storage for success of seach.</p> <ul style="list-style-type: none"> <li>• <code>parameterValue</code> - Storage for parameter value.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>IO_ReadIntInstance</b>	<p>This API is used to retrieve integer parameter values from input files for a specific instance. Overloaded API for Ipv6 compatibility.</p> <p><code>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)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>nodeId</code> - nodeId. Can be ANY_NODEID.</li> <li>• <code>address</code> - Address of interface.</li> <li>• <code>nodeInput</code> - Pointer to node input.</li> <li>• <code>parameterName</code> - Parameter name.</li> <li>• <code>parameterInstanceNumber</code> - Instance number.</li> <li>• <code>fallbackIfNoInstanceMatch</code> - Selects parameter without</li> <li>• <code>wasFound</code> - Storage for success of seach.</li> <li>• <code>parameterValue</code> - Storage for parameter value.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>IO_ReadDoubleInstance</b>	<p>This API is used to retrieve double parameter values from input files for a specific instance. Overloaded API for Ipv6 compatibility.</p> <p><code>void IO_ReadDoubleInstance (const NodeId nodeId, const inf6_addr* interfaceAddress, 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"> <li>• <code>nodeId</code> - nodeId. Can be ANY_NODEID.</li> <li>• <code>interfaceAddress</code> - IPv6 address of interface.</li> <li>• <code>nodeInput</code> - Pointer to node input.</li> <li>• <code>parameterName</code> - Parameter name.</li> <li>• <code>parameterInstanceNumber</code> - Instance number.</li> <li>• <code>fallbackIfNoInstanceMatch</code> - Selects parameter without</li> <li>• <code>wasFound</code> - Storage for success of seach.</li> <li>• <code>parameterValue</code> - Storage for parameter value.</li> </ul>

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

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

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

	<ul style="list-style-type: none"> <li>• <code>parameterValue</code> - Storage for parameter value.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>IO_ReadCachedFileInstance</b>	<p>This API is used to retrieve file parameter values from input files for a specific instance. Overloaded API for Ipv6 compatibility.</p> <pre>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)</pre> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>nodeId</code> - nodeId. Can be ANY_NODEID.</li> <li>• <code>address</code> - Address of interface.</li> <li>• <code>nodeInput</code> - Pointer to node input.</li> <li>• <code>parameterName</code> - Parameter name.</li> <li>• <code>parameterInstanceNumber</code> - Instance number.</li> <li>• <code>fallbackIfNoInstanceMatch</code> - Selects parameter without</li> <li>• <code>wasFound</code> - Storage for success of seach.</li> <li>• <code>parameterValue</code> - Storage for parameter value.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>IO_PrintStat</b>	<p>Print out the relevant stat in "buf", along with the node id and the layer type generating this stat. Overloaded API for Ipv6 compatibility.</p> <pre>void IO_PrintStat (Node* node, const char* layer, const char* protocol, const char* interfaceAddress, int instanceId, const char* buf)</pre> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - The node generating the stat.</li> <li>• <code>layer</code> - The layer generating the stat.</li> <li>• <code>protocol</code> - The protocol generating the stat.</li> <li>• <code>interfaceAddress</code> - The Interface address the stat.</li> <li>• <code>instanceId</code> - Instance id.</li> <li>• <code>buf</code> - String which has the statistic to</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>IO_ParseNodeIdHostOrNetworkAddress</b>	<pre>void IO_ParseNodeIdHostOrNetworkAddress (const char s[], in6_addr* ipAddress, BOOL* isIpAddr, NodeId* nodeId)</pre>

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

	<p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>IO_AppParseSourceAndDestStrings</b>	<p>void <b>IO_AppParseSourceAndDestStrings</b> (Node* node, const char* inputString, const char* sourceString, NodeId* sourceNodeId, Address* sourceAddr, const char* destString, NodeId* destNodeId, Address* destAddr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - A pointer to Node.</li> <li>• inputString - The input string.</li> <li>• sourceString - The source string.</li> <li>• sourceNodeId - A pointer to NodeId.</li> <li>• sourceAddr - A pointer to Address.</li> <li>• destString - Const char pointer.</li> <li>• destNodeId - A pointer to NodeId.</li> <li>• destAddr - A pointer to Address.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>IO_AppParseSourceString</b>	<p>void <b>IO_AppParseSourceString</b> (Node* node, const char* inputString, const char* sourceString, NodeAddress* sourceNodeId, Address* sourceAddr, NetworkType networkType)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - A pointer to Node.</li> <li>• inputString - The input string.</li> <li>• sourceString - The source string.</li> <li>• sourceNodeId - A pointer to NodeAddress.</li> <li>• sourceAddr - A pointer to Address.</li> <li>• networkType - used when sourceString</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>IO_AppParseDestString</b>	<p>void <b>IO_AppParseDestString</b> (Node* node, const char* inputString, const char* destString, NodeAddress* destNodeId, Address* destAddr, NetworkType networkType)</p> <p>Parameters:</p>

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

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

<p>This API is used to retrieve a string parameter value from input files.</p>	<p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• <code>node</code> - Node pointer for which string is</li> <li>• <code>nodeInput</code> - Pointer to node input.</li> <li>• <code>index</code> - Parameter name.</li> <li>• <code>wasFound</code> - Storage for success of search.</li> <li>• <code>readVal</code> - Storage for parameter value.</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<p><b>IO_CacheFile</b></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>	<p><b>void IO_CacheFile (const NodeInput* nodeInput, const char* filename)</b></p> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• <code>nodeInput</code> - Pointer to node input.</li> <li>• <code>filename</code> - Path to input file.</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>• <code>void</code> - NULL</li> </ul>
<p><b>IO_GetMaxLen</b></p> <p>This API is used to get the maximum length of a line in the file.</p>	<p><b>int IO_GetMaxLen (fileName char*)</b></p> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• <code>char*</code> - Pointer to the name of the file.</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>• <code>int</code> - Interger with the largest line length.</li> </ul>
<p><b>IO_GetMaxLen</b></p> <p>This API is used to get the maximum length of a line in the file.</p>	<p><b>int IO_GetMaxLen (fp FILE*)</b></p> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• <code>FILE*</code> - Pointer to a file stream.</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>• <code>int</code> - Interger with the largest line length.</li> </ul>
<p><b>NI_GetMaxLen</b></p> <p>This API is used to get the maximum length of a line in nodeInput.</p>	<p><b>int NI_GetMaxLen (nodeInput NodeInput*)</b></p> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• <code>NodeInput*</code> - Pointer to a node input.</li> </ul>

	<p>Returns:</p> <ul style="list-style-type: none"> <li>• int - Integer with the largest line length.</li> </ul>
<b>NI_GetMaxLen</b>	<p>int <b>NI_GetMaxLen</b> (nodeInput const NodeInput*)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• const NodeInput* - Pointer to a node input.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• int - Integer with the largest line length.</li> </ul>
<b>IO_ParseNetworkAddress</b>	<p>Parses a string for a network address. Overloaded API for ATM compatibility.</p> <p>void <b>IO_ParseNetworkAddress</b> (const char s[], unsigned int* u_atmVal)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• s[] - String to parse.</li> <li>• u_atmVal - Storage for icd, aid, ptp</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>IO_ConvertAddrToString</b>	<p>Convert generic address to appropriate network type address string format.</p> <p>void <b>IO_ConvertAddrToString</b> (Address* address, char* addrStr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• address - generic address</li> <li>• addrStr - address string</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - NULL</li> </ul>
<b>IO_ConvertAtmAddressToString</b>	<p>Convert Atm address to address string format.</p> <p>void <b>IO_ConvertAtmAddressToString</b> (AtmAddress addr, char* addrStr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• addr - Atm address</li> <li>• addrStr - address string</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - NULL</li> </ul>
<b>IO_InsertIntValue</b>	<p>Insert integer value for specific string in case</p> <p>void <b>IO_InsertIntValue</b> (const char s[], const unsigned int val, unsigned int u_atmVal)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• s[] - character array</li> </ul>

of ATM	<ul style="list-style-type: none"> <li>• <code>val</code> - value to be inserted</li> <li>• <code>u_atmVal</code> - atm_value need to be checked</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - NULL</li> </ul>
<b>IO_ReadCachedFileIndex</b>  Return Cached file index for the given parameter name	<pre>int IO_ReadCachedFileIndex (NodeAddress nodeId, NodeAddress interfaceAddress, unsigned int nodeInput)</pre> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>nodeId</code> - node Id</li> <li>• <code>interfaceAddress</code> - Interface Address for the given node</li> <li>• <code>nodeInput</code> - atm_value need to be checked</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>int</code> - None</li> </ul>
<b>IO_ReadString</b>  This API is used to retrieve a string parameter value from input files.	<pre>void IO_ReadString (Node* node, const NodeId nodeId, int interfaceIndex, const NodeInput* nodeInput, const char* parameterName, BOOL* wasFound, char* parameterValue)</pre> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - node structure pointer.</li> <li>• <code>nodeId</code> - nodeId.</li> <li>• <code>interfaceIndex</code> - interface Index.</li> <li>• <code>nodeInput</code> - Pointer to node input.</li> <li>• <code>parameterName</code> - Parameter name.</li> <li>• <code>wasFound</code> - Storage for success of seach.</li> <li>• <code>parameterValue</code> - Storage for parameter value.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
This API is used to retrieve a Int64 parameter value from input files.	<pre>void IO_ReadInt64 (Node* node, const NodeId nodeId, int interfaceIndex, const NodeInput* nodeInput, const char* parameterName, BOOL* wasFound, Int64* parameterValue)</pre> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - node structure pointer.</li> <li>• <code>nodeId</code> - nodeId.</li> </ul>

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

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

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

	<ul style="list-style-type: none"> <li>• nodeId - nodeId.</li> <li>• interfaceIndex - interface Index.</li> <li>• nodeInput - Pointer to node input.</li> <li>• parameterName - Parameter name.</li> <li>• parameterInstanceNumber - Instance number.</li> <li>• fallbackIfNoInstanceMatch - Selects parameter without</li> <li>• wasFound - Storage for success of seach.</li> <li>• parameterValue - Storage for parameter value.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>IO_ReadIntInstance</b>	<p>This API is used to retrieve int parameter values from input files for a specific instance.</p> <p>void <b>IO_ReadIntInstance</b> (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"> <li>• node - node structure pointer.</li> <li>• nodeId - nodeId.</li> <li>• interfaceIndex - interface Index.</li> <li>• nodeInput - Pointer to node input.</li> <li>• parameterName - Parameter name.</li> <li>• parameterInstanceNumber - Instance number.</li> <li>• fallbackIfNoInstanceMatch - Selects parameter without</li> <li>• wasFound - Storage for success of seach.</li> <li>• parameterValue - Storage for parameter value.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>IO_ReadTimeInstance</b>	<p>This API is used to retrieve clocktype parameter values from input files for a specific instance.</p> <p>void <b>IO_ReadTimeInstance</b> (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"> <li>• node - node structure pointer.</li> </ul>

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

	<ul style="list-style-type: none"> <li>• <code>interfaceIndex</code> - interface Index.</li> <li>• <code>nodeInput</code> - Pointer to node input.</li> <li>• <code>parameterName</code> - Parameter name.</li> <li>• <code>wasFound</code> - Storage for success of seach.</li> <li>• <code>parameterValue</code> - Storage for parameter value.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>IO_ReadString</b>	<p>This API is used to retrieve a string parameter value from input files.</p> <p><code>void IO_ReadString (const NodeAddress nodeId, NodeAddress ipv4Address, in6_addr* ipv6Address, const NodeInput* nodeInput, const char* parameterName, BOOL* wasFound, char* parameterValue)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>nodeId</code> - nodeId.</li> <li>• <code>ipv4Address</code> - IP address of an interface</li> <li>• <code>ipv6Address</code> - IPv6 address of an interface</li> <li>• <code>nodeInput</code> - Pointer to node input.</li> <li>• <code>parameterName</code> - Parameter name.</li> <li>• <code>wasFound</code> - Storage for success of seach.</li> <li>• <code>parameterValue</code> - Storage for parameter value.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>IO_ReadString</b>	<p>This API is used to retrieve a string parameter value from input files.</p> <p><code>void IO_ReadString (const NodeAddress nodeId, int interfaceIndex, const NodeAddress ipv4SubnetAddress, const in6_addr* ipv6SubnetAddress, const NodeInput* nodeInput, const char* parameterName, char* parameterValue, BOOL&amp; wasFound, int&amp; matchType)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>nodeId</code> - nodeId.</li> <li>• <code>interfaceIndex</code> - interface index</li> <li>• <code>ipv4SubnetAddress</code> - IPv4 subnet address</li> <li>• <code>ipv6SubnetAddress</code> - IPv6 subnet address</li> <li>• <code>nodeInput</code> - Pointer to node input.</li> <li>• <code>parameterName</code> - Parameter name.</li> </ul>

	<ul style="list-style-type: none"> <li>• <code>parameterValue</code> - Storage for parameter value.</li> <li>• <code>wasFound</code> - Storage for success of search.</li> <li>• <code>matchType</code> - Storage for matchType.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>IO_ReadChannelMask</b>	<p>This API is used to retrieve the value channel mask for a specific instance.</p> <p><code>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)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - node structure pointer.</li> <li>• <code>nodeId</code> - <code>nodeId</code>.</li> <li>• <code>address</code> - Pointer to address</li> <li>• <code>interfaceIndex</code> - interface Index.</li> <li>• <code>nodeInput</code> - Pointer to node input.</li> <li>• <code>parameterName</code> - Parameter name.</li> <li>• <code>parameterInstanceNumber</code> - Instance number.</li> <li>• <code>fallbackIfNoInstanceMatch</code> - Selects parameter without</li> <li>• <code>wasFound</code> - Storage for success of search.</li> <li>• <code>parseChannelList</code> - Storage for identifying if</li> <li>• <code>parameterValue</code> - Storage for parameter value.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>IO_ReadBool</b>	<p>This API is used to retrieve a boolean parameter value from input files.</p> <p><code>void IO_ReadBool (Node* node, const NodeId nodeId, int interfaceIndex, const NodeInput* nodeInput, const char* parameterName, BOOL* wasFound, BOOL* parameterValue)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - node structure pointer.</li> <li>• <code>nodeId</code> - <code>nodeId</code>.</li> <li>• <code>interfaceIndex</code> - interface Index.</li> <li>• <code>nodeInput</code> - Pointer to node input.</li> </ul>

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



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

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

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



# EXata 5.1 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	<a href="#">GUI_DEFAULT_STEP</a>
	The default interval before waiting for the Animator handshake/STEP.
CONSTANT	<a href="#">GUI_DEFAULT_ICON</a>
	Icon used in case none is specified for a node.
CONSTANT	<a href="#">MAX_LAYERS</a>
	By default, there are 8 layers, but users may add more
CONSTANT	<a href="#">GUI_DEFAULT_DATA_TYPE</a>
	Default value to use for data types.
CONSTANT	<a href="#">GUI_EMULATION_DATA_TYPE</a>
	Default value to use for data types.
CONSTANT	<a href="#">GUI_DEFAULT_LINK_TYPE</a>
	Default value to use for link types.
CONSTANT	<a href="#">GUI_DEFAULT_NODE_TYPE</a>
	Default value to use for node types.
CONSTANT	<a href="#">GUI_DEFAULT_INTERFACE</a>

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

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

## Function / Macro Summary

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

	Changes the orientation of a node. <a href="#"><code>GUI_SetNodeIcon</code></a> (NodeId nodeID, char* iconFile, clocktype time)
void	Changes the icon associated with a node. <a href="#"><code>GUI_SetNodeLabel</code></a> (NodeId nodeID, char* label, clocktype time)
void	Changes the label (the node name) of a node. <a href="#"><code>GUI_SetNodeRange</code></a> (NodeId nodeID, int interfaceIndex, double range, clocktype time)
void	Changes the transmission range of a node <a href="#"><code>GUI_SetNodeType</code></a> (NodeId nodeID, int type, clocktype time)
void	Changes the (symbolic) type of a node <a href="#"><code>GUI_SetPatternIndex</code></a> (Node* node, int interfaceIndex, int index, clocktype time)
void	Sets the antenna pattern to one of a previously specified antenna pattern file. <a href="#"><code>GUI_SetPatternAndAngle</code></a> (node node*, int interfaceIndex, int index, int angleInDegrees, clocktype time)
void	For steerable antennas, it sets the pattern to use, and also an angle relative to the node's current orientation. <a href="#"><code>GUI_AddLink</code></a> (NodeId sourceID, NodeId destID, GuiLayers layer, int type, NodeAddress subnetAddress, int numHostBits, clocktype time)
void	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. <a href="#"><code>GUI_AddLink</code></a> (NodeId sourceID, NodeId destID, GuiLayers layer, int type, int tla, int nla, int sla, clocktype time)
void	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. <a href="#"><code>GUI_AddLink</code></a> (NodeId sourceID, NodeId destID, GuiLayers layer, int type, IPv6subnetAddress ip6_addr, IPv6subnetPrefixLen unsigned int, clocktype time)
void	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. <a href="#"><code>GUI_DeleteLink</code></a> (NodeId sourceID, NodeId destID, GuiLayers layer, int type, clocktype time)
void	Removes link of a specific type. <a href="#"><code>GUI_DeleteLink</code></a> (NodeId sourceID, NodeId destID, GuiLayers layer, clocktype time)

	<p>Removes the aforementioned link, no matter the "type."</p>
void	<code>GUI_Broadcast(NodeId nodeID, GuiLayers layer, int type, int interfaceIndex, clocktype time)</code>
	<p>Indicates a broadcast.</p>
void	<code>GUI_EndBroadcast(NodeId nodeID, GuiLayers layer, int type, int interfaceIndex, clocktype time)</code>
	<p>Indicates the end of a broadcast.</p>
void	<code>GUI_Multicast(NodeId nodeID, GuiLayers layer, int type, int interfaceIndex, clocktype time)</code>
	<p>Indicates a multicast. (Probably need to add a destination address.)</p>
void	<code>GUI_Uncast(NodeId sourceID, NodeId destID, GuiLayers layer, int type, int sendingInterfaceIndex, int receivingInterfaceIndex, clocktype time)</code>
	<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>
void	<code>GUI_Receive(NodeId sourceID, NodeId destID, GuiLayers layer, int type, int sendingInterfaceIndex, int receivingInterfaceIndex, clocktype time)</code>
	<p>Shows a successful receipt at a destination.</p>
void	<code>GUI_Drop(NodeId sourceID, NodeId destID, GuiLayers layer, int type, int sendingInterfaceIndex, int receivingInterfaceIndex, clocktype time)</code>
	<p>Shows a packet/frame/signal being dropped by a node.</p>
void	<code>GUI_Collision(NodeId nodeID, GuiLayers layer, clocktype time)</code>
	<p>Shows a node detecting a collision.</p>
void	<code>GUI_CreateSubnet(GuiSubnetTypes type, NodeAddress subnetAddress, int numHostBits, const char* nodeList, clocktype time)</code>
	<p>Creates a subnet. Normally done at startup.</p>
void	<code>GUI_CreateSubnet(GuiSubnetTypes type, in6_addr IPv6subnetAddress, unsigned int IPv6subnetPrefixLen, const char* nodeList, clocktype time)</code>
	<p>Creates a IPv6 subnet. Normally done at startup.</p>
void	<code>GUI_CreateSubnet(GuiSubnetTypes type, IPv6subnetAddress ip6_addr, IPv6subnetPrefixLen unsigned int, const char* nodeList, clocktype time)</code>

	<p>Creates a IPv6 subnet. Normally done at startup.</p>
void	<p><a href="#"><code>GUI_CreateHierarchy</code></a>(int componentID, char* nodeList)</p>
	<p>Since the GUI supports hierarchical design, this function informs the GUI of the contents of a hierarchical component.</p>
void	<p><a href="#"><code>GUI_MoveHierarchy</code></a>(int hierarchyId, Coordinates centerCoordinates, Orientation orientation, clocktype time)</p>
	<p>Moves the center point of a hierarchy to a new position.</p>
void	<p><a href="#"><code>GUI_CreateWeatherPattern</code></a>(int patternID, char* inputLine)</p>
	<p>Sends the input line describing a weather pattern to the GUI.</p>
void	<p><a href="#"><code>GUI_MoveWeatherPattern</code></a>(int patternID, Coordinates coordinates, clocktype time)</p>
	<p>Moves the first point of a weather pattern to a new position.</p>
void	<p><a href="#"><code>GUI_AddApplication</code></a>(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><a href="#"><code>GUI_DeleteApplication</code></a>(NodeId sourceID, NodeId destID, char* appName, int uniqueId, clocktype time)</p>
	<p>Deletes the labels shown by AddApplication.</p>
void	<p><a href="#"><code>GUI_AddInterfaceQueue</code></a>(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><a href="#"><code>GUI_QueueInsertPacket</code></a>(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><a href="#"><code>GUI_QueueDropPacket</code></a>(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><a href="#"><code>GUI_QueueDequeuePacket</code></a>(NodeId nodeID, GuiLayers layer, int interfaceIndex, unsigned priority, int packetSize, clocktype time)</p>
	<p>Dequeueing one packet from a queue for a node, interface and priority</p>
int	<p><a href="#"><code>GUI_DefineMetric</code></a>(char* name, NodeId nodeID, GuiLayers layer, int linkID, GuiDataTypes datatype, GuiMetrics metricType)</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.
void	<a href="#">GUI_SendIntegerData</a> (NodeId nodeID, int metricID, int value, clocktype time)  Sends data for an integer metric.
void	<a href="#">GUI_SendUnsignedData</a> (NodeId nodeID, int metricID, unsigned value, clocktype time)  Sends data for an unsigned metric.
void	<a href="#">GUI_SendRealData</a> (NodeId nodeID, int metricID, double value, clocktype time)  Sends data for a floating point metric.
bool	<a href="#">GUI_isAnimateOrInteractive</a> ()  Returns true if the GUI was activated on the command line.
void	<a href="#">GUI_EXTERNAL_Bootstrap</a> (int argc, char** argv, NodeInput* nodeInput, int thisPartitionId)  Creates a connection to the GUI
void	<a href="#">GUI_EXTERNAL_Registration</a> (PartitionData* partitionData, EXTERNAL_InterfaceList* list)  Registers the GUI as an external interface
void	<a href="#">GUI_CreateReply</a> (GuiReplies replyType, std msg)  Function used to replace newline characters in a string being sent to the GUI.

## Constant / Data Structure Detail

Constant	GUI_DEFAULT_STEP 1 econds  The default interval before waiting for the Animator handshake/STEP.
Constant	GUI_DEFAULT_ICON ""

	Icon used in case none is specified for a node.
Constant	<code>MAX_LAYERS 12</code>  By default, there are 8 layers, but users may add more
Constant	<code>GUI_DEFAULT_DATA_TYPE 0</code>  Default value to use for data types.
Constant	<code>GUI_EMULATION_DATA_TYPE 1</code>  Default value to use for data types.
Constant	<code>GUI_DEFAULT_LINK_TYPE 0</code>  Default value to use for link types.
Constant	<code>GUI_DEFAULT_NODE_TYPE 0</code>  Default value to use for node types.
Constant	<code>GUI_DEFAULT_INTERFACE 0</code>  Default interface for GUI commands.
Constant	<code>GUI_WIRELESS_LINK_TYPE 1</code>  Used to distinguish wireless and wired links.
Constant	<code>GUI_ATM_LINK_TYPE 2</code>  Used to distinguish ATM links from other types.
Constant	<code>GUI_COVERAGE_LINK_TYPE 3</code>  Used by Stats Manager
Constant	<code>GUI_MAX_COMMAND_LENGTH 1024</code>  Maximum length for a single interchange with Animator.

Enumeration	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.
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
<b>GUI_HandleHITLInput</b>  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 <b>GUI_HandleHITLInput</b> (const char * args, PartitionData * partition)  Parameters: <ul style="list-style-type: none"><li>• args - the command itself</li><li>• partition - the partition pointer</li></ul> Returns: <ul style="list-style-type: none"><li>• void - NULL</li></ul>
<b>GUI_Initialize</b>  Initializes the GUI in order to start the animation. The terrain map should give the	void <b>GUI_Initialize</b> (NodeInput* nodeInput, int numNodes, int coordinateSystem, Coordinates origin, Coordinates dimensions, clocktype maxClock)  Parameters: <ul style="list-style-type: none"><li>• nodeInput - configuration file</li></ul>

<p>path (either absolute, or relative to <b>QUALNET_HOME</b>) of an file to represent the terrain.</p>	<ul style="list-style-type: none"> <li>• <code>numNodes</code> - the number of nodes in the simulation</li> <li>• <code>coordinateSystem</code> - LATLONALT or CARTESIAN</li> <li>• <code>origin</code> - Southwest corner</li> <li>• <code>dimensions</code> - Northeast corner, or size</li> <li>• <code>maxClock</code> - length of the simulation</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - NULL</li> </ul>
<p><b>GUI_SetEffect</b></p> <p>This function will allow the protocol designer to specify the effect to use to display certain events.</p>	<p><code>void GUI_SetEffect (GuiEvents event, GuiLayers layer, int type, GuiEffects effect, GuiColors color)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>event</code> - the type of event for the new effect</li> <li>• <code>layer</code> - the protocol layer</li> <li>• <code>type</code> - special key to distinguish similar events</li> <li>• <code>effect</code> - the effect to use</li> <li>• <code>color</code> - optional color for the effect</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - NULL</li> </ul>
<p><b>GUI_InitNode</b></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><code>void GUI_InitNode (Node* node, NodeInput* nodeInput, clocktype time)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - the node</li> <li>• <code>nodeInput</code> - configuration file</li> <li>• <code>time</code> - the current simulation time</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - NULL</li> </ul>
<p><b>GUI_InitWirelessInterface</b></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><code>void GUI_InitWirelessInterface (Node* node, int interfaceIndex)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - the node</li> <li>• <code>interfaceIndex</code> - the interface to initialize</li> </ul>

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

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

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

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

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

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

	<ul style="list-style-type: none"> <li>• <code>interfaceIndex</code> - which of the node's interfaces to use</li> <li>• <code>time</code> - the current simulation time</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - NULL</li> </ul>
<b>GUI_Uncast</b>	<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> <pre>void <b>GUI_Uncast</b> (NodeId sourceID, NodeId destID, GuiLayers layer, int type, int sendingInterfaceIndex, int receivingInterfaceIndex, clocktype time)</pre> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>sourceID</code> - the source node</li> <li>• <code>destID</code> - the destination node</li> <li>• <code>layer</code> - protocol layer associated w/ the event</li> <li>• <code>type</code> - a user-defined type</li> <li>• <code>sendingInterfaceIndex</code> - sender's interface to use</li> <li>• <code>receivingInterfaceIndex</code> - receiver's interface to use</li> <li>• <code>time</code> - the current simulation time</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - NULL</li> </ul>
<b>GUI_Receive</b>	<p>Shows a successful receipt at a destination.</p> <pre>void <b>GUI_Receive</b> (NodeId sourceID, NodeId destID, GuiLayers layer, int type, int sendingInterfaceIndex, int receivingInterfaceIndex, clocktype time)</pre> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>sourceID</code> - the source node</li> <li>• <code>destID</code> - the destination node</li> <li>• <code>layer</code> - protocol layer associated w/ the event</li> <li>• <code>type</code> - a user-defined type</li> <li>• <code>sendingInterfaceIndex</code> - sender's interface to use</li> <li>• <code>receivingInterfaceIndex</code> - receiver's interface to use</li> <li>• <code>time</code> - the current simulation time</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - NULL</li> </ul>

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

<b>GUI_CreateSubnet</b>  Creates a IPv6 subnet. Normally done at startup.	<p>void <b>GUI_CreateSubnet</b> (GuiSubnetTypes type, in6_addr IPv6subnetAddress, unsigned int IPv6subnetPrefixLen, const char* nodeList, clocktype time)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <b>type</b> - GUI_WIRED/WIRELESS/SATELLITE_NETWORK</li> <li>• <b>IPv6subnetAddress</b> - base address for the subnet</li> <li>• <b>IPv6subnetPrefixLen</b> - number of network bits present</li> <li>• <b>nodeList</b> - the rest of the .config file SUBNET line</li> <li>• <b>time</b> - the current simulation time</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <b>void</b> - NULL</li> </ul>
<b>GUI_CreateSubnet</b>  Creates a IPv6 subnet. Normally done at startup.	<p>void <b>GUI_CreateSubnet</b> (GuiSubnetTypes type, IPv6subnetAddress ip6_addr, IPv6subnetPrefixLen unsigned int, const char* nodeList, clocktype time)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <b>type</b> - GUI_WIRED/WIRELESS/SATELLITE_NETWORK</li> <li>• <b>ip6_addr</b> - IPv6 address</li> <li>• <b>unsigned int</b> - IPv6 address prefix length</li> <li>• <b>nodeList</b> - the rest of the .config file SUBNET line</li> <li>• <b>time</b> - the current simulation time</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <b>void</b> - NULL</li> </ul>
<b>GUI_CreateHierarchy</b>  Since the GUI supports hierarchical design, this function informs the GUI of the contents of a hierarchical component.	<p>void <b>GUI_CreateHierarchy</b> (int componentID, char* nodeList)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <b>componentID</b> - an identifier for the hierarchy</li> <li>• <b>nodeList</b> - the rest of the .config file COMPONENT line</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <b>void</b> - NULL</li> </ul>
<b>GUI_MoveHierarchy</b>	<p>void <b>GUI_MoveHierarchy</b> (int hierarchyId, Coordinates centerCoordinates, Orientation orientation, clocktype time)</p> <p>Parameters:</p>

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

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

	<p><code>time</code> - the current simulation time</p> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - NULL</li> </ul>
<b>GUI_QueueDropPacket</b>	<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"> <li>• <code>nodeID</code> - the node's ID</li> <li>• <code>layer</code> - protocol layer associated w/ the event</li> <li>• <code>interfaceIndex</code> - associated interface of node</li> <li>• <code>priority</code> - priority of queue</li> <li>• <code>time</code> - the current simulation time</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - NULL</li> </ul>
<b>GUI_QueueDequeuePacket</b>	<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"> <li>• <code>nodeID</code> - the node's ID</li> <li>• <code>layer</code> - protocol layer associated w/ the event</li> <li>• <code>interfaceIndex</code> - associated interface of node</li> <li>• <code>priority</code> - priority of queue</li> <li>• <code>packetSize</code> - size of packet</li> <li>• <code>time</code> - the current simulation time</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - NULL</li> </ul>
<b>GUI_DefineMetric</b>	<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</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"> <li>• <code>name</code> - the name of the metric</li> <li>• <code>nodeID</code> - the node's ID</li> <li>• <code>layer</code> - protocol layer associated w/ the event</li> </ul>

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

	<p>Returns:</p> <ul style="list-style-type: none"> <li>• void - NULL</li> </ul>
<b>GUI_isAnimateOrInteractive</b>	<p>bool <b>GUI_isAnimateOrInteractive</b> ()</p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> <li>• bool - True if the GUI is enabled.</li> </ul>
<b>GUI_EXTERNAL_Bootstrap</b>	<p>void <b>GUI_EXTERNAL_Bootstrap</b> (int argc, char** argv, NodeInput* nodeInput, int thisPartitionId)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• argc - number of command line parameters</li> <li>• argv - command line parameters</li> <li>• nodeInput - the contents of the .config file</li> <li>• thisPartitionId - the ID of this partition</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - NULL</li> </ul>
<b>GUI_EXTERNAL_Registration</b>	<p>void <b>GUI_EXTERNAL_Registration</b> (PartitionData* partitionData, EXTERNAL_InterfaceList* list)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• partitionData - the partition to register with</li> <li>• list - the list to add oneself to</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - NULL</li> </ul>
<b>GUI_CreateReply</b>	<p>void <b>GUI_CreateReply</b> (GuiReplies replyType, std msg)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• replyType - the type of reply</li> <li>• msg - string*</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - NULL</li> </ul>



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

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

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

# EXata 5.1 API Reference

## IP

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

### Constant / Data Structure Summary

Type	Name
CONSTANT	<a href="#">IPVERSION4</a> Version of IP
CONSTANT	<a href="#">IPTOS_LOWDELAY</a> Type of service ( low delay)
CONSTANT	<a href="#">IPTOS_THROUGHPUT</a> Type of service ( throughput)
CONSTANT	<a href="#">IPTOS_RELIABILITY</a> Type of service (reliability)
CONSTANT	<a href="#">IPTOS_MINCOST</a> Type of service ( minimum cost)
CONSTANT	<a href="#">IPTOS_ECT</a> Bits 6 and 7 in the IPv4 TOS octet are designated as the ECN field. Bit 6 is designated as the ECT bit.
CONSTANT	<a href="#">IPTOS_CE</a> Bits 6 and 7 in the IPv4 TOS octet are designated as the ECN field. Bit 7 is designated as the CE bit.
CONSTANT	<a href="#">IPTOS_DSCP_MAX</a>

	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	<a href="#">IPTOS_DSCP_MIN</a>
	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	<a href="#">IPTOS_PREC_EFINTERNETCONTROL</a>
	IP precedence 'EF classe internet control'
CONSTANT	<a href="#">IPTOS_PREC_NETCONTROL</a>
	IP precedence 'net control'
CONSTANT	<a href="#">IPTOS_PREC_INTERNETCONTROL</a>
	IP precedence 'internet control'
CONSTANT	<a href="#">IPTOS_PREC_CRITIC_ECP</a>
	IP precedence 'critic ecp'
CONSTANT	<a href="#">IPTOS_PREC_FLASHOVERRIDE</a>
	IP precedence 'flash override'
CONSTANT	<a href="#">IPTOS_PREC_FLASH</a>
	IP precedence 'flash'
CONSTANT	<a href="#">IPTOS_PREC_IMMEDIATE</a>
	IP precedence 'immediate'
CONSTANT	<a href="#">IPTOS_PREC_PRIORITY</a>
	IP precedence 'priority'
CONSTANT	<a href="#">IPTOS_PREC_ROUTINE</a>
	IP precedence 'routing'
CONSTANT	<a href="#">IPTOS_PREC_INTERNETCONTROL_MIN_DELAY_SET</a>
	IP precedence 'internet control'with the 'minimize delay' bit set

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

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

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

	Internet implementation parameters ( default maximum segment size )
CONSTANT	<a href="#">IPPROTO_IP</a>  IP protocol numbers.
CONSTANT	<a href="#">IPPROTO_ICMP</a>  IP protocol numbers for ICMP.
CONSTANT	<a href="#">IPPROTO_IGMP</a>  IP protocol numbers for IGMP.
CONSTANT	<a href="#">IPPROTO_IPIP</a>  IP protocol numbers for IP tunneling.
CONSTANT	<a href="#">IPPROTO_TCP</a>  IP protocol numbers for TCP .
CONSTANT	<a href="#">IPPROTO_UDP</a>  IP protocol numbers for UDP
CONSTANT	<a href="#">IPPROTO_IPV6</a>  IP protocol number for DUAL-IP.
CONSTANT	<a href="#">IPPROTO_RSVP</a>  IP protocol numbers for RSVP.
CONSTANT	<a href="#">IPPROTO_MOBILE_IP</a>  IP protocol numbers for MOBILE_IP.
CONSTANT	<a href="#">IPPROTO_CES_HAIEP</a>  IP protocol numbers.
CONSTANT	<a href="#">IPPROTO_ESP</a>

	IP protocol numbers for IPSEC.
CONSTANT	<a href="#">IPPROTO_AH</a>
	IP protocol numbers for IPSEC.
CONSTANT	<a href="#">IPPROTO_ISAKMP</a>
	IP protocol numbers for IPSEC.
CONSTANT	<a href="#">IPPROTO_CES_ISAKMP</a>
	IP protocol numbers for IPSEC.
CONSTANT	<a href="#">IPPROTO_IAHNP</a>
	IP protocol numbers.
CONSTANT	<a href="#">IPPROTO_OSPF</a>
	IP protocol numbers for OSPF .
CONSTANT	<a href="#">IPPROTO_PIM</a>
	IP protocol numbers for PIM .
CONSTANT	<a href="#">IPPROTO_RPIM</a>
	IP protocol numbers for PIM .
CONSTANT	<a href="#">IPPROTO_IGRP</a>
	IP protocol numbers for IGRP .
CONSTANT	<a href="#">IPPROTO_EIGRP</a>
	IP protocol numbers for EIGRP .
CONSTANT	<a href="#">IPPROTO_BELLMANFORD</a>
	IP protocol numbers for BELLMANFORD.
CONSTANT	<a href="#">IPPROTO_IPIP_RED</a>
	IP protocol numbers for IP_RED.

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

	IP protocol numbers for ROUTING_CES_SRW IP encapsulation.
CONSTANT	<a href="#">IPPROTO_IPIP_SDR</a>
	IP protocol numbers for SDR IP encapsulation.
CONSTANT	<a href="#">IPPROTO_IPIP_SDR</a>
	IP protocol numbers for SDR IP encapsulation.
CONSTANT	<a href="#">IPPROTO_MULTICAST_CES_SRW_MOSPF</a>
	IP protocol numbers for MULTICAST_CES_SRW_MOSPF IP encapsulation.
CONSTANT	<a href="#">IPPROTO_CES_HSLS</a>
	IP protocol numbers for HSLS protocol.
CONSTANT	<a href="#">IPPROTO_AODV</a>
	IP protocol numbers for AODV .
CONSTANT	<a href="#">IPPROTO_DYMO</a>
	IP protocol numbers for DYMO .
CONSTANT	<a href="#">IPPROTO_MAODV</a>
	IP protocol numbers for MAODV.
CONSTANT	<a href="#">IPPROTO_DSR</a>
	IP protocol numbers for DSR .
CONSTANT	<a href="#">IPPROTO_ODMRP</a>
	IP protocol numbers for ODMRP .
CONSTANT	<a href="#">IPPROTO_LAR1</a>
	IP protocol numbers for LAR1.
CONSTANT	<a href="#">IPPROTO_STAR</a>

	IP protocol numbers for STAR. <a href="#"><u>IPPROTO_DAWN</u></a>
CONSTANT	IP protocol numbers for DAWN. <a href="#"><u>IPPROTO_EPLRS</u></a>
CONSTANT	IP protocol numbers for EPLRS protocol. <a href="#"><u>IPPROTO_CES_EPLRS</u></a>
CONSTANT	IP protocol numbers for EPLRS protocol for CES. <a href="#"><u>IPPROTO_CES_EPLRS_MPR</u></a>
CONSTANT	IP protocol numbers for EPLRS MPR protocol. <a href="#"><u>IPPROTO_DVMRP</u></a>
CONSTANT	IP protocol numbers for DVMRP.
CONSTANT	IP protocol numbers for GSM. <a href="#"><u>IPPROTO_GSM</u></a>
CONSTANT	IP protocol for external interface. <a href="#"><u>IPPROTO_EXTERNAL</u></a>
CONSTANT	IP protocol numbers for Internet gateway for emulated nodes <a href="#"><u>IPPROTO_INTERNET_GATEWAY</u></a>
CONSTANT	IP protocol numbers for Internet gateway for emulated nodes <a href="#"><u>IPPROTO_EXATA_VIRTUAL_LAN</u></a>
CONSTANT	IP protocol numbers for Internet gateway for emulated nodes <a href="#"><u>IPPROTO_NDP</u></a>
CONSTANT	IP protocol numbers for NDP. <a href="#"><u>IPPROTO_BRP</u></a>

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

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

STRUCT	<a href="#">NetworkMulticastForwardingTable</a>
	Structure of multicast forwarding table
STRUCT	<a href="#">NetworkIpMulticastGroupEntry</a>
	Structure for Multicast Group Entry
STRUCT	<a href="#">IpPerHopBehaviorInfoType</a>
	Structure to maintain DS priority queue mapping
STRUCT	<a href="#">IpMftcParameter</a>
	Variables of the structure define a unique condition class
STRUCT	<a href="#">IpMultiFieldTrafficConditionerInfo</a>
	Structure used to store traffic condition.
STRUCT	<a href="#">IpOptionsHeaderType</a>
	Structure of optional header for IP source route
STRUCT	<a href="#">NetworkIpStatsType</a>
	Structure used to keep track of all stats of network layer.
STRUCT	<a href="#">NetworkForwardingTableRow</a>
	Structure of an entity of forwarding table.
STRUCT	<a href="#">NetworkForwardingTable</a>
	Structure of forwarding table.
STRUCT	<a href="#">IpInterfaceInfoType</a>
	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	<a href="#">ip_frag_data</a>
	QualNet typedefs struct ip_frag_data to IpFragData. is a simple queue to hold fragmented packets.

STRUCT	<a href="#">Ipv6FragQueue</a>
	Ipv6 fragment queue structure.
STRUCT	<a href="#">FragmetedMsg</a>
	QualNet typedefs struct fragmeted_msg_struct to ip6q. struct fragmeted_msg_struct is a simple fragmented packets msg hold structure.
STRUCT	<a href="#">NetworkDataIp:</a>
	Main structure of network layer.
STRUCT	<a href="#">IpReassemblyBufferType</a>
	Structure of reassembly buffer
STRUCT	<a href="#">IpReassemblyBufferListCellType</a>
	Structure of reassembly buffer cell listing
STRUCT	<a href="#">IpReassemblyBufferListType</a>
	Structure of reassembly buffer list
STRUCT	<a href="#">AddressChangeType</a>
	enumeration to define address change events by DHCP

## Function / Macro Summary

Return Type	Summary
MACRO	<a href="#">IPOPT_COPIED(o)</a>  IP option 'copied'.
MACRO	<a href="#">IPOPT_CLASS(o)</a>  IP option 'class'
MACRO	<a href="#">IPOPT_NUMBER(o)</a>

	IP option 'number' <a href="#"><u>IpHeaderSize(ipHeader)</u></a>
MACRO	Returns IP header ip_hl field * 4, which is the size of the IP header in bytes.
MACRO	<a href="#"><u>SetIpHeaderSize(IpHeader, Size)</u></a>
	Sets IP header ip_hl field (header length) to Size divided by 4
MACRO	<a href="#"><u>FragmentOffset(ipHeader)</u></a>
	Starting position of this fragment in actual packet.
MACRO	<a href="#"><u>SetFragmentOffset(ipHeader, offset)</u></a>
	To set offset of fragment.
void	<a href="#"><u>IpHeaderSetVersion()</u></a> (UInt32* ip_v_hl_tos_len, unsigned int version)
	Set the value of version number for IpHeaderType
void	<a href="#"><u>IpHeaderSetHLen()</u></a> (UInt32* ip_v_hl_tos_len, unsigned int hlen)
	Set the value of header length for IpHeaderType
void	<a href="#"><u>IpHeaderSetTOS()</u></a> (UInt32* ip_v_hl_tos_len, unsigned int ipTos)
	Set the value of Type of Service for IpHeaderType
void	<a href="#"><u>IpHeaderSetIpLength()</u></a> (UInt32* ip_v_hl_tos_len, unsigned int ipLen)
	Set the value of ip length for IpHeaderType
void	<a href="#"><u>IpHeaderSetIpFragOffset()</u></a> (UInt16* ipFragment, UInt16 offset)
	Set the value of ip_fragment_offset for IpHeaderType
void	<a href="#"><u>IpHeaderSetIpReserved()</u></a> (UInt16* ipFragment, UInt16 ipReserved)
	Set the value of ipReserved for IpHeaderType
void	<a href="#"><u>IpHeaderSetIpDontFrag()</u></a> (UInt16* ipFragment, UInt16 dontFrag)
	Set the value of ip_dont_fragment for IpHeaderType

void	<a href="#">IpHeaderSetIpMoreFrag()</a> (UInt16* ipFragment, UInt16 moreFrag)
	Set the value of ip_more_fragment for IpHeaderType
unsigned int	<a href="#">IpHeaderGetVersion()</a> (UInt32 ip_v_hl_tos_len)
	Returns the value of version number for IpHeaderType
unsigned int	<a href="#">IpHeaderGetHLen()</a> (UInt32 ip_v_hl_tos_len)
	Returns the value of header length for IpHeaderType
unsigned int	<a href="#">IpHeaderGetTOS()</a> (UInt32 ip_v_hl_tos_len)
	Returns the value of Type of Service for IpHeaderType
unsigned int	<a href="#">IpHeaderGetIpLength()</a> (UInt32 ip_v_hl_tos_len)
	Returns the value of ip length for IpHeaderType
UInt16	<a href="#">IpHeaderGetIpFragOffset()</a> (UInt16 ipFragment)
	Returns the value of ip_fragment_offset for IpHeaderType
BOOL	<a href="#">IpHeaderGetIpDontFrag()</a> (UInt16 ipFragment)
	Returns the value of ip_dont_fragment for IpHeaderType
BOOL	<a href="#">IpHeaderGetIpMoreFrag()</a> (UInt16 ipFragment)
	Returns the value of ip_more_fragment for IpHeaderType
BOOL	<a href="#">IpHeaderGetIpReserved()</a> (UInt16 ipFragment)
	Returns the value of ipReserved for IpHeaderType
void	<a href="#">Ip_timestampSetFlag()</a> (unsigned char flgOflw, unsigned char flag)
	Set the value of flag for ip_timestamp_str
void	<a href="#">Ip_timestampSetOvflw()</a> (unsigned char flgOflw, unsigned char ovflw)
	Set the value of ovflw for ip_timestamp_str
unsigned char	<a href="#">Ip_timestampGetFlag()</a> (unsigned char flgOflw)

	Returns the value of flag for ip_timestamp_str <a href="#">Ip_timestampGetOvflw()</a> (unsigned char flgOflw)
unsigned char	Returns the value of overflow counter for ip_timestamp_str <a href="#">ConvertNumHostBitsToSubnetMask</a> (int numHostBits)
NodeAddress	To generate subnetmask using number of host bit <a href="#">ConvertSubnetMaskToNumHostBits</a> (NodeAddress subnetMask)
int	To generate number of host bit using subnetmask.
NodeAddress	<a href="#">MaskIpAddress</a> (NodeAddress address, NodeAddress mask) To mask a ip address.
NodeAddress	<a href="#">MaskIpAddressWithNumHostBits</a> (NodeAddress address, int numHostBits) To mask a ip address.
NodeAddress	<a href="#">CalcBroadcastIpAddress</a> (NodeAddress address, int numHostBits) To generate broadcast address.
BOOL	<a href="#">IsIpAddressInSubnet</a> (NodeAddress address, NodeAddress subnetAddress, int numHostbits) To check if a ip address belongs to a subnet.
void	<a href="#">NetworkIpAddHeader</a> (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*	<a href="#">FindAnIpOptionField</a> (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	<a href="#">NetworkIpPreInit</a> (Node* node) IP initialization required before any of the other layers are initialized. This is mainly for MAC initialization, which requires certain

	<p>IP structures be pre-initialized.</p>
void	<p><a href="#"><code>NetworkIpInit</code></a>(Node* node, const NodeInput* nodeInput)</p> <p>Initialize IP variables, and all network-layer IP protocols..</p>
void	<p><a href="#"><code>NetworkIpLayer</code></a>(Node* node, Message* msg)</p> <p>Handle IP layer events, incoming messages and messages sent to itself (timers, etc.).</p>
void	<p><a href="#"><code>NetworkIpFinalize</code></a>(Node* node)</p> <p>Finalize function for the IP model. Finalize functions for all network-layer IP protocols are called here.</p>
void	<p><a href="#"><code>NetworkIpReceivePacketFromTransportLayer</code></a>(Node* node, Message* msg, NodeAddress sourceAddress, NodeAddress destinationAddress, int outgoingInterface, TosType priority, unsigned char protocol, BOOL isEcnCapable)</p> <p>Called by transport layer protocols (UDP, TCP) to send UDP datagrams and TCP segments using IP. Simply calls <code>NetworkIpSendRawMessage()</code>.</p>
void	<p><a href="#"><code>NetworkIpSendRawMessage</code></a>(Node* node, Message* msg, NodeAddress sourceAddress, NodeAddress destinationAddress, int outgoingInterface, TosType priority, unsigned char protocol, unsigned ttl)</p> <p>Called by <code>NetworkIpReceivePacketFromTransportLayer()</code> to send to send UDP datagrams, TCP segments using IP. Also called by network-layer routing protocols (AODV, OSPF, etc.) to send IP packets. This function adds an IP header and calls <code>RoutePacketAndSendToMac()</code>.</p>
void	<p><a href="#"><code>NetworkIpSendRawMessageWithDelay</code></a>(Node* node, Message* msg, NodeAddress sourceAddress, NodeAddress destinationAddress, int outgoingInterface, TosType priority, unsigned char protocol, unsigned ttl, clocktype delay)</p> <p>Same as <code>NetworkIpSendRawMessage()</code>, but schedules event after a simulation delay.</p>
void	<p><a href="#"><code>NetworkIpSendRawMessageToMacLayer</code></a>(Node* node, Message* msg, NodeAddress sourceAddress, NodeAddress destinationAddress, TosType priority, unsigned char protocol, unsigned ttl, int interfaceIndex, NodeAddress nextHop)</p> <p>Called by network-layer routing protocols (AODV, OSPF, etc.) to add an IP header to payload data, and with the resulting IP packet, calls <code>NetworkIpSendPacketOnInterface()</code>.</p>
void	<p><a href="#"><code>NetworkIpSendRawMessageToMacLayerWithDelay</code></a>(Node* node, Message* msg, NodeAddress sourceAddress, NodeAddress destinationAddress, TosType priority, unsigned char protocol, unsigned ttl, int interfaceIndex, NodeAddress nextHop, clocktype delay)</p> <p>Same as <code>NetworkIpSendRawMessageToMacLayer()</code>, but schedules the event after a simulation delay by calling <code>NetworkIpSendPacketOnInterfaceWithDelay()</code>.</p>
void	<p><a href="#"><code>NetworkIpSendPacketToMacLayer</code></a>(Node* node, Message* msg, int interfaceIndex, NodeAddress nextHop)</p>

	<p>This function is called once the outgoing interface index and next hop address to which to route an IP packet are known.</p> <pre><code>void NetworkIpSendPacketOnInterface(Node* node, Message* msg, int incomingInterface, int outgoingInterface, NodeAddress nextHop)</code></pre>
	<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	<pre><code>NetworkIpSendPacketToMacLayerWithDelay(Node* node, Message* msg, int interfaceIndex, NodeAddress nextHop, clocktype delay)</code></pre> <p>Same as NetworkIpSendPacketOnInterface(), but schedules event after a simulation delay.</p>
void	<pre><code>NetworkIpSendPacketOnInterfaceWithDelay(Node* node, Message* msg, int incomingInterface, int outgoingInterface, NodeAddress nextHop, clocktype delay)</code></pre> <p>Same as NetworkIpSendPacketOnInterface(), but schedules event after a simulation delay.</p>
void	<pre><code>NetworkIpSendRawPacketOnInterfaceWithDelay(Node* node, Message* msg, int incomingInterface, int outgoingInterface, NodeAddress nextHop, clocktype delay)</code></pre> <p>Same as NetworkIpSendPacketOnInterface(), but schedules event after a simulation delay and denotes raw packet.</p>
void	<pre><code>NetworkIpSendPacketToMacLayerWithNewStrictSourceRoute(Node* node, Message* msg, NodeAddress[] newRouteAddresses, int numNewRouteAddresses, BOOL removeExistingRecordedRoute)</code></pre> <p>Tacks on a new source route to an existing IP packet and then sends the packet to the MAC layer.</p>
void	<pre><code>NetworkIpReceivePacketFromMacLayer(Node* node, Message* msg, NodeAddress previousHopNodeId, int interfaceIndex)</code></pre> <p>IP received IP packet from MAC layer. Updates the Stats database and then calls NetworkIpReceivePacket.</p>
void	<pre><code>NetworkIpReceivePacket(Node* node, Message* msg, NodeAddress previousHopNodeId, int interfaceIndex)</code></pre> <p>IP received IP packet. Determine whether the packet is to be delivered to this node, or needs to be forwarded. ipHeader-&gt;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	<pre><code>NetworkIpNotificationOfPacketDrop(Node* node, Message* msg, NodeAddress nextHopNodeId, int interfaceIndex)</code></pre> <p>Invoke callback functions when a packet is dropped.</p>
MacLayerStatusEventHandlerFunctionType	<pre><code>NetworkIpGetMacLayerStatusEventHandlerFunction(Node* node, int interfaceIndex)</code></pre>

	Get the status event handler function pointer.
void	<p><a href="#"><code>NetworkIpSetMacLayerStatusEventHandlerFunction</code></a>(Node* node, MacLayerStatusEventHandlerType StatusEventHandlerPtr, int interfaceIndex)</p> <p>Allows the MAC layer to send status messages (e.g., packet drop, link failure) to a network-layer routing protocol for routing optimization.</p>
void	<p><a href="#"><code>NetworkIpSneakPeekAtMacPacket</code></a>(Node* node, const Message* msg, int interfaceIndex, NodeAddress prevHop)</p> <p>Called Directly by the MAC layer, this allows a routing protocol to "sneak a peek" or "tap" messages it would not normally see from the MAC layer. This function will possibly unfragment such packets and call the function registered by the routing protocol to do the "Peek".</p>
PromiscuousMessagePeekFunctionType	<p><a href="#"><code>NetworkIpGetPromiscuousMessagePeekFunction</code></a>(Node* node, int interfaceIndex)</p> <p>Returns the network-layer function which will promiscuously inspect packets. See <code>NetworkIpSneakPeekAtMacPacket()</code>.</p>
void	<p><a href="#"><code>NetworkIpSetPromiscuousMessagePeekFunction</code></a>(Node* node, PromiscuousMessagePeekFunctionType PeekFunctionPtr, int interfaceIndex)</p> <p>Sets the network-layer function which will promiscuously inspect packets. See <code>NetworkIpSneakPeekAtMacPacket()</code>.</p>
void	<p><a href="#"><code>NetworkIpReceiveMacAck</code></a>(Node* node, int interfaceIndex, const Message* msg, NodeAddress nextHop)</p> <p>MAC received an ACK, so call ACK handler function.</p>
MacLayerAckHandlerType	<p><a href="#"><code>NetworkIpGetMacLayerAckHandler</code></a>(Node* node, int interfaceIndex)</p> <p>Get MAC layer ACK handler</p>
void	<p><a href="#"><code>NetworkIpSetMacLayerAckHandler</code></a>(Node* node, MacLayerAckHandlerType macAckHandlerPtr, int interfaceIndex)</p> <p>Set MAC layer ACK handler</p>
void	<p><a href="#"><code>SendToUdp</code></a>(Node* node, Message* msg, TosType priority, NodeAddress sourceAddress, NodeAddress destinationAddress, int incomingInterfaceIndex)</p> <p>Sends a UDP packet to UDP in the transport layer. The source IP address, destination IP address, and priority of the packet are also sent.</p>
void	<p><a href="#"><code>SendToTcp</code></a>(Node* node, Message* msg, TosType priority, NodeAddress sourceAddress, NodeAddress destinationAddress, BOOL aCongestionExperienced)</p> <p>Sends a TCP packet to TCP in the transport layer. The source IP address, destination IP address, and priority of the packet are also sent..</p>
void	<p><a href="#"><code>SendToRsvp</code></a>(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.
void	<a href="#"><b>NetworkIpRemoveIpHeader</b></a> (Node* node, Message* msg, NodeAddress* sourceAddress, NodeAddress* destinationAddress, TosType* priority, unsigned char* protocol, unsigned* ttl)  Removes the IP header from a message while also returning all the fields of the header.
void	<a href="#"><b>AddIpOptionField</b></a> (Node* node, Message* msg, int optionCode, int optionSize)  Inserts an option field in the header of an IP packet.
void	<a href="#"><b>ExtractIpSourceAndRecordedRoute</b></a> (Message* msg, NodeAddress[] RouteAddresses, int* NumAddresses, int* RouteAddressIndex)  Retrieves a copy of the source and recorded route from the options field in the header.
RouterFunctionType	<a href="#"><b>NetworkIpGetRouterFunction</b></a> (Node* node, int interfaceIndex)  Get the router function pointer.
void	<a href="#"><b>NetworkIpSetRouterFunction</b></a> (Node* node, RouterFunctionType RouterFunctionPtr, int interfaceIndex)  Allows a routing protocol to set the "routing function" (one of its functions) which is called when a packet needs to be routed. NetworkIpSetRouterFunction() allows a routing protocol to define the routing function. The routing function is called by the network layer to ask the routing protocol to route the packet. The routing function is given the packet and its destination. The routing protocol can route the packet and set "PacketWasRouted" to TRUE; or not route the packet and set to FALSE. If the packet, was not routed, then the network layer will try to use the forwarding table or the source route the source route in the IP header. This function will also be given packets for the local node the routing protocols can look at packets for protocol reasons. In this case, the message should not be modified and PacketWasRouted must be set to FALSE.
void	<a href="#"><b>NetworkIpAddUnicastRoutingProtocolType</b></a> (Node* node, NetworkRoutingProtocolType routingProtocolType, int interfaceIndex)  Add unicast routing protocol type to interface.
void	<a href="#"><b>NetworkIpAddUnicastIntraRegionRoutingProtocolType</b></a> (Node* node, NetworkRoutingProtocolType routingProtocolType, int interfaceIndex)  Add unicast intra region routing protocol type to interface.
void*	<a href="#"><b>NetworkIpGetRoutingProtocol</b></a> (Node* node, NetworkRoutingProtocolType routingProtocolType)  Get routing protocol structure associated with routing protocol running on this interface.
NetworkRoutingProtocolType	<a href="#"><b>NetworkIpGetUnicastRoutingProtocolType</b></a> (Node* node, int interfaceIndex)

	Get unicast routing protocol type on this interface.
void	<a href="#"><code>NetworkIpSetHsrpOnInterface</code></a> (Node* node, int interfaceIndex)  To enable hsrp on a interface
BOOL	<a href="#"><code>NetworkIpIsHsrpEnabled</code></a> (Node* node, int interfaceIndex)  To test if any interface is hsrp enabled.
void	<a href="#"><code>NetworkIpAddNewInterface</code></a> (Node* node, NodeAddress interfaceIpAddress, int numHostBits, int* newInterfaceIndex, const NodeInput* nodeInput)  Add new interface to node.
void	<a href="#"><code>NetworkIpInitCpuQueueConfiguration</code></a> (Node* node, const NodeInput* nodeInput)  Initializes cpu queue parameters during startup.
void	<a href="#"><code>NetworkIpInitInputQueueConfiguration</code></a> (Node* node, const NodeInput* nodeInput, int interfaceIndex)  Initializes input queue parameters during startup.
void	<a href="#"><code>NetworkIpInitOutputQueueConfiguration</code></a> (Node* node, const NodeInput* nodeInput, int interfaceIndex)  Initializes queue parameters during startup.
void	<a href="#"><code>NetworkIpCreateQueues</code></a> (Node* node, const NodeInput* nodeInput, int interfaceIndex)  Initializes input and output queue parameters during startup
void	<a href="#"><code>NetworkIpSchedulerParameterInit</code></a> (Scheduler* schedulerPtr, const int numPriorities, Queue* queue)  Initialize the scheduler parameters and also allocate memory for queues if require.
void	<a href="#"><code>NetworkIpSchedulerInit</code></a> (Node* node, const NodeInput* nodeInput, int interfaceIndex, Scheduler* schedulerPtr, const char* schedulerTypeString)  Call initialization function for appropriate scheduler.
void	<a href="#"><code>NetworkIpCpuQueueInsert</code></a> (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	<p><code>NetworkIpInputQueueInsert</code>(Node* node, int incomingInterface, Message* msg, NodeAddress nextHopAddress, 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><code>NetworkIpOutputQueueInsert</code>(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><code>NetworkIpInputQueueDequeuePacket</code>(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><code>NetworkIpOutputQueueDequeuePacket</code>(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><code>NetworkIpOutputQueueDequeuePacketForAPriority</code>(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><code>NetworkIpOutputQueueDequeuePacketWithIndex</code>(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><code>NetworkIpInputQueueTopPacket</code>(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><code>NetworkIpOutputQueueTopPacket</code>(Node* node, int interfaceIndex, Message** msg, NodeAddress* nextHopAddress, int* networkType, QueuePriorityType* priority)</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.
BOOL	<a href="#">NetworkIpOutputQueuePeekWithIndex</a> (Node* node, int interfaceIndex, int msgIndex, Message** msg, NodeAddress* nextHopAddress, QueuePriorityType* priority)
	Same as NetworkIpOutputQueueDequeuePacket(), except the packet is not actually dequeued. Note that the message containing the packet is not copied; the contents may (inadvertently or not) be directly modified. This function is called by MAC_OutputQueueTopPacket() (mac/mac.pc), which itself is called from mac/mac_802_11.pc and other MAC protocol source files. This function will assert false if the scheduler cannot return an IP packet for whatever reason.
BOOL	<a href="#">NetworkIpOutputQueueTopPacketForAPriority</a> (Node* node, int interfaceIndex, QueuePriorityType priority, Message** msg, NodeAddress* nextHopAddress, int posInQueue)
	Same as NetworkIpOutputQueueDequeuePacketForAPriority(), except the packet is not actually dequeued. Note that the message containing the packet is not copied; the contents may (inadvertently or not) be directly modified. This function is called by MAC_OutputQueueTopPacketForAPriority() (mac/mac.pc), which itself is called from mac/mac_802_11.pc and other MAC protocol source files. This function will assert false if the scheduler cannot return an IP packet for whatever reason.
BOOL	<a href="#">NetworkIpInputQueueIsEmpty</a> (Node* node, int incomingInterface)
	Calls the packet scheduler for an interface to determine whether the interface's input queue is empty
BOOL	<a href="#">NetworkIpOutputQueueIsEmpty</a> (Node* node, int interfaceIndex)
	Calls the packet scheduler for an interface to determine whether the interface's output queue is empty.
int	<a href="#">NetworkIpOutputQueueNumberInQueue</a> (Node* node, int interfaceIndex, BOOL specificPriorityOnly, QueuePriorityType priority)
	Calls the packet scheduler for an interface to determine how many packets are in a queue. There may be multiple queues on an interface, so the priority of the desired queue is also provided.
NodeAddress	<a href="#">NetworkIpOutputQueueDropPacket</a> (Node* node, int interfaceIndex, Message** msg)
	Drop a packet from the queue.
void	<a href="#">NetworkIpDeleteOutboundPacketsToANode</a> (Node* node, const NodeAddress nextHopAddress, const NodeAddress destinationAddress, const BOOL returnPacketsToRoutingProtocol)
	Deletes all packets in the queue going (probably broken), to the specified next hop address. There is option to return all such packets back to the routing protocols via the usual mechanism (callback).
unsigned	<a href="#">GetQueueNumberFromPriority</a> (TosType userTos, int numQueues)

	<p>Get queue number through which a given user priority will be forwarded.</p> <p><a href="#"><code>ReturnPriorityForPHB</code></a>(Node* node, TosType tos)</p>
QueuePriorityType	Returns the priority queue corresponding to the DS/TOS field.
void	<p><a href="#"><code>NetworkGetInterfaceAndNextHopFromForwardingTable</code></a>(Node* node, NodeAddress destinationAddress, int* interfaceIndex, NodeAddress* nextHopAddress)</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>
void	<p><a href="#"><code>NetworkGetInterfaceAndNextHopFromForwardingTable</code></a>(Node* node, int currentInterface, NodeAddress destinationAddress, int* interfaceIndex, NodeAddress* nextHopAddress)</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>
void	<p><a href="#"><code>NetworkGetInterfaceAndNextHopFromForwardingTable</code></a>(Node* node, 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>
void	<p><a href="#"><code>NetworkGetInterfaceAndNextHopFromForwardingTable</code></a>(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>
int	<p><a href="#"><code>NetworkIpGetInterfaceIndexForNextHop</code></a>(Node* node, NodeAddress nextHopAddress)</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. (used by NetworkUpdateForwardingTable() and ospfv2.pc)</p>
int	<p><a href="#"><code>NetworkGetInterfaceIndexForDestAddress</code></a>(Node* node, NodeAddress destAddress)</p> <p>Get interface for the destination address.</p>
NetworkRoutingAdminDistanceType	<p><a href="#"><code>NetworkRoutingGetAdminDistance</code></a>(Node* node, NetworkRoutingProtocolType type)</p> <p>Get the administrative distance of a routing protocol. These values don't quite match those recommended by Cisco.</p>
void	<p><a href="#"><code>NetworkInitForwardingTable</code></a>(Node* node)</p> <p>Initialize the IP fowarding table, allocate enough memory for number of rows.</p>

void	<a href="#"><code>NetworkUpdateForwardingTable</code></a> (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	<a href="#"><code>NetworkRemoveForwardingTableEntry</code></a> (Node* node, NodeAddress destAddress, NodeAddress destAddressMask, NodeAddress nextHopAddress, int outgoingInterfaceIndex)
	Remove single entries in the routing table
void	<a href="#"><code>NetworkEmptyForwardingTable</code></a> (Node* node, NetworkRoutingProtocolType type)
	Remove entries in the routing table corresponding to a given routing protocol.
void	<a href="#"><code>NetworkPrintForwardingTable</code></a> (Node* node)
	Display all entries in node's routing table.
int	<a href="#"><code>NetworkGetMetricForDestAddress</code></a> (Node* node, NodeAddress destAddress)
	Get the cost metric for a destination from the forwarding table.
void	<a href="#"><code>NetworkIpSetRouteUpdateEventFunction</code></a> (Node* node, NetworkRouteUpdateEventType routeUpdateFunctionPtr)
	Set a callback function when a route changes from forwarding table.
NetworkRouteUpdateEventType	<a href="#"><code>NetworkIpGetRouteUpdateEventFunction</code></a> (Node* node)
	Print packet headers when packet tracing is enabled.
NodeAddress	<a href="#"><code>NetworkIpGetInterfaceAddress</code></a> (Node* node, int interfaceIndex)
	Get the interface address on this interface
char*	<a href="#"><code>NetworkIpGetInterfaceName</code></a> (Node* node, int interfaceIndex)
	To get the interface name associated with the interface
NodeAddress	<a href="#"><code>NetworkIpGetInterfaceNetworkAddress</code></a> (Node* node, int interfaceIndex)
	To get network address associated with interface
NodeAddress	<a href="#"><code>NetworkIpGetInterfaceSubnetMask</code></a> (Node* node, int interfaceIndex)

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

	To join a multicast group
void	<a href="#"><code>NetworkIpJoinMulticastGroup</code></a> (Node* node, Int32 interfaceId, NodeAddress mcastAddr, clocktype delay, char[] filterMode, vector sourceList)
	To join a multicast group
void	<a href="#"><code>NetworkIpJoinMulticastGroup</code></a> (Node* node, NodeAddress mcastAddr, clocktype delay, char[] filterMode, vector sourceList)
	To join a multicast group
void	<a href="#"><code>NetworkIpLeaveMulticastGroup</code></a> (Node* node, NodeAddress mcastAddr, clocktype delay)
	To leave a multicast group
void	<a href="#"><code>NetworkIpLeaveMulticastGroup</code></a> (Node* node, NodeAddress mcastAddr, clocktype delay)
	To leave a multicast group
void	<a href="#"><code>NetworkIpSetMulticastTimer</code></a> (Node* node, long eventType, NodeAddress mcastAddr, clocktype delay)
	To set a multicast timer to join or leave multicast groups
void	<a href="#"><code>NetworkIpSetMulticastRoutingProtocol</code></a> (Node* node, void* multicastRoutingProtocol, int interfaceIndex)
	Assign a multicast routing protocol to an interface
void *	<a href="#"><code>NetworkIpGetMulticastRoutingProtocol</code></a> (Node* node, NetworkRoutingProtocolType routingProtocolType)
	To get the Multicast Routing Protocol structure
void	<a href="#"><code>NetworkIpAddMulticastProtocolType</code></a> (Node* node, NetworkRoutingProtocolType multicastProtocolType, int interfaceIndex)
	Assign a multicast protocol type to an interface
void	<a href="#"><code>NetworkIpSetMulticastRouterFunction</code></a> (Node* node, MulticastRouterFunctionType routerFunctionPtr, int interfaceIndex)
	Set a multicast router function to an interface
MulticastRouterFunctionType	<a href="#"><code>NetworkIpGetMulticastRouterFunction</code></a> (Node* node, int interfaceIndex)
	Get the multicast router function for an interface

void	<a href="#"><code>NetworkIpUpdateMulticastRoutingProtocolAndRouterFunction</code></a> (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	<a href="#"><code>NetworkIpUpdateUnicastRoutingProtocolAndRouterFunction</code></a> (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	<a href="#"><code>NetworkIpGetInterfaceIndexFromAddress</code></a> (Node* node, NodeAddress address)
	Get the interface index from an IP address.
int	<a href="#"><code>NetworkIpGetInterfaceIndexFromSubnetAddress</code></a> (Node* node, NodeAddress address)
	Get the interface index from an IP subnet address.
BOOL	<a href="#"><code>NetworkIpIsMulticastAddress</code></a> (Node* node, NodeAddress address)
	Check if an address is a multicast address.
void	<a href="#"><code>NetworkInitMulticastForwardingTable</code></a> (Node* node)
	initialize the multicast fowarding table, allocate enough memory for number of rows, used by ip
void	<a href="#"><code>NetworkEmptyMulticastForwardingTable</code></a> (Node* node)
	empty out all the entries in the multicast forwarding table. basically set the size of table back to 0.
LinkedList*	<a href="#"><code>NetworkGetOutgoingInterfaceFromMulticastForwardingTable</code></a> (Node* node, NodeAddress sourceAddress, NodeAddress groupAddress)
	get the interface Id node that lead to the (source, multicast group) pair.
void	<a href="#"><code>NetworkUpdateMulticastForwardingTable</code></a> (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	<a href="#"><code>NetworkPrintMulticastForwardingTable</code></a> (Node* node)
	display all entries in multicast forwarding table of the node.

void	<a href="#"><code>NetworkPrintMulticastOutgoingInterface</code></a> (Node* node, list* list)  Print multicast outgoing interfaces.
BOOL	<a href="#"><code>NetworkInMulticastOutgoingInterface</code></a> (Node* node, List* list, int interfaceIndex)  Determine if interface is in multicast outgoing interface list.
void	<a href="#"><code>NetworkIpPrintTraceXML</code></a> (Node* node, Message* msg)  Print packet trace information in XML format.
void	<a href="#"><code>RouteThePacketUsingLookupTable</code></a> (Node* node, Message* msg, int incomingInterface)  Tries to route and send the packet using the node's forwarding table.
int	<a href="#"><code>GetNetworkIPFragUnit</code></a> (Node* node, int interfaceIndex)  Returns the network ip fragmentation unit.
void	<a href="#"><code>NetworkIpUserProtocolInit</code></a> (Node* node, const NodeInput* nodeInput, const char* routingProtocolString, NetworkRoutingProtocolType* routingProtocolType, void** routingProtocolData)  Initialization of user protocol(disabled)
void	<a href="#"><code>NetworkIpUserHandleProtocolEvent</code></a> (Node* node, Message* msg)  Event handler function of user protocol(disabled)
void	<a href="#"><code>NetworkIpUserHandleProtocolPacket</code></a> (Node* node, Message* msg, unsigned char ipProtocol, NodeAddress sourceAddress, NodeAddress destinationAddress, int ttl)  Process a user protocol generated control packet(disabled)
void	<a href="#"><code>NetworkIpUserProtocolFinalize</code></a> (Node* node, int userProtocolNumber)  Finalization of user protocol(disabled)
void	<a href="#"><code>Atm_RouteThePacketUsingLookupTable</code></a> (Node* node, NodeAddress* destAddr, int* outIntf, NodeAddress* nextHop)  Routing packet received at ATM node
void	<a href="#"><code>RouteThePacketUsingMulticastForwardingTable</code></a> (Node* node, Message* msg, int incomingInterface)  Tries to route the multicast packet using the multicast forwarding table.

int	<a href="#"><b>NETWORKIpRoutingInit</b></a> (Node * node, const NodeInput *nodeInput nodeInput)
	Initialization function for network layer. Initializes IP.
Int64	<a href="#"><b>NetworkIpGetBandwidth</b></a> (Node* node, int interfaceIndex)
	getting the bandwidth information
clocktype	<a href="#"><b>NetworkIpGetPropDelay</b></a> (Node* node, int interfaceIndex)
	getting the propagation delay information
BOOL	<a href="#"><b>NetworkIpInterfaceIsEnabled</b></a> (Node* node, int interfaceIndex)
	To check the interface is enabled or not?
BOOL	<a href="#"><b>NetworkIpIsWiredNetwork</b></a> (Node* node, int interfaceIndex)
	Determines if an interface is a wired interface.
BOOL	<a href="#"><b>NetworkIpIsPointToPointNetwork</b></a> (Node* node, int interfaceIndex)
	Determines if an interface is a point-to-point.
BOOL	<a href="#"><b>IsIPV4MulticastEnabledOnInterface</b></a> (Node* node, int interfaceIndex)
	To check if IPV4 Multicast is enabled on interface?
BOOL	<a href="#"><b>IsIPV4RoutingEnabledOnInterface</b></a> (Node* node, int interfaceIndex)
	To check if IPV4 Routing is enabled on interface?
NetworkProtocolType	<a href="#"><b>NetworkIpGetNetworkProtocolType</b></a> (Node* node, NodeAddress nodeId)
	Get Network Protocol Type for the node
NetworkType	<a href="#"><b>ResolveNetworkTypeFromSrcAndDestNodeId</b></a> (Node* node, NodeId sourceNodeId, NodeId destNodeId)
	Resolve the NetworkType from source and destination node id's.
BOOL	<a href="#"><b>NetworkIpIsWiredBroadcastNetwork</b></a> (Node* node, int interfaceIndex)
	Determines if an interface is a wired interface.
ip_traceroute*	<a href="#"><b>FindTraceRouteOption</b></a> (const IpHeaderType* ipHeader)

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

### Constant / Data Structure Detail

Constant	IPVERSION4 4  Version of IP
Constant	IPTOS_LOWDELAY 0x10  Type of service ( low delay)
Constant	IPTOS_THROUGHPUT 0x08  Type of service ( throughput)
Constant	IPTOS_RELIABILITY 0x04  Type of service (reliability)
Constant	IPTOS_MINCOST 0x02  Type of service ( minimum cost)
Constant	IPTOS_ECT 0x02  Bits 6 and 7 in the IPv4 TOS octet are designated as the ECN field. Bit 6 is designated as the ECT bit.
Constant	IPTOS_CE 0x01  Bits 6 and 7 in the IPv4 TOS octet are designated as the ECN field. Bit 7 is designated as the CE bit.
Constant	IPTOS_DSCP_MAX 0x3f  Bits 0 to 5 in the IPv4 TOS octet are designated as DSCP field.The range for this 6-bit field is < 0 - 63 >.
Constant	IPTOS_DSCP_MIN 0x00

	Bits 0 to 5 in the IPv4 TOS octet are designated as DSCP field. The range for this 6-bit field is < 0 - 63 >.
Constant	IPTOS_PREC_EFINTERNETCONTROL 0xb8  IP precedence 'EF classe internet control'
Constant	IPTOS_PREC_NETCONTROL 0xe0  IP precedence 'net control'
Constant	IPTOS_PREC_INTERNETCONTROL 0xc0  IP precedence 'internet control'
Constant	IPTOS_PREC_CRITIC_ECP 0xa0  IP precedence 'critic ecp'
Constant	IPTOS_PREC_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 <code>ipt_flg</code> (specified modules only);
Constant	<p><code>IPOPT_SECUR_UNCLASS</code> 0x0000</p> <p>'unclass' bits for security in IP option field</p>
Constant	<p><code>IPOPT_SECUR_CONFID</code> 0xf135</p> <p>'confid' bits for security in IP option field</p>
Constant	<p><code>IPOPT_SECUR_EFTO</code> 0x789a</p> <p>'efto' bits for security in IP option field</p>
Constant	<p><code>IPOPT_SECUR_MMMM</code> 0xbc4d</p> <p>'mmmm' bits for security in IP option field</p>
Constant	<p><code>IPOPT_SECUR_RESTR</code> 0xaf13</p> <p>'restr' bits for security in IP option field</p>
Constant	<p><code>IPOPT_SECUR_SECRET</code> 0xd788</p> <p>'secreat' bits for security in IP option field</p>
Constant	<p><code>IPOPT_SECUR_TOPSECRET</code> 0x6bc5</p> <p>'top secret' bits for security in IP option field</p>
Constant	<p><code>MAXTTL</code> 255</p> <p>Internet implementation parameters (maximum time to live (seconds) )</p>
Constant	<p><code>IPDEFTTL</code> 64</p> <p>Internet implementation parameters (default ttl, from RFC 1340 )</p>
Constant	<p><code>IPFRAGTTL</code> 60</p> <p>Internet implementation parameters (time to live for frags, slowhz )</p>
Constant	<code>IPTTLDEC</code> 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 OSPF 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 OSPF 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	IpMftcParameter

	Variables of the structure define a unique condition class
Structure	<p>IpMultiFieldTrafficConditionerInfo</p> <p>Structure used to store traffic condition.</p>
Structure	<p>IpOptionsHeaderType</p> <p>Structure of optional header for IP source route</p>
Structure	<p>NetworkIpStatsType</p> <p>Structure used to keep track of all stats of network layer.</p>
Structure	<p>NetworkForwardingTableRow</p> <p>Structure of an entity of forwarding table.</p>
Structure	<p>NetworkForwardingTable</p> <p>Structure of forwarding table.</p>
Structure	<p>IpInterfaceInfoType</p> <p>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.</p>
Structure	<p>ip_frag_data</p> <p>QualNet typedefs struct ip_frag_data to IpFragData. is a simple queue to hold fragmented packets.</p>
Structure	<p>Ipv6FragQueue</p> <p>Ipv6 fragment queue structure.</p>
Structure	<p>FragmetedMsg</p> <p>QualNet typedefs struct fragmeted_msg_struct to ip6q. struct fragmeted_msg_struct is a simple fragmented packets msg hold structure.</p>
Structure	NetworkDataIp;

	Main structure of network layer.
Structure	<p>IpReassemblyBufferType</p> <p>Structure of reassembly buffer</p>
Structure	<p>IpReassemblyBufferListCellType</p> <p>Structure of reassembly buffer cell listing</p>
Structure	<p>IpReassemblyBufferListType</p> <p>Structure of reassembly buffer list</p>
Structure	<p>AddressChangeType</p> <p>enumeration to define address change events by DHCP</p>

## 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 <b>IpHeaderSetVersion()</b> (UInt32* ip_v_hl_tos_len, unsigned int version)

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

	<p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>IpHeaderSetIpReserved()</b>	<p>void <b>IpHeaderSetIpReserved()</b> (UInt16* ipFragment, UInt16 ipReserved)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• ipFragment - The variable containing the value of</li> <li>• ipReserved - Input value for set operation</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>IpHeaderSetIpDontFrag()</b>	<p>void <b>IpHeaderSetIpDontFrag()</b> (UInt16* ipFragment, UInt16 dontFrag)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• ipFragment - The variable containing the value of</li> <li>• dontFrag - Input value for set operation</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>IpHeaderSetIpMoreFrag()</b>	<p>void <b>IpHeaderSetIpMoreFrag()</b> (UInt16* ipFragment, UInt16 moreFrag)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• ipFragment - The variable containing the value of</li> <li>• moreFrag - Input value for set operation</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>IpHeaderGetVersion()</b>	<p>unsigned int <b>IpHeaderGetVersion()</b> (UInt32 ip_v_hl_tos_len)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• ip_v_hl_tos_len - The variable containing the value of ip_v</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• unsigned int - None</li> </ul>
<b>IpHeaderGetHLen()</b>	<p>unsigned int <b>IpHeaderGetHLen()</b> (UInt32 ip_v_hl_tos_len)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• ip_v_hl_tos_len - The variable containing the value of ip_v</li> </ul>

	<p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>unsigned int</code> - None</li> </ul>
<b>IpHeaderGetTOS()</b>	<p>unsigned int <b>IpHeaderGetTOS()</b> (UInt32 ip_v_hl_tos_len)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>ip_v_hl_tos_len</code> - The variable containing the value of ip_v</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>unsigned int</code> - None</li> </ul>
<b>IpHeaderGetIpLength()</b>	<p>unsigned int <b>IpHeaderGetIpLength()</b> (UInt32 ip_v_hl_tos_len)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>ip_v_hl_tos_len</code> - The variable containing the value of ip_v</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>unsigned int</code> - None</li> </ul>
<b>IpHeaderGetIpFragOffset()</b>	<p>UInt16 <b>IpHeaderGetIpFragOffset()</b> (UInt16 ipFragment)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>ipFragment</code> - The variable containing the value of</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>UInt16</code> - None</li> </ul>
<b>IpHeaderGetIpDontFrag()</b>	<p>BOOL <b>IpHeaderGetIpDontFrag()</b> (UInt16 ipFragment)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>ipFragment</code> - The variable containing the value of</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>BOOL</code> - None</li> </ul>
<b>IpHeaderGetIpMoreFrag()</b>	<p>BOOL <b>IpHeaderGetIpMoreFrag()</b> (UInt16 ipFragment)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>ipFragment</code> - The variable containing the value of</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>BOOL</code> - None</li> </ul>

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

<b>ConvertNumHostBitsToSubnetMask</b>	<p>NodeAddress <b>ConvertNumHostBitsToSubnetMask</b> (int numHostBits)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• numHostBits - number of host bit.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• NodeAddress - subnetmask</li> </ul>
<b>ConvertSubnetMaskToNumHostBits</b>	<p>int <b>ConvertSubnetMaskToNumHostBits</b> (NodeAddress subnetMask)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• subnetMask - subnetmask.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• int - number of host bit.</li> </ul>
<b>MaskIpAddress</b>	<p>NodeAddress <b>MaskIpAddress</b> (NodeAddress address, NodeAddress mask)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• address - address of a node</li> <li>• mask - mask of subnet.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• NodeAddress - masked node address.</li> </ul>
<b>MaskIpAddressWithNumHostBits</b>	<p>NodeAddress <b>MaskIpAddressWithNumHostBits</b> (NodeAddress address, int numHostBits)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• address - address of a node.</li> <li>• numHostBits - number of host bit.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• NodeAddress - masked node address.</li> </ul>
<b>CalcBroadcastIpAddress</b>	<p>NodeAddress <b>CalcBroadcastIpAddress</b> (NodeAddress address, int numHostBits)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• address - address of a node.</li> <li>• numHostBits - number of host bit.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• NodeAddress - Broadcast address.</li> </ul>

<b>IsIpAddressInSubnet</b>  To check if a ip address belongs to a subnet.	<p><b>BOOL IsIpAddressInSubnet</b> (NodeAddress address, NodeAddress subnetAddress, int numHostbits)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>address</code> - address of a node.</li> <li>• <code>subnetAddress</code> - address of a subnet.</li> <li>• <code>numHostbits</code> - number of host bit.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>BOOL</code> - TRUE if ip address belongs to a subnet else FALSE.</li> </ul>
<b>NetworkIpAddHeader</b>  Add an IP packet header to a message. Just calls AddIpHeader.	<p><b>void NetworkIpAddHeader</b> (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"> <li>• <code>node</code> - Pointer to node.</li> <li>• <code>msg</code> - Pointer to message.</li> <li>• <code>sourceAddress</code> - Source IP address.</li> <li>• <code>destinationAddress</code> - Destination IP address.</li> <li>• <code>priority</code> - Currently a TosType.</li> <li>• <code>protocol</code> - IP protocol number.</li> <li>• <code>ttl</code> - Time to live.If 0, uses default</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>FindAnIpOptionField</b>  Searches the IP header for the option field with option code that matches optionKey, and returns a pointer to the option field header.	<p><b>IpOptionsHeaderType* FindAnIpOptionField</b> (const IpHeaderType* ipHeader, const int optionKey)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>ipHeader</code> - Pointer to an IP header.</li> <li>• <code>optionKey</code> - Option code for desired option field.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>IpOptionsHeaderType*</code> - to the header of the desired option field. NULL if no option fields, or the desired option field cannot be found.</li> </ul>
<b>NetworkIpPreInit</b>	<p><b>void NetworkIpPreInit</b> (Node* node)</p> <p>Parameters:</p>

<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"> <li>• node - pointer to node.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<p><b>NetworkIpInit</b></p> <p>Initialize IP variables, and all network-layer IP protocols..</p>	<p><b>void NetworkIpInit (Node* node, const NodeInput* nodeInput)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - pointer to node.</li> <li>• nodeInput - Pointer to node input.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<p><b>NetworkIpLayer</b></p> <p>Handle IP layer events, incoming messages and messages sent to itself (timers, etc.).</p>	<p><b>void NetworkIpLayer (Node* node, Message* msg)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Pointer to node.</li> <li>• msg - Pointer to message.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<p><b>NetworkIpFinalize</b></p> <p>Finalize function for the IP model. Finalize functions for all network-layer IP protocols are called here.</p>	<p><b>void NetworkIpFinalize (Node* node)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Pointer to node.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<p><b>NetworkIpReceivePacketFromTransportLayer</b></p> <p>Called by transport layer protocols (UDP, TCP) to send UDP datagrams and TCP segments using IP. Simply calls NetworkIpSendRawMessage().</p>	<p><b>void NetworkIpReceivePacketFromTransportLayer (Node* node, Message* msg, NodeAddress sourceAddress, NodeAddress destinationAddress, int outgoingInterface, TosType priority, unsigned char protocol, BOOL isEcnCapable)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Pointer to node.</li> <li>• msg - Pointer to message from transport</li> <li>• sourceAddress - Source IP address.</li> <li>• destinationAddress - Destination IP address.</li> <li>• outgoingInterface - outgoing interface to use to</li> </ul>

	<ul style="list-style-type: none"> <li>• <code>priority</code> - Priority of packet.</li> <li>• <code>protocol</code> - IP protocol number.</li> <li>• <code>isEcnCapable</code> - Is this node ECN capable?</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>NetworkIpSendRawMessage</b>	<p>Called by NetworkIpReceivePacketFromTransportLayer() to send to send UDP datagrams, TCP segments using IP. Also called by network-layer routing protocols (AODV, OSPF, etc.) to send IP packets. This function adds an IP header and calls RoutePacketAndSendToMac().</p> <p><code>void NetworkIpSendRawMessage (Node* node, Message* msg, NodeAddress sourceAddress, NodeAddress destinationAddress, int outgoingInterface, TosType priority, unsigned char protocol, unsigned ttl)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to node.</li> <li>• <code>msg</code> - Pointer to message with payload data</li> <li>• <code>sourceAddress</code> - Source IP address.</li> <li>• <code>destinationAddress</code> - Destination IP address.</li> <li>• <code>outgoingInterface</code> - outgoing interface to use to</li> <li>• <code>priority</code> - Priority of packet.</li> <li>• <code>protocol</code> - IP protocol number.</li> <li>• <code>ttl</code> - Time to live.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>NetworkIpSendRawMessageWithDelay</b>	<p>Same as NetworkIpSendRawMessage(), but schedules event after a simulation delay.</p> <p><code>void NetworkIpSendRawMessageWithDelay (Node* node, Message* msg, NodeAddress sourceAddress, NodeAddress destinationAddress, int outgoingInterface, TosType priority, unsigned char protocol, unsigned ttl, clocktype delay)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to node.</li> <li>• <code>msg</code> - Pointer to message with payload data</li> <li>• <code>sourceAddress</code> - Source IP address.</li> <li>• <code>destinationAddress</code> - Destination IP address.</li> <li>• <code>outgoingInterface</code> - outgoing interface to use to</li> <li>• <code>priority</code> - TOS of packet.</li> </ul>

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

	<ul style="list-style-type: none"> <li>• <code>ttl</code> - Time to live.</li> <li>• <code>interfaceIndex</code> - Index of outgoing interface.</li> <li>• <code>nextHop</code> - Next hop IP address.</li> <li>• <code>delay</code> - delay.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>NetworkIpSendPacketToMacLayer</b>	<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><b>void NetworkIpSendPacketToMacLayer (Node* node, Message* msg, int interfaceIndex, NodeAddress nextHop)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to node.</li> <li>• <code>msg</code> - Pointer to message with ip packet.</li> <li>• <code>interfaceIndex</code> - Index of outgoing interface.</li> <li>• <code>nextHop</code> - Next hop IP address.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>NetworkIpSendPacketOnInterface</b>	<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><b>void NetworkIpSendPacketOnInterface (Node* node, Message* msg, int incomingInterface, int outgoingInterface, NodeAddress nextHop)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to node.</li> <li>• <code>msg</code> - Pointer to message with ip packet.</li> <li>• <code>incomingInterface</code> - Index of incoming interface.</li> <li>• <code>outgoingInterface</code> - Index of outgoing interface.</li> <li>• <code>nextHop</code> - Next hop IP address.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>NetworkIpSendPacketToMacLayerWithDelay</b>	<p>Same as <code>NetworkIpSendPacketOnInterface()</code>, but schedules event after a simulation delay.</p> <p><b>void NetworkIpSendPacketToMacLayerWithDelay (Node* node, Message* msg, int interfaceIndex, NodeAddress nextHop, clocktype delay)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to node.</li> </ul>

	<ul style="list-style-type: none"> <li>• <code>msg</code> - Pointer to message with ip packet.</li> <li>• <code>interfaceIndex</code> - Index of outgoing interface.</li> <li>• <code>nextHop</code> - Next hop IP address.</li> <li>• <code>delay</code> - delay</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>NetworkIpSendPacketOnInterfaceWithDelay</b>	<p>Same as <code>NetworkIpSendPacketOnInterface()</code>, but schedules event after a simulation delay.</p> <p><code>void NetworkIpSendPacketOnInterfaceWithDelay (Node* node, Message* msg, int incommingInterface, int outgoingInterface, NodeAddress nextHop, clocktype delay)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to node.</li> <li>• <code>msg</code> - Pointer to message with ip packet.</li> <li>• <code>incommingInterface</code> - Index of incomming interface.</li> <li>• <code>outgoingInterface</code> - Index of outgoing interface.</li> <li>• <code>nextHop</code> - Next hop IP address.</li> <li>• <code>delay</code> - delay</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>NetworkIpSendRawPacketOnInterfaceWithDelay</b>	<p>Same as <code>NetworkIpSendPacketOnInterface()</code>, but schedules event after a simulation delay and denotes raw packet.</p> <p><code>void NetworkIpSendRawPacketOnInterfaceWithDelay (Node* node, Message* msg, int incommingInterface, int outgoingInterface, NodeAddress nextHop, clocktype delay)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to node.</li> <li>• <code>msg</code> - Pointer to message with ip packet.</li> <li>• <code>incommingInterface</code> - Index of incomming interface.</li> <li>• <code>outgoingInterface</code> - Index of outgoing interface.</li> <li>• <code>nextHop</code> - Next hop IP address.</li> <li>• <code>delay</code> - delay</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>NetworkIpSendPacketToMacLayerWithNewStrictSourceRoute</b>	<code>void NetworkIpSendPacketToMacLayerWithNewStrictSourceRoute (Node* node, Message* msg,</code>

	<p>Tacks on a new source route to an existing IP packet and then sends the packet to the MAC layer.</p> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• node - Pointer to node.</li> <li>• msg - Pointer to message with ip packet.</li> <li>• newRouteAddresses - Source route (address array).</li> <li>• numNewRouteAddresses - Number of array elements.</li> <li>• removeExistingRecordedRoute - Flag to indicate previous record</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>NetworkIpReceivePacketFromMacLayer</b>	<p>IP received IP packet from MAC layer. Updates the Stats database and then calls NetworkIpReceivePacket.</p> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• node - Pointer to node.</li> <li>• msg - Pointer to message with ip packet.</li> <li>• previousHopNodeId - nodeId of the previous hop.</li> <li>• interfaceIndex - Index of interface on which packet arrived.</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>NetworkIpReceivePacket</b>	<p>IP received IP packet. Determine whether the packet is to be delivered to this node, or needs to be forwarded. ipHeader-&gt;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><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• node - Pointer to node.</li> <li>• msg - Pointer to message with ip packet.</li> <li>• previousHopNodeId - nodeId of the previous hop.</li> <li>• interfaceIndex - Index of interface on which packet arrived.</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
	<b>NetworkIpNotificationOfPacketDrop</b>
	<p>void <b>NetworkIpNotificationOfPacketDrop</b> (Node* node, Message* msg, NodeAddress nextHopNodeAddres, int interfaceIndex)</p>

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

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

	<ul style="list-style-type: none"> <li>• <code>interfaceIndex</code> - Interface associated with ACK handler function</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>MacLayerAckHandlerType</code> - MAC acknowledgement function pointer</li> </ul>
<b>NetworkIpSetMacLayerAckHandler</b>	<p>Set MAC layer ACK handler</p> <p><code>void NetworkIpSetMacLayerAckHandler (Node* node, MacLayerAckHandlerType macAckHandlerPtr, int interfaceIndex)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to node.</li> <li>• <code>macAckHandlerPtr</code> - Callback function handling</li> <li>• <code>interfaceIndex</code> - Interface associated with the ACK handler</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>SendToUdp</b>	<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><code>void SendToUdp (Node* node, Message* msg, TosType priority, NodeAddress sourceAddress, NodeAddress destinationAddress, int incomingInterfaceIndex)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to node.</li> <li>• <code>msg</code> - Pointer to message with UDP packet.</li> <li>• <code>priority</code> - TOS of UDP packet.</li> <li>• <code>sourceAddress</code> - Source IP address.</li> <li>• <code>destinationAddress</code> - Destination IP address.</li> <li>• <code>incomingInterfaceIndex</code> - interface that received the packet</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>SendToTcp</b>	<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><code>void SendToTcp (Node* node, Message* msg, TosType priority, NodeAddress sourceAddress, NodeAddress destinationAddress, BOOL aCongestionExperienced)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to node.</li> <li>• <code>msg</code> - Pointer to message with TCP packet.</li> <li>• <code>priority</code> - TOS of TCP packet.</li> <li>• <code>sourceAddress</code> - Source IP address.</li> </ul>

	<ul style="list-style-type: none"> <li>• <code>destinationAddress</code> - Destination IP address.</li> <li>• <code>aCongestionExperienced</code> - Determine if congestion is</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>SendToRsvp</b>	<p>Sends a RSVP packet to RSVP in the transport layer. The source IP address, destination IP address, and priority of the packet are also sent.</p> <p><b>void <code>SendToRsvp</code> (Node* node, Message* msg, TosType priority, NodeAddress sourceAddress, NodeAddress destinationAddress, int interfaceIndex, unsigned ttl)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to node.</li> <li>• <code>msg</code> - Pointer to message with RSVP packet.</li> <li>• <code>priority</code> - TOS of UDP packet.</li> <li>• <code>sourceAddress</code> - Source IP address.</li> <li>• <code>destinationAddress</code> - Destination IP address.</li> <li>• <code>interfaceIndex</code> - incoming interface index.</li> <li>• <code>ttl</code> - Receiving TTL</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>NetworkIpRemoveIpHeader</b>	<p>Removes the IP header from a message while also returning all the fields of the header.</p> <p><b>void <code>NetworkIpRemoveIpHeader</code> (Node* node, Message* msg, NodeAddress* sourceAddress, NodeAddress* destinationAddress, TosType* priority, unsigned char* protocol, unsigned* ttl)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to node.</li> <li>• <code>msg</code> - Pointer to message</li> <li>• <code>sourceAddress</code> - Storage for source IP address.</li> <li>• <code>destinationAddress</code> - Storage for destination IP</li> <li>• <code>priority</code> - Storage for TosType.(values are</li> <li>• <code>protocol</code> - Storage for IP protocol number</li> <li>• <code>ttl</code> - Storage for time to live.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>AddIpOptionField</b>	<b>void <code>AddIpOptionField</code> (Node* node, Message* msg, int optionCode, int optionSize)</b>

	<p>Inserts an option field in the header of an IP packet.</p> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• node - Pointer to node.</li> <li>• msg - Pointer to message</li> <li>• optionCode - The option code</li> <li>• optionSize - Size of the option</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>ExtractIpSourceAndRecordedRoute</b> <p>Retrieves a copy of the source and recorded route from the options field in the header.</p>	<p><b>void ExtractIpSourceAndRecordedRoute (Message* msg, NodeAddress[] RouteAddresses, int* NumAddresses, int* RouteAddressIndex)</b></p> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• msg - Pointer to message with IP packet.</li> <li>• RouteAddresses - Storage for source/recorded route.</li> <li>• NumAddresses - Storage for size of RouteAddresses[] array.</li> <li>• RouteAddressIndex - The index of the first address of the</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>NetworkIpGetRouterFunction</b> <p>Get the router function pointer.</p>	<p><b>RouterFunctionType NetworkIpGetRouterFunction (Node* node, int interfaceIndex)</b></p> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• node - Pointer to node.</li> <li>• interfaceIndex - interface associated with router function</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>• RouterFunctionType - router function pointer.</li> </ul>
<b>NetworkIpSetRouterFunction</b> <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;</p>	<p><b>void NetworkIpSetRouterFunction (Node* node, RouterFunctionType RouterFunctionPtr, int interfaceIndex)</b></p> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• node - Pointer to node.</li> <li>• RouterFunctionPtr - Router function to set.</li> <li>• interfaceIndex - interface associated with router function.</li> </ul>

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

	<ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to node.</li> <li>• <code>interfaceIndex</code> - network interface for request.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>NetworkRoutingProtocolType</code> - The unicast routing protocol type.</li> </ul>
<b>NetworkIpSetHsrpOnInterface</b>	<p><b>void NetworkIpSetHsrpOnInterface (Node* node, int interfaceIndex)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to node.</li> <li>• <code>interfaceIndex</code> - network interface.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>NetworkIpIsHsrpEnabled</b>	<p><b>BOOL NetworkIpIsHsrpEnabled (Node* node, int interfaceIndex)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to node.</li> <li>• <code>interfaceIndex</code> - network interface.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>BOOL</code> - return TRUE if any one interface is hsrp enabled else return FALSE.</li> </ul>
<b>NetworkIpAddNewInterface</b>	<p><b>void NetworkIpAddNewInterface (Node* node, NodeAddress interfaceIpAddress, int numHostBits, int* newInterfaceIndex, const NodeInput* nodeInput)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to node.</li> <li>• <code>interfaceIpAddress</code> - Interface to add.</li> <li>• <code>numHostBits</code> - Number of host bits for the interface.</li> <li>• <code>newInterfaceIndex</code> - The interface number of the new interface.</li> <li>• <code>nodeInput</code> - Provides access to</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>NetworkIpInitCpuQueueConfiguration</b>	<p><b>void NetworkIpInitCpuQueueConfiguration (Node* node, const NodeInput* nodeInput)</b></p> <p>Parameters:</p>

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

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

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

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

	<ul style="list-style-type: none"> <li>• <code>networkType</code> - Type of network packet is using (IP,</li> <li>• <code>posInQueue</code> - Position of packet in Queue.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>BOOL</code> - TRUE if dequeued successfully, FALSE otherwise.</li> </ul>
<b>NetworkIpOutputQueueDequeuePacketWithIndex</b>	<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><b>BOOL NetworkIpOutputQueueDequeuePacketWithIndex</b> (<code>Node* node, int interfaceIndex, int msgIndex, Message** msg, NodeAddress* nextHopAddress, int* networkType</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to node.</li> <li>• <code>interfaceIndex</code> - index to interface .</li> <li>• <code>msgIndex</code> - index of packet.</li> <li>• <code>msg</code> - Storage for pointer to message</li> <li>• <code>nextHopAddress</code> - Storage for Packet's next hop address.</li> <li>• <code>networkType</code> - Type of network packet is using (IP,</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>BOOL</code> - TRUE if dequeued successfully, FALSE otherwise.</li> </ul>
<b>NetworkIpInputQueueTopPacket</b>	<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><b>BOOL NetworkIpInputQueueTopPacket</b> (<code>Node* node, int incomingInterface, Message** msg, NodeAddress* nextHopAddress, int* outgoingInterface, int* networkType, QueuePriorityType* priority</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to node.</li> <li>• <code>incomingInterface</code> - interface to get top packet from.</li> <li>• <code>msg</code> - Storage for pointer to message</li> <li>• <code>nextHopAddress</code> - Storage for Packet's next hop addr.</li> <li>• <code>outgoingInterface</code> - Used to determine where packet should go</li> <li>• <code>networkType</code> - Type of network packet is using (IP,</li> <li>• <code>priority</code> - Storage for priority</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>BOOL</code> - TRUE if there is a packet, FALSE otherwise.</li> </ul>
<b>NetworkIpOutputQueueTopPacket</b>	<p><b>BOOL NetworkIpOutputQueueTopPacket</b> (<code>Node* node, int interfaceIndex, Message** msg, NodeAddress* nextHopAddress, int* networkType, QueuePriorityType* priority</code>)</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><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• node - Pointer to node.</li> <li>• interfaceIndex - index to interface .</li> <li>• msg - Storage for pointer to message</li> <li>• nextHopAddress - Storage for Packet's next hop addr.</li> <li>• networkType - Type of network of the packet</li> <li>• priority - Storage for priority</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>• BOOL - TRUE if there is a packet, FALSE otherwise.</li> </ul>
<p><b>NetworkIpOutputQueuePeekWithIndex</b></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><b>BOOL NetworkIpOutputQueuePeekWithIndex</b> (Node* node, int interfaceIndex, int msgIndex, Message** msg, NodeAddress* nextHopAddress, QueuePriorityType* priority)</p> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• node - Pointer to node.</li> <li>• interfaceIndex - index to interface .</li> <li>• msgIndex - index to message .</li> <li>• msg - Storage for pointer to message</li> <li>• nextHopAddress - Storage for Packet's next hop addr.</li> <li>• priority - Storage for priority</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>• BOOL - TRUE if there is a packet, FALSE otherwise.</li> </ul>
<p><b>NetworkIpOutputQueueTopPacketForAPriority</b></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><b>BOOL NetworkIpOutputQueueTopPacketForAPriority</b> (Node* node, int interfaceIndex, QueuePriorityType priority, Message** msg, NodeAddress* nextHopAddress, int posInQueue)</p> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• node - Pointer to node.</li> <li>• interfaceIndex - index to interface .</li> <li>• priority - priority of packet</li> <li>• msg - Storage for pointer to message</li> <li>• nextHopAddress - Storage for packet's next hop address.</li> <li>• posInQueue - Position of packet in Queue.</li> </ul>

	<p>Returns:</p> <ul style="list-style-type: none"> <li>• <b>BOOL</b> - TRUE if there is a packet, FALSE otherwise.</li> </ul>
<b>NetworkIpInputQueueIsEmpty</b>	<p><b>BOOL NetworkIpInputQueueIsEmpty</b> (Node* node, int incomingInterface)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <b>node</b> - Pointer to node.</li> <li>• <b>incomingInterface</b> - Index of interface.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <b>BOOL</b> - 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.</li> </ul>
<b>NetworkIpOutputQueueIsEmpty</b>	<p><b>BOOL NetworkIpOutputQueueIsEmpty</b> (Node* node, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <b>node</b> - Pointer to node.</li> <li>• <b>interfaceIndex</b> - Index of interface.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <b>BOOL</b> - 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.</li> </ul>
<b>NetworkIpOutputQueueNumberInQueue</b>	<p><b>int NetworkIpOutputQueueNumberInQueue</b> (Node* node, int interfaceIndex, BOOL specificPriorityOnly, QueuePriorityType priority)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <b>node</b> - Pointer to node.</li> <li>• <b>interfaceIndex</b> - Index of interface.</li> <li>• <b>specificPriorityOnly</b> - Should we only get the number of packets</li> <li>• <b>priority</b> - Priority of queue.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <b>int</b> - Number of packets in queue.</li> </ul>
<b>NetworkIpOutputQueueDropPacket</b>	<p><b>NodeAddress NetworkIpOutputQueueDropPacket</b> (Node* node, int interfaceIndex, Message** msg)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <b>node</b> - Pointer to node.</li> <li>• <b>interfaceIndex</b> - index to interface .</li> </ul>

	<ul style="list-style-type: none"> <li>• <code>msg</code> - Storage for pointer to message</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>NodeAddress</code> - Next hop of dropped packet.</li> </ul>
<b>NetworkIpDeleteOutboundPacketsToANode</b>	<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><b>void NetworkIpDeleteOutboundPacketsToANode</b> (<code>Node* node, const NodeAddress nextHopAddress, const NodeAddress destinationAddress, const BOOL returnPacketsToRoutingProtocol</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to node.</li> <li>• <code>nextHopAddress</code> - Next hop associated with</li> <li>• <code>destinationAddress</code> - destination associated with</li> <li>• <code>returnPacketsToRoutingProtocol</code> - Determine whether or not</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>GetQueueNumberFromPriority</b>	<p>Get queue number through which a given user priority will be forwarded.</p> <p><b>unsigned GetQueueNumberFromPriority</b> (<code>TosType userTos, int numQueues</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>userTos</code> - user TOS.</li> <li>• <code>numQueues</code> - Number of queues.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>unsigned</code> - Index of the queue.</li> </ul>
<b>ReturnPriorityForPHB</b>	<p>Returns the priority queue corresponding to the DS/TOS field.</p> <p><b>QueuePriorityType ReturnPriorityForPHB</b> (<code>Node* node, TosType tos</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to node.</li> <li>• <code>tos</code> - TOS field</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>QueuePriorityType</code> - priority queue</li> </ul>
<b>NetworkGetInterfaceAndNextHopFromForwardingTable</b>	<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><b>void NetworkGetInterfaceAndNextHopFromForwardingTable</b> (<code>Node* node, NodeAddress destinationAddress, int* interfaceIndex, NodeAddress* nextHopAddress</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to node.</li> </ul>

	<p><code>destinationAddress</code> - Destination IP address.</p> <ul style="list-style-type: none"> <li>• <code>interfaceIndex</code> - Storage for index of outgoing</li> <li>• <code>nextHopAddress</code> - Storage for next hop IP address.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>NetworkGetInterfaceAndNextHopFromForwardingTable</b>	<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><b>void NetworkGetInterfaceAndNextHopFromForwardingTable (Node* node, int currentInterface, NodeAddress destinationAddress, int* interfaceIndex, NodeAddress* nextHopAddress)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to node.</li> <li>• <code>currentInterface</code> - Current interface in use.</li> <li>• <code>destinationAddress</code> - Destination IP address.</li> <li>• <code>interfaceIndex</code> - Storage for index of outgoing</li> <li>• <code>nextHopAddress</code> - Storage for next hop IP address.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>NetworkGetInterfaceAndNextHopFromForwardingTable</b>	<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><b>void NetworkGetInterfaceAndNextHopFromForwardingTable (Node* node, NodeAddress destinationAddress, int* interfaceIndex, NodeAddress* nextHopAddress, BOOL testType, NetworkRoutingProtocolType type)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to node.</li> <li>• <code>destinationAddress</code> - Destination IP address.</li> <li>• <code>interfaceIndex</code> - Storage for index of outgoing</li> <li>• <code>nextHopAddress</code> - Storage for next hop IP address.</li> <li>• <code>testType</code> - Same protocol's routes if true</li> <li>• <code>type</code> - routing protocol type.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>NetworkGetInterfaceAndNextHopFromForwardingTable</b>	<p><b>void NetworkGetInterfaceAndNextHopFromForwardingTable (Node* node, int operatingInterface, NodeAddress destinationAddress, int* interfaceIndex, NodeAddress* nextHopAddress, BOOL testType, NetworkRoutingProtocolType type)</b></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><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• node - Pointer to node.</li> <li>• operatingInterface - interface currently being</li> <li>• destinationAddress - Destination IP address.</li> <li>• interfaceIndex - Storage for index of outgoing</li> <li>• nextHopAddress - Storage for next hop IP address.</li> <li>• testType - Same protocol's routes if true</li> <li>• type - routing protocol type.</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<p><b>NetworkIpGetInterfaceIndexForNextHop</b></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. (used by NetworkUpdateForwardingTable() and ospfv2.pc)</p>	<p><b>int NetworkIpGetInterfaceIndexForNextHop (Node* node, NodeAddress nextHopAddress)</b></p> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• node - Pointer to node.</li> <li>• nextHopAddress - Destination IP address.</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>• int - Index of outgoing interface, if nextHopAddress is on a directly connected network. -1, otherwise</li> </ul>
<p><b>NetworkGetInterfaceIndexForDestAddress</b></p> <p>Get interface for the destination address.</p>	<p><b>int NetworkGetInterfaceIndexForDestAddress (Node* node, NodeAddress destAddress)</b></p> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• node - Pointer to node.</li> <li>• destAddress - Destination IP address.</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>• int - interface index associated with destination.</li> </ul>
<p><b>NetworkRoutingGetAdminDistance</b></p> <p>Get the administrative distance of a routing protocol. These values don't quite match those recommended by Cisco.</p>	<p><b>NetworkRoutingAdminDistanceType NetworkRoutingGetAdminDistance (Node* node, NetworkRoutingProtocolType type)</b></p> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• node - Pointer to node.</li> <li>• type - Type value of routing protocol.</li> </ul> <p><b>Returns:</b></p>

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

<b>NetworkEmptyForwardingTable</b>	<p>void <b>NetworkEmptyForwardingTable</b> (Node* node, NetworkRoutingProtocolType type)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Pointer to node.</li> <li>• type - Type of routing protocol whose</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>NetworkPrintForwardingTable</b>	<p>void <b>NetworkPrintForwardingTable</b> (Node* node)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Pointer to node.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>NetworkGetMetricForDestAddress</b>	<p>int <b>NetworkGetMetricForDestAddress</b> (Node* node, NodeAddress destAddress)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Pointer to node.</li> <li>• destAddress - destination to get cost metric from.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• int - Cost metric associated with destination.</li> </ul>
<b>NetworkIpSetRouteUpdateEventFunction</b>	<p>void <b>NetworkIpSetRouteUpdateEventFunction</b> (Node* node, NetworkRouteUpdateEventType routeUpdateFunctionPtr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Pointer to node.</li> <li>• routeUpdateFunctionPtr - Route update</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>NetworkIpGetRouteUpdateEventFunction</b>	<p>NetworkRouteUpdateEventType <b>NetworkIpGetRouteUpdateEventFunction</b> (Node* node)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Pointer to node.</li> </ul> <p>Returns:</p>

	<ul style="list-style-type: none"> <li>• <code>NetworkRouteUpdateEventType</code> - Route update callback function to set.</li> </ul>
<b>NetworkIpGetInterfaceAddress</b>	<p>Get the interface address on this interface</p> <p>NodeAddress <b>NetworkIpGetInterfaceAddress</b> (Node* node, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to the node</li> <li>• <code>interfaceIndex</code> - Number of interface</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>NodeAddress</code> - IP address associated with the interface</li> </ul>
<b>NetworkIpGetInterfaceName</b>	<p>To get the interface name associated with the interface</p> <p>char* <b>NetworkIpGetInterfaceName</b> (Node* node, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to the node</li> <li>• <code>interfaceIndex</code> - Number of interface</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>char*</code> - interface name</li> </ul>
<b>NetworkIpGetInterfaceNetworkAddress</b>	<p>To get network address associated with interface</p> <p>NodeAddress <b>NetworkIpGetInterfaceNetworkAddress</b> (Node* node, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to the node</li> <li>• <code>interfaceIndex</code> - Number of interface</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>NodeAddress</code> - network address associated with interface</li> </ul>
<b>NetworkIpGetInterfaceSubnetMask</b>	<p>To retrieve subnet mask of the node interface</p> <p>NodeAddress <b>NetworkIpGetInterfaceSubnetMask</b> (Node* node, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to the node</li> <li>• <code>interfaceIndex</code> - Number of interface</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>NodeAddress</code> - subnet mask of the specified interface</li> </ul>
<b>NetworkIpGetInterfaceNumHostBits</b>	<p>Get the number of host bits on this interface</p> <p>int <b>NetworkIpGetInterfaceNumHostBits</b> (Node* node, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to the node</li> </ul>

	<ul style="list-style-type: none"> <li>• <code>interfaceIndex</code> - Number of interface</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>int</code> - Number of host bits on the specified interface</li> </ul>
<b>NetworkIpGetInterfaceBroadcastAddress</b>	<p>NodeAddress <b>NetworkIpGetInterfaceBroadcastAddress</b> (Node* node, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to the node</li> <li>• <code>interfaceIndex</code> - Number of interface</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>NodeAddress</code> - Broadcast address of specified interface</li> </ul>
<b>IsOutgoingBroadcast</b>	<p>BOOL <b>IsOutgoingBroadcast</b> (Node* node, NodeAddress destAddress, int* outgoingInterface, NodeAddress* outgoingBroadcastAddress)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to the node</li> <li>• <code>destAddress</code> - IP packet's destination IP address.</li> <li>• <code>outgoingInterface</code> - Outgoing interface index.</li> <li>• <code>outgoingBroadcastAddress</code> - Broadcast address</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>BOOL</code> - Returns true if destination is broadcast address</li> </ul>
<b>NetworkIpIsMyIP</b>	<p>BOOL <b>NetworkIpIsMyIP</b> (Node* node, NodeAddress ipAddress)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to the node</li> <li>• <code>ipAddress</code> - An IP packet's destination IP address.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>BOOL</code> - Returns if it belongs to it.</li> </ul>
<b>NetworkIpConfigurationError</b>	<p>void <b>NetworkIpConfigurationError</b> (Node* node, char [] parameterName, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to the node</li> <li>• <code>parameterName</code> - Error message to print</li> </ul>

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

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

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

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

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

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

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

	<ul style="list-style-type: none"> <li>• <code>BOOL</code> - TRUE if interface is in multicast outgoing interface list, FALSE otherwise.</li> </ul>
<b>NetworkIpPrintTraceXML</b>	<p>Print packet trace information in XML format.</p> <p><b>void NetworkIpPrintTraceXML (Node* node, Message* msg)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - this node</li> <li>• <code>msg</code> - Packet to print headers from.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>RouteThePacketUsingLookupTable</b>	<p>Tries to route and send the packet using the node's forwarding table.</p> <p><b>void RouteThePacketUsingLookupTable (Node* node, Message* msg, int incomingInterface)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - this node</li> <li>• <code>msg</code> - Pointer to message with IP packet.</li> <li>• <code>incomingInterface</code> - incoming interface of packet</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - NULL</li> </ul>
<b>GetNetworkIPFragUnit</b>	<p>Returns the network ip fragmentation unit.</p> <p><b>int GetNetworkIPFragUnit (Node* node, int interfaceIndex)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - this node</li> <li>• <code>interfaceIndex</code> - interface of node</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>int</code> - None</li> </ul>
<b>NetworkIpUserProtocolInit</b>	<p>Initialization of user protocol(disabled)</p> <p><b>void NetworkIpUserProtocolInit (Node* node, const NodeInput* nodeInput, const char* routingProtocolString, NetworkRoutingProtocolType* routingProtocolType, void** routingProtocolData)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - this node</li> <li>• <code>nodeInput</code> - Provides access to</li> <li>• <code>routingProtocolString</code> - routing protocol</li> <li>• <code>routingProtocolType</code> - routing protocol</li> <li>• <code>routingProtocolData</code> - Access to routing protocol data</li> </ul>

	<p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>NetworkIpUserHandleProtocolEvent</b>	<p>void <b>NetworkIpUserHandleProtocolEvent</b> (Node* node, Message* msg)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - The node that is handling the event.</li> <li>• msg - the event that is being handled</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>NetworkIpUserHandleProtocolPacket</b>	<p>void <b>NetworkIpUserHandleProtocolPacket</b> (Node* node, Message* msg, unsigned char ipProtocol, NodeAddress sourceAddress, NodeAddress destinationAddress, int ttl)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - this node</li> <li>• msg - message that is being received.</li> <li>• ipProtocol - ip protocol</li> <li>• sourceAddress - source address</li> <li>• destinationAddress - destination address</li> <li>• ttl - time to live</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>NetworkIpUserProtocolFinalize</b>	<p>void <b>NetworkIpUserProtocolFinalize</b> (Node* node, int userProtocolNumber)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - this node</li> <li>• userProtocolNumber - protocol number</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>Atm_RouteThePacketUsingLookupTable</b>	<p>void <b>Atm_RouteThePacketUsingLookupTable</b> (Node* node, NodeAddress* destAddr, int* outIntf, NodeAddress* nextHop)</p> <p>Parameters:</p>

	<ul style="list-style-type: none"> <li>• node - this node</li> <li>• destAddr - destination Address</li> <li>• outIntf - this node</li> <li>• nextHop - nextHop address</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>RouteThePacketUsingMulticastForwardingTable</b>	<p>Tries to route the multicast packet using the multicast forwarding table.</p> <p><b>void RouteThePacketUsingMulticastForwardingTable (Node* node, Message* msg, int incomingInterface)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - this node</li> <li>• msg - Pointer to Message</li> <li>• incomingInterface - Incomming Interface</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - NULL.</li> </ul>
<b>NETWORKIpRoutingInit</b>	<p>Initialization function for network layer. Initializes IP.</p> <p><b>int NETWORKIpRoutingInit (Node * node, const NodeInput *nodeInput nodeInput)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Pointer to node.</li> <li>• nodeInput - Pointer to node input.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• int - None</li> </ul>
<b>NetworkIpGetBandwidth</b>	<p>getting the bandwidth information</p> <p><b>Int64 NetworkIpGetBandwidth (Node* node, int interfaceIndex)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - the node who's bandwidth is needed.</li> <li>• interfaceIndex - interface Index.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• Int64 - inverted bandwidth ASSUMPTION : Bandwidth read from interface is in front of bps unit. To invert the bandwidth we use the equation <math>10000000 / \text{bandwidth}</math>. Where bandwidth is in Kbps unit.</li> </ul>
<b>NetworkIpGetPropDelay</b>	<p><b>clocktype NetworkIpGetPropDelay (Node* node, int interfaceIndex)</b></p> <p>Parameters:</p>

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

	<ul style="list-style-type: none"> <li>• BOOL - None</li> </ul>
<b>IsIPV4RoutingEnabledOnInterface</b>	<p>To check if IPV4 Routing is enabled on interface?</p> <p><b>BOOL IsIPV4RoutingEnabledOnInterface</b> (Node* node, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - node structure pointer.</li> <li>• interfaceIndex - interface Index.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• BOOL - None</li> </ul>
<b>NetworkIpGetNetworkProtocolType</b>	<p>Get Network Protocol Type for the node</p> <p><b>NetworkProtocolType NetworkIpGetNetworkProtocolType</b> (Node* node, NodeAddress nodeId)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - node structure pointer.</li> <li>• nodeId - node id.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• NetworkProtocolType - None</li> </ul>
<b>ResolveNetworkTypeFromSrcAndDestNodeId</b>	<p>Resolve the NetworkType from source and destination node id's.</p> <p><b>NetworkType ResolveNetworkTypeFromSrcAndDestNodeId</b> (Node* node, NodeId sourceNodeId, NodeId destNodeId)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Pointer to the node.</li> <li>• sourceNodeId - Source node id.</li> <li>• destNodeId - Destination node id.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• NetworkType - None</li> </ul>
<b>NetworkIpIsWiredBroadcastNetwork</b>	<p>Determines if an interface is a wired interface.</p> <p><b>BOOL NetworkIpIsWiredBroadcastNetwork</b> (Node* node, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - node structure pointer.</li> <li>• interfaceIndex - interface Index.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• BOOL - None</li> </ul>
<b>FindTraceRouteOption</b>	ip_traceroute* <b>FindTraceRouteOption</b> (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 [EXata Support](#) for questions pertaining to the EXata API Reference. This document is confidential and proprietary.  
It may not be reproduced or distributed without the expressed written consent of [SCALABLE Network Technologies](#).

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

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



# EXata 5.1 API Reference

## IPv6

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

### Constant / Data Structure Summary

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

	Authentication IPv6 Next header field value.
CONSTANT	<a href="#">IP6_NHDR_ESP</a>
	Encryption IPv6 Next header field value.
CONSTANT	<a href="#">IP6_NHDR_IPCP</a>
	Compression IPv6 Next header field value.
CONSTANT	<a href="#">IP6_NHDR_OSPF</a>
	Compression IPv6 Next header field value.
CONSTANT	<a href="#">IP6_NHDR_DOPT</a>
	Destination IPv6 Next header field value.
CONSTANT	<a href="#">IP6_NHDR_NONH</a>
	No next header IPv6 Next header field value.
CONSTANT	<a href="#">IPV6_FLOWINFO_VERSION</a>
	Flow information version.
CONSTANT	<a href="#">IPV6_VERSION</a>
	IPv6 version no.
CONSTANT	<a href="#">IP6_MMTU</a>
	Minimal MTU and reassembly.
CONSTANT	<a href="#">IPPROTO_ICMPV6</a>
	ICMPv6 protocol no.
CONSTANT	<a href="#">IP6ANY_ANycast</a>
	IPv6 anycast.
CONSTANT	<a href="#">ND_DEFAULT_HOPLIM</a>
	Node Discovery hop count.

CONSTANT	<a href="#">IP6_INSOPT_NOALLOC</a>	IPv6 insert option with no allocation.
CONSTANT	<a href="#">IP6_INSOPT_RAW</a>	IPv6 insert raw option.
CONSTANT	<a href="#">IP_FORWARDING</a>	IPv6 forwarding flag.
CONSTANT	<a href="#">IP6F_RESERVED_MASK</a>	Reserved fragment flag.
CONSTANT	<a href="#">IP_DF</a>	Don't fragment flag.
CONSTANT	<a href="#">IP6F_MORE_FRAG</a>	More fragments flag.
CONSTANT	<a href="#">IP6F_OFF_MASK</a>	Mask for fragmenting bits.
CONSTANT	<a href="#">IP6_FRAGTTL</a>	Time to live for frags.
CONSTANT	<a href="#">IP6_T_FLAG</a>	T Flag if set indicates transient multicast address.
CONSTANT	<a href="#">Multicast_Address_Scope_Related_constants.</a>	
CONSTANT	<a href="#">IP_FRAGMENT_HOLD_TIME</a>	IP Fragment hold time.
CONSTANT	<a href="#">IP_ROUTETOIF</a>	

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

	unreachable time <a href="#"><u>RETRANS_TIMER</u></a>
CONSTANT	retransmission timer
CONSTANT	<a href="#"><u>MAX_NEIGHBOR_ADVERTISEMENT</u></a>
	maximum neighbor advertisement
CONSTANT	<a href="#"><u>MAX_RTR_SOLICITATIONS</u></a>
	maximum Router Solicitations NOTE : Sending only one Solicitation; modify it once autoconfiguration supported.
CONSTANT	<a href="#"><u>MAX_MULTICAST_SOLICIT</u></a>
	maximum multicast solicitation
CONSTANT	<a href="#"><u>MAX_UNICAST_SOLICIT</u></a>
	maximum unicast solicitation
CONSTANT	<a href="#"><u>PKT_EXPIRE_DURATION</u></a>
	Packet expiration interval
CONSTANT	<a href="#"><u>INVALID_LINK_ADDR</u></a>
	Invalid Link Layer Address
CONSTANT	<a href="#"><u>MAX_HASHTABLE_SIZE</u></a>
	Maximum size of Hash-Table
CONSTANT	<a href="#"><u>MAX_REVLOOKUP_SIZE</u></a>
	Maximum Rev Look up hash table size
CONSTANT	<a href="#"><u>IPV6_HEADER_LENGTH</u></a>
	Length of IPv6 header
CONSTANT	<a href="#"><u>IP6_LSRRT</u></a>
	type 0

CONSTANT	<a href="#">IP6_NIMRT</a>
	type 1
CONSTANT	<a href="#">IP6_RT_MAX</a>
	Maximum number of addresses.
CONSTANT	<a href="#">IP6ANY_HOST_PROXY</a>
	proxy (host)
CONSTANT	<a href="#">IP6ANY_ROUTER_PROXY</a>
	proxy (router)
STRUCT	<a href="#">ip6_hdr_struct</a>
	QualNet typedefs struct ip6_hdr_struct to ip6_hdr. struct ip6_hdr_struct is 40 bytes, just like in the BSD code.
STRUCT	<a href="#">in6_multi_struct</a>
	QualNet typedefs struct in6_multi_struct to in6_multi. struct in6_multi_struct is just like in the BSD code.
STRUCT	<a href="#">ipv6_h2hhdr_struct</a>
	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	<a href="#">ipv6_rthdr_struct</a>
	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	<a href="#">ipv6_rthdr_struct</a>
	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	<a href="#">ip_moptions_struct</a>
	QualNet typedefs struct ip_moptions_struct to ip_moptions. struct ip_moptions_struct is multicast option structure, just like in the BSD code.
STRUCT	<a href="#">ip6_frag_struct</a>

	QualNet typedefs struct ip6_frag_struct to ipv6_fraghdr. struct ip6_frag_struct is fragmentation header structure. <a href="#">ip6Stat_struct</a>
STRUCT	QualNet typedefs struct ip6stat_struct to ip6Stat. struct ip6stat_struct is statistic information structure. <a href="#">Ipv6MulticastForwardingTableRow</a>
STRUCT	Structure of an entity of multicast forwarding table. <a href="#">Ipv6MulticastForwardingTable</a>
STRUCT	Structure of multicast forwarding table <a href="#">Ipv6MulticastGroupEntry</a>
STRUCT	Structure for Multicast Group Entry <a href="#">IPv6InterfaceInfo</a>
STRUCT	QualNet typedefs struct ipv6_interface_struct to IPv6InterfaceInfo. struct ipv6_interface_struct is interface information structure. <a href="#">messageBuffer</a>
STRUCT	QualNet typedefs struct messageBufferStruct to messageBuffer. struct messageBufferStruct is the buffer to hold messages when neighbour discovery is not done.
STRUCT	QualNet typedefs struct ip6q_struct to ip6q. struct ip6q is a simple queue to hold fragmented packets. <a href="#">Ipv6FragQueue</a>
STRUCT	Ipv6 fragment queue structure. <a href="#">FragmetedMsg</a>
STRUCT	QualNet typedefs struct fragmeted_msg_struct to ip6q. struct fragmeted_msg_struct is a simple fragmented packets msg hold structure. <a href="#">defaultRouterList</a>
STRUCT	default router list structure. <a href="#">destination_route_struct</a>

	QualNet typedefs struct destination_route_struct to destinationRoute. struct destination_route_struct is destination information structure of a node.
STRUCT	<a href="#">DestinationCache</a>  Destination cache entry structure
STRUCT	<a href="#">Ipv6HashData</a>  Ipv6 hash data structure.
STRUCT	<a href="#">Ipv6HashBlockData</a>  Ipv6 hash block-data structure.
STRUCT	<a href="#">Ipv6HashBlock</a>  Ipv6 hash block structure.
STRUCT	<a href="#">Ipv6HashTable</a>  Ipv6 hash table structure
STRUCT	<a href="#">IPv6Data</a>  QualNet typedefs struct ipv6_data_struct to IPv6Data. struct ipv6_data_struct is ipv6 information structure of a node.
STRUCT	<a href="#">ndpNadvEvent</a>  QualNet typedefs struct ndp_event_struct to IPv6Data. struct ndp_event_struct is neighbor advertisement information structure.

## Function / Macro Summary

Return Type	Summary
MACRO	<a href="#">ND_DEFAULT_CLASS(0xe0)</a>  Node Discovery sets class.
MACRO	<a href="#">NDP_DELAY</a>  NDP neighbor advertisement delay.

MACRO	<a href="#"><u>IPV6JITTER_RANGE</u></a>
	IPv6 jitter timer.
MACRO	<a href="#"><u>IPV6_SET_CLASS(hdr, priority)</u></a>
	Sets the flow class.
MACRO	<a href="#"><u>IPV6_GET_CLASS(hdr)</u></a>
	Gets the flow class.
void	<a href="#"><u>ip6_hdrSetVersion()</u></a> (UInt32 ipv6HdrVcf, UInt32 version)
	Set the value of version for ip6_hdr
void	<a href="#"><u>ip6_hdrSetClass()</u></a> (UInt32 ipv6HdrVcf, unsigned char ipv6Class)
	Set the value of class for ip6_hdr
void	<a href="#"><u>ip6_hdrSetFlow()</u></a> (UInt32 ipv6HdrVcf, UInt32 flow)
	Set the value of flow for ip6_hdr
UInt32	<a href="#"><u>ip6_hdrGetVersion()</u></a> (unsigned int ipv6HdrVcf)
	Returns the value of version for ip6_hdr
UInt32	<a href="#"><u>ip6_hdrGetClass()</u></a> (unsigned int ipv6HdrVcf)
	Returns the value of ip6_class for ip6_hdr
UInt32	<a href="#"><u>ip6_hdrGetFlow()</u></a> (unsigned int ipv6HdrVcf)
	Returns the value of ip6_flow for ip6_hdr
int	<a href="#"><u>in6_isanycast</u></a> (Node* node, in6_addr addr)
	Checks whether the address is anycast address of the node.
None	<a href="#"><u>Ipv6AddIpv6Header</u></a> (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	<a href="#"><b>Ipv6AddFragmentHeader</b></a> (Node *node node, Message *msg msg, unsigned char nextHeader, unsigned short offset, unsigned int id)
	Adds fragment header
None	<a href="#"><b>Ipv6RemoveIpv6Header</b></a> (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	<a href="#"><b>Ipv6PreInit</b></a> (Node* node)
	IPv6 Pre Initialization.
None	<a href="#"><b>IPv6Init</b></a> (Node* node, const NodeInput* nodeInput)
	IPv6 Initialization.
BOOL	<a href="#"><b>Ipv6IsMyPacket</b></a> (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	<a href="#"><b>Ipv6IsAddressInNetwork</b></a> (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	<a href="#"><b>Ipv6GetLinkLayerAddress</b></a> (Node* node, int interfaceId, char* ll_addr_str)
	Returns 32 bit link layer address of the interface.
None	<a href="#"><b>Ipv6AddNewInterface</b></a> (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	<a href="#"><b>Ipv6IsForwardingEnabled</b></a> (IPv6Data* ipv6)
	Checks whether the node is forwarding enabled.
None	<a href="#"><b>Ipv6Layer</b></a> (Node* node, Message* msg)
	Handle IPv6 layer events, incoming messages and messages sent to itself (timers, etc.).
None	<a href="#"><b>Ipv6Finalize</b></a> (Node* node)

	Finalize function for the IPv6 model. Finalize functions for all network-layer IPv6 protocols are called here.
int	<a href="#">Ipv6GetMTU</a> (Node* node, int interfaceId)
	Returns the maximum transmission unit of the interface.
int	<a href="#">Ipv6GetInterfaceIndexFromAddress</a> (Node* node, in6_addr* dst)
	Returns interface index of the specified address.
None	<a href="#">Ipv6CpuQueueInsert</a> (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	<a href="#">Ipv6InputQueueInsert</a> (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	<a href="#">Ipv6OutputQueueInsert</a> (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	<a href="#">QueueUpIpv6FragmentForMacLayer</a> (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	<a href="#">Ipv6SendPacketOnInterface</a> (Node* node, Message* msg, int incommingInterface, int outgoingInterface, NodeAddress nextHop)
	This function is called once the outgoing interface index and next hop address to which to route an IPv6 packet are known. This queues an IPv6 packet for delivery to the MAC layer. This functions calls QueueUpIpFragmentForMacLayer(). This function is used to initiate fragmentation if required, before calling the next function.
None	<a href="#">Ipv6SendOnBackplane</a> (Node* node, Message* msg, int incommingInterface, int outgoingInterface, NodeAddress hopAddr)
	This function is called when the packet delivered through backplane delay. required, before calling the next function.
None	<a href="#">Ipv6SendRawMessage</a> (Node* node, Message* msg, in6_addr sourceAddress, in6_addr destinationAddress, int outgoingInterface, TosType priority, unsigned char protocol, unsigned ttl)

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

	Function to assign all routers multicast address.
None	<a href="#">IPv6GetLinkLocalAddress</a> ( node, int interface, in6_addr* addr)
	Gets ipv6 link local address of the interface in output parameter addr.
None	<a href="#">IPv6GetSiteLocalAddress</a> ( node, int interface, in6_addr* addr)
	Gets ipv6 site local address of the interface in output parameter addr.
None	<a href="#">IPv6GetSiteLocalAddress</a> ( node, int interface, in6_addr* addr)
	Gets ipv6 global agreeable address of the interface in output parameter addr.
None	<a href="#">Ipv6GetPrefix</a> (in6_addr* addr, in6_addr* prefix)
	Gets ipv6 prefix from address.
Prefix for this interface	<a href="#">Ipv6GetPrefixFromInterfaceIndex</a> (Node* node, int interfaceIndex)
	Gets ipv6 prefix from address.
BOOL	<a href="#">Ipv6OutputQueueIsEmpty</a> (Node *node node, int interfaceIndex)
	Check weather output queue is empty
None	<a href="#">Ipv6RoutingStaticInit</a> (Node *node node, const NodeInput nodeInput, NetworkRoutingProtocolType type)
	Ipv6 Static routing initialization function.
None	<a href="#">Ipv6RoutingStaticEntry</a> (Node *node node, char currentLine[] currentLine)
	Static routing route entry function
None	<a href="#">Ipv6AddDestination</a> (Node* node node, route* ro ro)
	Adds destination in the destination cache.
None	<a href="#">Ipv6DeleteDestination</a> (Node* node node)
	Deletes destination from the destination cache.
int	<a href="#">Ipv6CheckForValidPacket</a> (Node* node node, SchedulerType* scheduler scheduler, unsigned int* pIndex pIndex)

	Checks the packet's validity <a href="#">Ipv6NdpProcessing</a> (Node* node node)
None	Ipv6 Destination cache and neighbor cache : processing function <a href="#">Ipv6UpdateForwardingTable</a> (Node* node node, in6_addr destPrefix destPrefix, in6_addr nextHopPrefix nextHopPrefix, int interfaceIndex, int metric metric)
None	Updates Ipv6 Forwarding Table <a href="#">Ipv6EmptyForwardingTable</a> (Node* node node, NetworkRoutingProtocolType type type)
None	Empties Ipv6 Forwarding Table for a particular routing protocol entry <a href="#">Ipv6PrintForwardingTable</a> (Node* node node)
Interface index associated with specified subnet address.	Prints the forwarding table. <a href="#">Ipv6InterfaceIndexFromSubnetAddress</a> (Node* node node, in6_addr* address)
void	Get the interface index from an IPv6 subnet address. <a href="#">Ipv6GetInterfaceAndNextHopFromForwardingTable</a> (Node* node node, in6_addr destAddr, int* interfaceIndex, in6_addr* nextHopAddr)
interface index associated with destination.	Do a lookup on the routing table with a destination IPv6 address to obtain an outgoing interface and a next hop Ipv6 address. <a href="#">Ipv6GetInterfaceIndexForDestAddress</a> (Node* node node, in6_addr destAddr)
interface index associated with destination.	Get interface for the destination address. <a href="#">Ipv6GetMetricForDestAddress</a> (Node* node node, in6_addr destAddr)
Interface index associated with destination if found,	Get the cost metric for a destination from the forwarding table. <a href="#">Ipv6IpGetInterfaceIndexForNextHop</a> (Node* node node, in6_addr destAddr)
Ipv6RouterFunctionType	This function looks at the network address of each of a node's network interfaces. When nextHopAddress is matched to a network, the interface index corresponding to the network is returned. <a href="#">Ipv6GetRouterFunction</a> (Node* node, int interfaceIndex)
void	Get the router function pointer. <a href="#">Ipv6SendPacketToMacLayer</a> (Node* node node, Message* msg, in6_addr destAddr, in6_addr* nextHopAddr, int* interfaceIndex)

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

	Create IPv6 Broadcast Address (ff02 followed by all one).
Prefix Length.	<a href="#">Ipv6GetPrefixLength()</a>
	Get prefix length of an interface.
void	<a href="#">Ipv6SetMacLayerStatusEventHandlerFunction</a> (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	<a href="#">Ipv6DeleteOutboundPacketsToANode</a> (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	<a href="#">Ipv6IsLoopbackAddress</a> (Node* node, in6_addr address)
	Check if address is self loopback address.
TRUE if my Ip, FALSE otherwise.	<a href="#">Ipv6IsMyIp</a> (Node* node, in6_addr* dst_addr)
	Check if address is self loopback address.
Scope value if valid multicast address, 0 otherwise.	<a href="#">Ipv6IsValidGetMulticastScope</a> (Node* node, in6_addr multiAddr)
	Check if multicast address has valid scope.
BOOL	<a href="#">IsIPV6RoutingEnabledOnInterface</a> (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 Length.
Constant	IP6_NHDR_HOP 0  Hop-by_hop IPv6 Next header field value.
Constant	IP6_NHDR_RT 43  Routing IPv6 Next header field value.
Constant	IP6_NHDR_FRAG 44  Fragment IPv6 Next header field value.
Constant	IP6_NHDR_AUTH 51  Authentication IPv6 Next header field value.
Constant	IP6_NHDR_ESP 50  Encryption IPv6 Next header field value.
Constant	IP6_NHDR_IPCP 108  Compression IPv6 Next header field value.
Constant	IP6_NHDR OSPF 89  Compression IPv6 Next header field value.
Constant	IP6_NHDR_DOPT 60

	Destination IPv6 Next header field value.
Constant	IP6_NHDR_NONH 59  No next header IPv6 Next header field value.
Constant	IPV6_FLOWINFO_VERSION 0x000000f0  Flow information version.
Constant	IPV6_VERSION 6  IPv6 version no.
Constant	IP6_MMTU 1280  Minimal MTU and reassembly.
Constant	IPPROTO_ICMPV6 58  ICMPv6 protocol no.
Constant	IP6ANY_ANycast 3  IPv6 anycast.
Constant	ND_DEFAULT_HOPLIM 255  Node Discovery hop count.
Constant	IP6_INSOPT_NOALLOC 1  IPv6 insert option with no allocation.
Constant	IP6_INSOPT_RAW 2  IPv6 insert raw option.
Constant	IP_FORWARDING 1  IPv6 forwarding flag.
Constant	IP6F_RESERVED_MASK 0x0600

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

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

	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	<p>IP6_RT_MAX_3</p> <p>Maximum number of addresses.</p>
Constant	<p>IP6ANY_HOST_PROXY_1</p> <p>proxy (host)</p>
Constant	<p>IP6ANY_ROUTER_PROXY_2</p> <p>proxy (router)</p>
Structure	<p>ip6_hdr_struct</p> <p>QualNet typedefs struct ip6_hdr_struct to ip6_hdr. struct ip6_hdr_struct is 40 bytes, just like in the BSD code.</p>
Structure	<p>in6_multi_struct</p> <p>QualNet typedefs struct in6_multi_struct to in6_multi. struct in6_multi_struct is just like in the BSD code.</p>
Structure	<p>ipv6_h2hhdr_struct</p> <p>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.</p>
Structure	<p>ipv6_rthdr_struct</p> <p>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.</p>
Structure	<p>ipv6_rthdr_struct</p> <p>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.</p>
Structure	<p>ip_moptions_struct</p> <p>QualNet typedefs struct ip_moptions_struct to ip_moptions. struct ip_moptions_struct is multicast option structure, just like in the BSD code.</p>
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

	<b>default router list structure.</b>
Structure	destination_route_struct  QualNet typedefs struct destination_route_struct to destinationRoute. struct destination_route_struct is destination information structure of a node.
Structure	DestinationCache  Destination cache entry structure
Structure	Ipv6HashData  Ipv6 hash data structure.
Structure	Ipv6HashBlockData  Ipv6 hash block-data structure.
Structure	Ipv6HashBlock  Ipv6 hash block structure.
Structure	Ipv6HashTable  Ipv6 hash table structure
Structure	IPv6Data  QualNet typedefs struct ipv6_data_struct to IPv6Data. struct ipv6_data_struct is ipv6 information structure of a node.
Structure	ndpNadvEvent  QualNet typedefs struct ndp_event_struct to IPv6Data. struct ndp_event_struct is neighbor advertisement information structure.

**Function / Macro Detail**

Function / Macro	Format
<b>ND_DEFAULT_CLASS(0xe0)</b>	Node Discovery sets class.

<b>NDP_DELAY</b>	NDP neighbor advertisement delay.
<b>IPV6JITTER_RANGE</b>	IPv6 jitter timer.
<b>IPV6_SET_CLASS(hdr, priority)</b>	Sets the flow class.
<b>IPV6_GET_CLASS(hdr)</b>	Gets the flow class.
<b>ip6_hdrSetVersion()</b>  Set the value of version for ip6_hdr	<p>void <b>ip6_hdrSetVersion()</b> (UInt32 ipv6HdrVcf, UInt32 version)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>ipv6HdrVcf</code> - The variable containing the value of <code>ip6_v,ip6_class</code></li> <li>• <code>version</code> - Input value for set operation</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - NULL.</li> </ul>
<b>ip6_hdrSetClass()</b>  Set the value of class for ip6_hdr	<p>void <b>ip6_hdrSetClass()</b> (UInt32 ipv6HdrVcf, unsigned char ipv6Class)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>ipv6HdrVcf</code> - The variable containing the value of <code>ip6_v,ip6_class</code></li> <li>• <code>ipv6Class</code> - Input value for set operation</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - NULL.</li> </ul>
<b>ip6_hdrSetFlow()</b>  Set the value of flow for ip6_hdr	<p>void <b>ip6_hdrSetFlow()</b> (UInt32 ipv6HdrVcf, UInt32 flow)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>ipv6HdrVcf</code> - The variable containing the value of <code>ip6_v,ip6_class</code></li> <li>• <code>flow</code> - Input value for set operation</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - NULL.</li> </ul>
<b>ip6_hdrGetVersion()</b>  Returns the value of version for ip6_hdr	<p>UInt32 <b>ip6_hdrGetVersion()</b> (unsigned int ipv6HdrVcf)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>ipv6HdrVcf</code> - The variable containing the value of <code>ip6_v,ip6_class</code></li> </ul>

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

	<p>Returns:</p> <ul style="list-style-type: none"> <li>• None - None</li> </ul>
<b>Ipv6AddFragmentHeader</b>	<p>Adds fragment header</p> <p>None <b>Ipv6AddFragmentHeader</b> (Node *node node, Message *msg msg, unsigned char nextHeader, unsigned short offset, unsigned int id)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Pointer to node</li> <li>• msg - Pointer to Message</li> <li>• nextHeader - nextHeader</li> <li>• offset - offset</li> <li>• id - id</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• None - None</li> </ul>
<b>Ipv6RemoveIpv6Header</b>	<p>Removes Ipv6 header</p> <p>None <b>Ipv6RemoveIpv6Header</b> (Node *node node, Message *msg msg, Address* sourceAddress, Address* destinationAddress destinationAddress, TosType *priority priority, unsigned char *protocol protocol, unsigned *hLim hLim)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Pointer to node</li> <li>• msg - Pointer to message</li> <li>• sourceAddress - Poineter Source address</li> <li>• destinationAddress - Destination address</li> <li>• priority - Priority</li> <li>• protocol - protocol</li> <li>• hLim - hLim</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• None - None</li> </ul>
<b>Ipv6PreInit</b>	<p>IPv6 Pre Initialization.</p> <p>None <b>Ipv6PreInit</b> (Node* node)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Pointer to node structure.</li> </ul> <p>Returns:</p>

	<ul style="list-style-type: none"> <li>• None - None</li> </ul>
<b>IPv6Init</b>	<p>None <b>IPv6Init</b> (Node* node, const NodeInput* nodeInput)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Pointer to node structure.</li> <li>• nodeInput - Node input.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• None - None</li> </ul>
<b>Ipv6IsMyPacket</b>	<p>BOOL <b>Ipv6IsMyPacket</b> (Node* node, in6_addr* dst_addr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Pointer to node structure.</li> <li>• dst_addr - ipv6 packet destination address.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• BOOL - None</li> </ul>
<b>Ipv6IsAddressInNetwork</b>	<p>BOOL <b>Ipv6IsAddressInNetwork</b> (const in6_addr* globalAddr, unsigned int tla, unsigned int vla, unsigned int sla)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• globalAddr - Pointer to ipv6 address.</li> <li>• tla - Top level ipv6 address.</li> <li>• vla - Next level ipv6 address.</li> <li>• sla - Site local ipv6 address.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• BOOL - None</li> </ul>
<b>Ipv6GetLinkLayerAddress</b>	<p>NodeAddress <b>Ipv6GetLinkLayerAddress</b> (Node* node, int interfaceId, char* ll_addr_str)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Pointer to node structure.</li> <li>• interfaceId - Interface Id.</li> <li>• ll_addr_str - Pointer to character link layer</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• NodeAddress - None</li> </ul>

<b>Ipv6AddNewInterface</b>	<p>Adds an ipv6 interface to the node.</p> <p>None <b>Ipv6AddNewInterface</b> (Node* node, in6_addr* globalAddr, unsigned int tla, unsigned int nla, unsigned int sla, int* newinterfaceIndex, const NodeInput* nodeInput)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Pointer to node structure.</li> <li>• globalAddr - Global ipv6 address pointer.</li> <li>• tla - Top level id.</li> <li>• nla - Next level id.</li> <li>• sla - Site level id.</li> <li>• newinterfaceIndex - Pointer to new interface index.</li> <li>• nodeInput - Node Input.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• None - None</li> </ul>
<b>Ipv6IsForwardingEnabled</b>	<p>Checks whether the node is forwarding enabled.</p> <p>BOOL <b>Ipv6IsForwardingEnabled</b> (IPv6Data* ipv6)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• ipv6 - Pointer to ipv6 data structure.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• BOOL - None</li> </ul>
<b>Ipv6Layer</b>	<p>Handle IPv6 layer events, incoming messages and messages sent to itself (timers, etc.).</p> <p>None <b>Ipv6Layer</b> (Node* node, Message* msg)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Pointer to node.</li> <li>• msg - Pointer to message.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• None - None</li> </ul>
<b>Ipv6Finalize</b>	<p>Finalize function for the IPv6 model. Finalize functions for all network-layer IPv6 protocols are called here.</p> <p>None <b>Ipv6Finalize</b> (Node* node)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Pointer to node.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• None - None</li> </ul>

<b>Ipv6GetMTU</b>  Returns the maximum transmission unit of the interface.	<p><b>int Ipv6GetMTU (Node* node, int interfaceId)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <b>node</b> - Pointer to node.</li> <li>• <b>interfaceId</b> - Interface Id.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <b>int</b> - None</li> </ul>
<b>Ipv6GetInterfaceIndexFromAddress</b>  Returns interface index of the specified address.	<p><b>int Ipv6GetInterfaceIndexFromAddress (Node* node, in6_addr* dst)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <b>node</b> - Pointer to node.</li> <li>• <b>dst</b> - IPv6 address.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <b>int</b> - None</li> </ul>
<b>Ipv6CpuQueueInsert</b>  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>None <b>Ipv6CpuQueueInsert (Node* node, Message* msg, NodeAddress nextHopAddress, in6_addr destinationAddress, int outgoingInterface, int networkType, BOOL* queueIsFull)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <b>node</b> - Pointer to node.</li> <li>• <b>msg</b> - Pointer to message with IPv6 packet.</li> <li>• <b>nextHopAddress</b> - Packet's next hop link layer address.</li> <li>• <b>destinationAddress</b> - Packet's destination address.</li> <li>• <b>outgoingInterface</b> - Used to determine where packet</li> <li>• <b>networkType</b> - Type of network packet is using (IPv6,</li> <li>• <b>queueIsFull</b> - Storage for boolean indicator.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <b>None</b> - None</li> </ul>
<b>Ipv6InputQueueInsert</b>  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	<p>None <b>Ipv6InputQueueInsert (Node* node, int incomingInterface, Message* msg, NodeAddress nextHopAddress, in6_addr destinationAddress, int outgoingInterface, int networkType, BOOL* queueIsFull)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <b>node</b> - Pointer to node.</li> </ul>

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

	<ul style="list-style-type: none"> <li>• <code>networkType</code> - Type of network packet is using (IPv6,</li> <li>• <code>queueIsFull</code> - Storage for boolean indicator.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>None</code> - None</li> </ul>
<b>Ipv6SendPacketOnInterface</b>	<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 <code>QueueUpIpFragmentForMacLayer()</code>. This function is used to initiate fragmentation if required, before calling the next function.</p> <p><code>None Ipv6SendPacketOnInterface (Node* node, Message* msg, int incommingInterface, int outgoingInterface, NodeAddress nextHop)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to node.</li> <li>• <code>msg</code> - Pointer to message with ip packet.</li> <li>• <code>incommingInterface</code> - Index of incoming interface.</li> <li>• <code>outgoingInterface</code> - Index of outgoing interface.</li> <li>• <code>nextHop</code> - Next hop link layer address.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>None</code> - None</li> </ul>
<b>Ipv6SendOnBackplane</b>	<p>This function is called when the packet delivered through backplane delay. required, before calling the next function.</p> <p><code>None Ipv6SendOnBackplane (Node* node, Message* msg, int incommingInterface, int outgoingInterface, NodeAddress hopAddr)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to node.</li> <li>• <code>msg</code> - Pointer to message with ip packet.</li> <li>• <code>incommingInterface</code> - Index of incomming interface.</li> <li>• <code>outgoingInterface</code> - Index of outgoing interface.</li> <li>• <code>hopAddr</code> - Next hop link layer address.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>None</code> - None</li> </ul>
<b>Ipv6SendRawMessage</b>	<p>Called by <code>NetworkIpReceivePacketFromTransportLayer()</code> to send to send UDP datagrams using IPv6. This function adds an IPv6 header and calls <code>RoutePacketAndSendToMac()</code>.</p> <p><code>None Ipv6SendRawMessage (Node* node, Message* msg, in6_addr sourceAddress, in6_addr destinationAddress, int outgoingInterface, TosType priority, unsigned char protocol, unsigned ttl)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to node.</li> <li>• <code>msg</code> - Pointer to message with payload data</li> </ul>

	<ul style="list-style-type: none"> <li>• <code>sourceAddress</code> - Source IPv6 address.</li> <li>• <code>destinationAddress</code> - Destination IPv6 address.</li> <li>• <code>outgoingInterface</code> - outgoing interface to use to</li> <li>• <code>priority</code> - Priority of packet.</li> <li>• <code>protocol</code> - IPv6 protocol number.</li> <li>• <code>ttl</code> - Time to live.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>None</code> - None</li> </ul>
<b>Ipv6SendToUdp</b>	<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><code>None Ipv6SendToUdp (Node* node, Message* msg, TosType priority, Address sourceAddress, Address destinationAddress, int incomingInterfaceIndex)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to node.</li> <li>• <code>msg</code> - Pointer to message with UDP packet.</li> <li>• <code>priority</code> - Priority of UDP</li> <li>• <code>sourceAddress</code> - Source IP address info.</li> <li>• <code>destinationAddress</code> - Destination IP address info.</li> <li>• <code>incomingInterfaceIndex</code> - interface that received the packet</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>None</code> - None</li> </ul>
<b>Ipv6SendToTCP</b>	<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><code>None Ipv6SendToTCP (Node* node, Message* msg, TosType priority, Address sourceAddress, Address destinationAddress, int incomingInterfaceIndex)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to node.</li> <li>• <code>msg</code> - Pointer to message with UDP packet.</li> <li>• <code>priority</code> - Priority of TCP</li> <li>• <code>sourceAddress</code> - Source IP address info.</li> <li>• <code>destinationAddress</code> - Destination IP address info.</li> <li>• <code>incomingInterfaceIndex</code> - interface that received the packet</li> </ul> <p>Returns:</p>

	<ul style="list-style-type: none"> <li>• None - None</li> </ul>
<b>Ipv6ReceivePacketFromMacLayer</b>	<p>None <b>Ipv6ReceivePacketFromMacLayer</b> (Node* node, Message* msg, NodeAddress previousHopNodeId, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Pointer to node.</li> <li>• msg - Pointer to message with ip packet.</li> <li>• previousHopNodeId - nodeId of the previous hop.</li> <li>• interfaceIndex - Index of interface on which packet arrived.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• None - None</li> </ul>
<b>Ipv6AddMessageInBuffer</b>	<p>BOOL <b>Ipv6AddMessageInBuffer</b> (Node* node, Message* msg, in6_addr* nextHopAddr, int inCommingInterface)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Pointer to node structure.</li> <li>• msg - Pointer to message with ip packet.</li> <li>• nextHopAddr - Source IPv6 address.</li> <li>• inCommingInterface - Incoming interface</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• BOOL - None</li> </ul>
<b>Ipv6DeleteMessageInBuffer</b>	<p>BOOL <b>Ipv6DeleteMessageInBuffer</b> (Node* node, messageBuffer* mBuf)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Pointer to node structure.</li> <li>• mBuf - Pointer to messageBuffer tail.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• BOOL - None</li> </ul>
<b>Ipv6DropMessageFromBuffer</b>	<p>void <b>Ipv6DropMessageFromBuffer</b> (Node* node, messageBuffer* mBuf)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Pointer to node structure.</li> <li>• mBuf - Pointer to messageBuffer tail.</li> </ul>

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

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

	<p>Returns:</p> <ul style="list-style-type: none"> <li>• Prefix for this interface - None</li> </ul>
<b>Ipv6OutputQueueIsEmpty</b>	<p>BOOL <b>Ipv6OutputQueueIsEmpty</b> (Node *node node, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Pointer to Node</li> <li>• interfaceIndex - interfaceIndex</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• BOOL - None</li> </ul>
<b>Ipv6RoutingStaticInit</b>	<p>None <b>Ipv6RoutingStaticInit</b> (Node *node node, const NodeInput nodeInput, NetworkRoutingProtocolType type)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Pointer to node</li> <li>• nodeInput - *nodeInput</li> <li>• type - type</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• None - None</li> </ul>
<b>Ipv6RoutingStaticEntry</b>	<p>None <b>Ipv6RoutingStaticEntry</b> (Node *node node, char currentLine[] currentLine)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Pointer to node</li> <li>• currentLine - Static entry's current line.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• None - None</li> </ul>
<b>Ipv6AddDestination</b>	<p>None <b>Ipv6AddDestination</b> (Node* node node, route* ro ro)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Pointer to node</li> <li>• ro - Pointer to destination route.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• None - None</li> </ul>

<b>Ipv6DeleteDestination</b>	<p>Deletes destination from the destination cache.</p> <p><b>None <code>Ipv6DeleteDestination</code> (Node* node node)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to node</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>None</code> - None</li> </ul>
<b>Ipv6CheckForValidPacket</b>	<p>Checks the packet's validity</p> <p><b>int <code>Ipv6CheckForValidPacket</code> (Node* node node, SchedulerType* scheduler scheduler, unsigned int* pIndex pIndex)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to node</li> <li>• <code>scheduler</code> - pointer to scheduler</li> <li>• <code>pIndex</code> - packet index</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>int</code> - None</li> </ul>
<b>Ipv6NdpProcessing</b>	<p>Ipv6 Destination cache and neighbor cache : processing function</p> <p><b>None <code>Ipv6NdpProcessing</code> (Node* node node)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to node</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>None</code> - None</li> </ul>
<b>Ipv6UpdateForwardingTable</b>	<p>Updates Ipv6 Forwarding Table</p> <p><b>None <code>Ipv6UpdateForwardingTable</code> (Node* node node, in6_addr destPrefix destPrefix, in6_addr nextHopPrefix nextHopPrefix, int interfaceIndex, int metric metric)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to node</li> <li>• <code>destPrefix</code> - IPv6 destination address</li> <li>• <code>nextHopPrefix</code> - IPv6 next hop address for this destination</li> <li>• <code>interfaceIndex</code> - interfaceIndex</li> <li>• <code>metric</code> - hop count between source and destination</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>None</code> - None</li> </ul>
<b>Ipv6EmptyForwardingTable</b>	<p><b>None <code>Ipv6EmptyForwardingTable</code> (Node* node node, NetworkRoutingProtocolType type type)</b></p>

	<p>Empties Ipv6 Forwarding Table for a particular routing protocol entry</p> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• node - Pointer to node</li> <li>• type - Routing protocol type</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>• None - None</li> </ul>
<b>Ipv6PrintForwardingTable</b>  Prints the forwarding table.	<p><b>None <b>I Pv6PrintForwardingTable</b> (Node* node node)</b></p> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• node - Pointer to node</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>• None - None</li> </ul>
<b>Ipv6InterfaceIndexFromSubnetAddress</b>  Get the interface index from an IPv6 subnet address.	<p>Interface index associated with specified subnet address. <b>Ipv6InterfaceIndexFromSubnetAddress</b> (Node* node node, in6_addr* address)</p> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• node - Pointer to node</li> <li>• address - Subnet Address</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>• Interface index associated with specified subnet address. - None</li> </ul>
<b>Ipv6GetInterfaceAndNextHopFromForwardingTable</b>  Do a lookup on the routing table with a destination IPv6 address to obtain an outgoing interface and a next hop IPv6 address.	<p><b>void <b>I Pv6GetInterfaceAndNextHopFromForwardingTable</b> (Node* node node, in6_addr destAddr, int* interfaceIndex, in6_addr* nextHopAddr)</b></p> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• node - Pointer to node</li> <li>• destAddr - Destination Address</li> <li>• interfaceIndex - Pointer to interface index</li> <li>• nextHopAddr - Next Hop Addr for destination.</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>• void - NULL.</li> </ul>
<b>Ipv6GetInterfaceIndexForDestAddress</b>  Get interface for the destination address.	<p>interface index associated with destination. <b>Ipv6GetInterfaceIndexForDestAddress</b> (Node* node node, in6_addr destAddr)</p> <p><b>Parameters:</b></p>

	<ul style="list-style-type: none"> <li>• node - Pointer to node</li> <li>• destAddr - Destination Address</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• interface index associated with destination. - None</li> </ul>
<b>Ipv6GetMetricForDestAddress</b>	interface index associated with destination. <b>Ipv6GetMetricForDestAddress</b> (Node* node node, in6_addr destAddr) <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Pointer to node</li> <li>• destAddr - Destination Address</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• interface index associated with destination. - None</li> </ul>
<b>Ipv6IpGetInterfaceIndexForNextHop</b>	Interface index associated with destination if found, <b>Ipv6IpGetInterfaceIndexForNextHop</b> (Node* node node, in6_addr destAddr) <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Pointer to node</li> <li>• destAddr - Destination Address</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• Interface index associated with destination if found, - None</li> </ul>
<b>Ipv6GetRouterFunction</b>	Ipv6RouterFunctionType <b>Ipv6GetRouterFunction</b> (Node* node, int interfaceIndex) <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Pointer to node.</li> <li>• interfaceIndex - interface associated with router function</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• Ipv6RouterFunctionType - router function pointer.</li> </ul>
<b>Ipv6SendPacketToMacLayer</b>	void <b>Ipv6SendPacketToMacLayer</b> (Node* node node, Message* msg, in6_addr destAddr, in6_addr* nextHopAddr, int* interfaceIndex) <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Pointer to node</li> <li>• msg - Pointer to message</li> <li>• destAddr - Destination Address</li> </ul>

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

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

	<ul style="list-style-type: none"> <li>• <code>list</code> - Pointer to Linked List.</li> <li>• <code>interfaceIndex</code> - Interface Index.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• TRUE if interface is in multicast outgoing interface - None</li> </ul>
<b>Ipv6UpdateMulticastForwardingTable</b>	<p>update entry with (sourceAddress, multicastGroupAddress) pair. search for the row with (sourceAddress, multicastGroupAddress) and update its interface.</p> <p><b>void Ipv6UpdateMulticastForwardingTable (Node* node, in6_addr sourceAddress, in6_addr multicastGroupAddress)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to node.</li> <li>• <code>sourceAddress</code> - Source Address.</li> <li>• <code>multicastGroupAddress</code> - multicast group.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - NULL.</li> </ul>
<b>Ipv6GetOutgoingInterfaceFromMulticastTable</b>	<p>get the interface List that lead to the (source, multicast group) pair.</p> <p>Interface List if match found, NULL otherwise. <b>Ipv6GetOutgoingInterfaceFromMulticastTable (Node* node, in6_addr sourceAddress, in6_addr groupAddress)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to node.</li> <li>• <code>sourceAddress</code> - Source Address</li> <li>• <code>groupAddress</code> - multicast group address</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• Interface List if match found, NULL otherwise. - None</li> </ul>
<b>Ipv6CreateBroadcastAddress</b>	<p>Create IPv6 Broadcast Address (ff02 followed by all one).</p> <p><b>void Ipv6CreateBroadcastAddress ()</b></p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - NULL.</li> </ul>
<b>Ipv6GetPrefixLength</b>	<p>Get prefix length of an interface.</p> <p>Prefix Length. <b>Ipv6GetPrefixLength ()</b></p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> <li>• Prefix Length. - None</li> </ul>

<b>Ipv6SetMacLayerStatusEventHandlerFunction</b>	<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><b>void Ipv6SetMacLayerStatusEventHandlerFunction (Node* node, Ipv6MacLayerStatusEventHandlerFunctionType StatusEventHandlerPtr, int interfaceIndex)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Pointer to node.</li> <li>• StatusEventHandlerPtr - Function Pointer</li> <li>• interfaceIndex - Interface Index</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - NULL.</li> </ul>
<b>Ipv6DeleteOutboundPacketsToANode</b>	<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><b>void Ipv6DeleteOutboundPacketsToANode (Node* node, const in6_addr nextHopAddress, const in6_addr destinationAddress, const BOOL returnPacketsToRoutingProtocol)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Pointer to node.</li> <li>• nextHopAddress - Next Hop Address.</li> <li>• destinationAddress - Destination Address</li> <li>• returnPacketsToRoutingProtocol - bool</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - NULL.</li> </ul>
<b>Ipv6IsLoopbackAddress</b>	<p>Check if address is self loopback address.</p> <p><b>void Ipv6IsLoopbackAddress (Node* node, in6_addr address)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Pointer to node.</li> <li>• address - ipv6 address</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - NULL.</li> </ul>
<b>Ipv6IsMyIp</b>	<p>TRUE if my Ip, FALSE otherwise. <b>Ipv6IsMyIp (Node* node, in6_addr* dst_addr)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Pointer to node.</li> <li>• dst_addr - Pointer to ipv6 address</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• TRUE if my Ip, FALSE otherwise. - None</li> </ul>

<b>Ipv6IsValidGetMulticastScope</b>  Check if multicast address has valid scope.	<p>Scope value if valid multicast address, 0 otherwise. <b>Ipv6IsValidGetMulticastScope</b> (Node* node, in6_addr multiAddr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Pointer to node.</li> <li>• multiAddr - multicast address.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• Scope value if valid multicast address, 0 otherwise. - None</li> </ul>
<b>IsIPV6RoutingEnabledOnInterface</b>  To check if IPV6 Routing is enabled on interface?	<p>BOOL <b>IsIPV6RoutingEnabledOnInterface</b> (Node* node, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - node structure pointer.</li> <li>• interfaceIndex - interface Index.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• BOOL - None</li> </ul>



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

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

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



# EXata 5.1 API Reference

## LIST

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

### Constant / Data Structure Summary

Type	Name
STRUCT	<a href="#">ListItem template</a> Structure for each item of a generic container list
STRUCT	<a href="#">List</a> A list that stores different types of structures.
STRUCT	<a href="#">IntList</a> A list that stores integers.

### Function / Macro Summary

Return Type	Summary
void	<a href="#">ListInit</a> (Node* node, LinkedList** list) Initialize the list
BOOL	<a href="#">ListIsEmpty</a> (Node* node, LinkedList* list) Check if list is empty
int	<a href="#">ListGetSize</a> (Node* node, LinkedList* list) Get the size of the list
void	<a href="#">ListInsert</a> (Node* node, LinkedList* list, clocktype timeStamp, void* data)

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

**Constant / Data Structure Detail**

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

**Function / Macro Detail**

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

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

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

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

Free the entire list

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

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

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



# EXata 5.1 API Reference

## MAC LAYER

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

### Constant / Data Structure Summary

Type	Name
CONSTANT	<a href="#">MAC_PROPAGATION_DELAY</a>  Peer to Peer Propagation delay in the MAC
CONSTANT	<a href="#">MAC_ADDRESS_LENGTH_IN_BYTE</a>  MAC address length
CONSTANT	<a href="#">Max_MacAddress_Length</a>  Maximum MAC address length
CONSTANT	<a href="#">MAC_ADDRESS_DEFAULT_LENGTH</a>  MAC address length in byte or octets
CONSTANT	<a href="#">MAC_CONFIGURATION_ATTRIBUTE</a>  Number of attribute of mac address file
CONSTANT	<a href="#">HW_TYPE_NETROM</a>  From KA9Q NET/ROM pseudo Hardware type.
CONSTANT	<a href="#">HW_TYPE_ETHER</a>  Ethernet 10/100Mbps Hardware type Ethernet.
CONSTANT	<a href="#">HW_TYPE_EETHER</a>

	Hardware type Experimental Ethernet
CONSTANT	<a href="#">HW_TYPE_AX25</a>
	Hardware type AX.25 Level 2
CONSTANT	<a href="#">HW_TYPE_PRONET</a>
	Hardware type PROnet token ring
CONSTANT	<a href="#">HW_TYPE_CHAOS</a>
	Hardware type Chaosnet
CONSTANT	<a href="#">HW_TYPE_IEEE802</a>
	IEEE 802.2 Ethernet/TR/TB
CONSTANT	<a href="#">HW_TYPE_ARCNET</a>
	Hardware type ARCnet
CONSTANT	<a href="#">HW_TYPE_APPLETALK</a>
	Hardware type APPLETalk
CONSTANT	<a href="#">HW_TYPE_DLCI</a>
	Frame Relay DLCI
CONSTANT	<a href="#">HW_TYPE_ATM</a>
	ATM 10/100Mbps
CONSTANT	<a href="#">HW_TYPE_METRICOM</a>
	Hardware type HW_TYPE_METRICOM
CONSTANT	<a href="#">HW_TYPE_IEEE_1394</a>
	Hardware type IEEE_1394
CONSTANT	<a href="#">HW_TYPE_EUI_64</a>
	Hardware identifier

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

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

	Fields for keeping track of interface faults
STRUCT	<p><a href="#">RandFault</a></p> <p>Structure containing random fault information.</p>

**Function / Macro Summary**

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

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

	To notify MAC of packet drop
void	<a href="#"><code>MAC_NotificationOfPacketDrop</code></a> (Node* node, Mac802Address nextHopAddress, int interfaceIndex, Message* msg)
	To notify MAC of packet drop
BOOL	<a href="#"><code>MAC_OutputQueueTopPacketForAPriority</code></a> (Node* node, int interfaceIndex, TosType priority, Message** msg, NodeAddress nextHopAddress)
	To notify MAC of priority packet arrival
BOOL	<a href="#"><code>MAC_OutputQueueTopPacketForAPriority</code></a> (Node* node, int interfaceIndex, TosType priority, Message** msg, Mac802Address* nextHopAddress)
	To notify MAC of priority packet arrival
BOOL	<a href="#"><code>MAC_OutputQueueTopPacketForAPriority</code></a> (Node* node, int interfaceIndex, TosType priority, Message** msg, MacHWAddress* nextHopAddress)
	To notify MAC of priority packet arrival
BOOL	<a href="#"><code>MAC_OutputQueueDequeuePacketForAPriority</code></a> (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	<a href="#"><code>MAC_OutputQueueDequeuePacketForAPriority</code></a> (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	<a href="#"><code>MAC_OutputQueueDequeuePacketForAPriority</code></a> (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	<a href="#"><code>MAC_OutputQueueDequeuePacketForAPriority</code></a> (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	<a href="#"><code>MAC_OutputQueueDequeuePacketForAPriority</code></a> (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	<a href="#"><code>MAC_SneakPeekAtMacPacket</code></a> (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	<a href="#"><b>MAC_SneakPeekAtMacPacket</b></a> (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	<a href="#"><b>MAC_SneakPeekAtMacPacket</b></a> (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	<a href="#"><b>MAC_MacLayerAcknowledgement</b></a> (Node* node, int interfaceIndex, Message* msg, NodeAddress nextHop)
	To send acknowledgement from MAC
void	<a href="#"><b>MAC_MacLayerAcknowledgement</b></a> (Node* node, int interfaceIndex, Message* msg, MacHWAddress& nextHop)
	To send acknowledgement from MAC
void	<a href="#"><b>MAC_MacLayerAcknowledgement</b></a> (Node* node, int interfaceIndex, Message* msg, Mac802Address& nextHop)
	To send acknowledgement from MAC
void	<a href="#"><b>MAC_HandOffSuccessfullyReceivedPacket</b></a> (Node* node, int interfaceIndex, Message* msg, NodeAddress lastHopAddress)
	Pass packet successfully up to the network layer
void	<a href="#"><b>MAC_HandOffSuccessfullyReceivedPacket</b></a> (Node* node, int interfaceIndex, Message* msg, MacHWAddress* lastHopAddr)
	Pass packet successfully up to the network layer
void	<a href="#"><b>MAC_HandOffSuccessfullyReceivedPacket</b></a> (Node* node, int interfaceIndex, Message* msg, Mac802Address* lastHopAddr)
	Pass packet successfully up to the network layer
void	<a href="#"><b>MAC_HandOffSuccessfullyReceivedPacket</b></a> (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	<a href="#"><b>MAC_HandOffSuccessfullyReceivedPacket</b></a> (Node* node, int interfaceIndex, Message* msg, Mac802Address* lastHopAddress, int arpMessageType)

	Pass packet successfully up to the network layer It is overloading function used for ARP packet
BOOL	<code>MAC_OutputQueueTopPacket</code> (Node* node, int interfaceIndex, Message** msg, NodeAddress* nextHopAddress, int networkType, TosType* priority)
	To check packet at the top of output queue
BOOL	<code>MAC_OutputQueueTopPacket</code> (Node* node, int interfaceIndex, Message** msg, MacHWAddress* nextHopAddress, int networkType, TosType* priority)
	To check packet at the top of output queue
BOOL	<code>MAC_OutputQueueTopPacket</code> (Node* node, int interfaceIndex, Message** msg, Mac802Address* nextHopAddress, int networkType, TosType* priority)
	To check packet at the top of output queue
BOOL	<code>MAC_OutputQueueDequeuePacket</code> (Node* node, int interfaceIndex, Message** msg, NodeAddress* nextHopAddress, int networkType, TosType * priority)
	To remove packet from front of output queue
BOOL	<code>MAC_OutputQueueDequeuePacket</code> (Node* node, int interfaceIndex, Message** msg, MacHWAddress* nextHopAddress, int networkType, TosType * priority)
	To remove packet from front of output queue
BOOL	<code>MAC_OutputQueueDequeuePacket</code> (Node* node, int interfaceIndex, Message** msg, Mac802Address* nextHopAddress, int networkType, TosType * priority)
	To remove packet from front of output queue, Its a overloaded function
BOOL	<code>MAC_OutputQueueDequeuePacket</code> (Node* node, int interfaceIndex, Message** msg, NodeAddress* nextHopAddress, int* networkType, TosType * priority, MacOutputQueueDequeueOption dequeueOption, MacOutputQueueDequeueCriteria dequeueCriteria, int * numFreeByte, int* numPacketPacked, TraceProtocolType tracePrt, BOOL eachWithMacHeader, int maxHeaderSize, BOOL returnPackedMsg)
	To remove packet(s) from front of output queue; process packets with options for example, pakcing multiple packets with same next hop address together
BOOL	<code>MAC_OutputQueueDequeuePacketForAPriority</code> (Node* node, int interfaceIndex, int priority, Message** msg, NodeAddress* nextHopAddress, int* networkType, MacOutputQueueDequeueOption dequeueOption, MacOutputQueueDequeueCriteria dequeueCriteria, int * numFreeByte, int* numPacketPacked, TraceProtocolType tracePrt, BOOL eachWithMacHeader, int maxHeaderSize, BOOL returnPackedMsg)
	To remove packet(s) from front of output queue; process packets with options for example, pakcing multiple packets with same next hop address together
BOOL	<code>MAC_IsMyUnicastFrame</code> (Node* node, NodeAddress destAddr)

	<p>Check if a packet (or frame) belongs to this node Should be used only for four byte mac address</p>
BOOL	<p>To check if an interface is a wired interface</p> <p><a href="#"><b>MAC_IsWiredNetwork</b>(Node* node, int interfaceIndex)</a></p>
BOOL	<p>Checks if an interface belongs to Point to PointNetwork</p> <p><a href="#"><b>MAC_IsPointToPointNetwork</b>(Node* node, int interfaceIndex)</a></p>
BOOL	<p>Checks if an interface belongs to Point to Multi-Point network.</p> <p><a href="#"><b>MAC_IsPointToMultiPointNetwork</b>(Node* node, int interfaceIndex)</a></p>
BOOL	<p>Determines if an interface is a wired broadcast interface</p> <p><a href="#"><b>MAC_IsWiredBroadcastNetwork</b>(Node* node, int interfaceIndex)</a></p>
BOOL	<p>Determine if a node's interface is a wireless interface</p> <p><a href="#"><b>MAC_IsWirelessNetwork</b>(Node* node, int interfaceIndex)</a></p>
BOOL	<p>Determine if a node's interface is a possible wireless ad hoc interface</p> <p><a href="#"><b>MAC_IsWirelessAdHocNetwork</b>(Node* node, int interfaceIndex)</a></p>
BOOL	<p>Determines if an interface is a single Hop Broadcast interface</p> <p><a href="#"><b>MAC_IsOneHopBroadcastNetwork</b>(Node* node, int interfaceIndex)</a></p>
BOOL	<p>To check if a node is a switch</p> <p><a href="#"><b>MAC_IsASwitch</b>(Node* node)</a></p>
void	<p>To set MAC address</p> <p><a href="#"><b>MAC_SetVirtualMacAddress</b>(Node* node, int interfaceIndex, NodeAddress virtualMacAddress)</a></p>
void	<p>Set Default interface Hardware Address of node</p> <p><a href="#"><b>MacSetDefaultHWAddress</b>(NodeId nodeId, MacHWAddress* macAddr, int interfaceIndex)</a></p>
NodeAddress	<p>To check if received mac address belongs to itself</p> <p><a href="#"><b>MAC_IsMyMacAddress</b>(Node* node, int interfaceIndex, NodeAddress destAddr)</a></p>

BOOL	<a href="#"><code>MAC_IsMyHWAddress</code></a> (Node* node, int interfaceIndex, MacAddress* macAddr)
	Checks for own MAC address.
void	<a href="#"><code>MacValidateAndSetHWAddress</code></a> (char* macAddrStr, MacHWAddress* macAddr)
	Validate MAC Address String after fetching from user
NodeAddress	<a href="#"><code>DefaultMacHWAddressToIpv4Address</code></a> (Node* node, MacHWAddress* macAddr)
	Retrieve the IP Address from Default HW Address . Default HW address is equal to 6 bytes
void	<a href="#"><code>MacGetHardwareLength</code></a> (Node* node, int interface, unsigned short hwLength)
	Retrieve the Hardware Length.
void	<a href="#"><code>MacGetHardwareType</code></a> (Node* node, int interface, unsigned short* type)
	Retrieve the Hardware Type.
void	<a href="#"><code>MacGetHardwareAddressString</code></a> (Node* node, int interface)
	Retrieve the Hardware Address String.
void	<a href="#"><code>MacAddNewInterface</code></a> (Node* node, NodeAddress interfaceAddress, int numHostBits, int* interfaceIndex, const NodeInput nodeInput, char* macProtocolName)
	To add a new Interface at MAC
void	<a href="#"><code>MacAddVlanInfoForThisInterface</code></a> (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	<a href="#"><code>MacReleaseVlanInfoForThisInterface</code></a> (Node* node, int interfaceIndex)
	To flush VLAN info for an interface
BOOL	<a href="#"><code>MAC_IsBroadcastHWAddress</code></a> (MacHWAddress* macAddr)
	Checks Broadcast MAC address
BOOL	<a href="#"><code>MAC_IsIdenticalHWAddress</code></a> (MacHWAddress* macAddr1, MacHWAddress* macAddr2)

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

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

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

**Constant / Data Structure Detail**

Constant	<code>MAC_PROPAGATION_DELAY 1 * MICRO_SECOND</code>  Peer to Peer Propogation delay in the MAC

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

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

	Length of 2 byte MacAddress
Constant	IPV4_LINKADDRESS 28  Hardware identifier
Constant	HW_NODE_ID 29  Hardware identifier
Constant	INVALID_MAC_ADDRESS MacHWAddress()  INVALID MAC ADDRESS
Constant	STATION_VLAN_TAGGING_DEFAULT FALSE  Default VLAN TAGGING Value for a STATION node
Enumeration	MacInterfaceState  Describes one out of two possible states of MAC interface - enable or disable
Enumeration	MacLinkType  Describes different link type
Enumeration	MAC_PROTOCOL  Specifies different MAC_PROTOCOLs used
Enumeration	MAC_SECURITY  Specifies different MAC_SECURITY_PROTOCOLS used
Enumeration	ManagementRequestType  Type of management request message
Enumeration	ManagementResponseType  Type of management response message
Enumeration	MacLinkType

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

**Function / Macro Detail**

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

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

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

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

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

To notify MAC of priority packet arrival

- `node` - Pointer to a network node
- `interfaceIndex` - interfaceIndex
- `priority` - Message Priority
- `msg` - Pointer to Message
- `nextHopAddress` - Next hop address

Returns:

- `BOOL` - TRUE if there is a packet, FALSE otherwise.

#### **MAC\_OutputQueueTopPacketForAPriority**

To notify MAC of priority packet arrival

`BOOL MAC_OutputQueueTopPacketForAPriority (Node* node, int interfaceIndex, TosType priority, Message** msg, Mac802Address* nextHopAddress)`

Parameters:

- `node` - Pointer to a network node
- `interfaceIndex` - interfaceIndex
- `priority` - Message Priority
- `msg` - Pointer to Message
- `nextHopAddress` - Next hop mac address

Returns:

- `BOOL` - TRUE if there is a packet, FALSE otherwise.

#### **MAC\_OutputQueueTopPacketForAPriority**

To notify MAC of priority packet arrival

`BOOL MAC_OutputQueueTopPacketForAPriority (Node* node, int interfaceIndex, TosType priority, Message** msg, MacHWAddress* nextHopAddress)`

Parameters:

- `node` - Pointer to a network node
- `interfaceIndex` - interfaceIndex
- `priority` - Message Priority
- `msg` - Pointer to Message
- `nextHopAddress` - Next hop mac address

Returns:

- `BOOL` - TRUE if there is a packet, FALSE otherwise.

#### **MAC\_OutputQueueDequeuePacketForAPriority**

`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

Parameters:

- `node` - Pointer to a network node
- `interfaceIndex` - interfaceIndex
- `priority` - Message Priority
- `msg` - Pointer to Message
- `nextHopAddress` - Next hop address
- `networkType` - network type

Returns:

- `BOOL` - TRUE if dequeued successfully, FALSE otherwise.

#### **MAC\_OutputQueueDequeuePacketForAPriority**

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)`

Parameters:

- `node` - Pointer to a network node
- `interfaceIndex` - interfaceIndex
- `priority` - Message Priority
- `msg` - Pointer to Message
- `nextHopAddress` - Next hop mac address
- `networkType` - network type

Returns:

- `BOOL` - TRUE if dequeued successfully, FALSE otherwise.

#### **MAC\_OutputQueueDequeuePacketForAPriority**

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)`

Parameters:

- `node` - Pointer to a network node
- `interfaceIndex` - interfaceIndex
- `priority` - Message Priority
- `msg` - Pointer to Message
- `nextHopAddress` - Next hop mac address

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

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

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

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

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

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

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

with options for example, packing multiple packets with same next hop address together

Parameters:

- `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

Returns:

- `BOOL` - TRUE if dequeued successfully, FALSE otherwise.

### **MAC\_OutputQueueDequeuePacketForAPriority**

To remove packet(s) from front of output queue; process packets with options for example, packing multiple packets with same next hop address together

`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)`

Parameters:

- `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

	<ul style="list-style-type: none"> <li>• <code>dequeueOption</code> - option</li> <li>• <code>dequeueCriteria</code> - criteria</li> <li>• <code>numFreeByte</code> - number of bytes can be packed in 1 transmission</li> <li>• <code>numPacketPacked</code> - number of packets packed</li> <li>• <code>tracePrt</code> - Trace Protocol Type</li> <li>• <code>eachWithMacHeader</code> - Each msg has its own MAC header?</li> <li>• <code>maxHeaderSize</code> - max mac header size</li> <li>• <code>returnPackedMsg</code> - return Packed msg or a list of msgs</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>BOOL</code> - TRUE if dequeued successfully, FALSE otherwise.</li> </ul>
<b>MAC_IsMyUnicastFrame</b>	<p>BOOL <b>MAC_IsMyUnicastFrame</b> (Node* node, NodeAddress destAddr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to a network node</li> <li>• <code>destAddr</code> - Destination Address</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>BOOL</code> - boolean</li> </ul>
<b>MAC_IsWiredNetwork</b>	<p>BOOL <b>MAC_IsWiredNetwork</b> (Node* node, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to a network node</li> <li>• <code>interfaceIndex</code> - interfaceIndex</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>BOOL</code> - boolean</li> </ul>
<b>MAC_IsPointToPointNetwork</b>	<p>BOOL <b>MAC_IsPointToPointNetwork</b> (Node* node, int interfaceIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to a network node</li> <li>• <code>interfaceIndex</code> - interfaceIndex</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>BOOL</code> - boolean</li> </ul>

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

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

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

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

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

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

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

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

	<ul style="list-style-type: none"> <li>• node - Node pointer</li> <li>• interfaceIndex - Interface index</li> <li>• msgIndex - Message index</li> <li>• msg - Double pointer to message</li> <li>• nextHopMacAddress - Next hop mac address</li> <li>• networkType - Type of network</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>BOOL</code> - TRUE if the message dequeued properly, FALSE otherwise</li> </ul>
<b>MAC_IPv4addressIsMulticastAddress</b>	<p><code>BOOL MAC_IPv4addressIsMulticastAddress (NodeAddress ipV4)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>ipV4</code> - ipV4 address</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>BOOL</code> - TRUE or FALSE</li> </ul>
<b>MAC_IsBroadcastMacAddress</b>	<p><code>BOOL MAC_IsBroadcastMacAddress (MacAddress* macAddr)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>macAddr</code> - Mac Address.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>BOOL</code> - None</li> </ul>
<b>IPv4AddressToDefaultMac802Address</b>	<p><code>void IPv4AddressToDefaultMac802Address (Node* node, int index, NodeAddress ipv4Address, Mac802Address* macAddr)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to Node structure</li> <li>• <code>index</code> - Interface Index</li> <li>• <code>ipv4Address</code> - Ipv4 address from which the</li> <li>• <code>macAddr</code> - Pointer to Mac802address structure</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - NULL</li> </ul>
<b>ConvertVariableHWAddressTo802Address</b>	<p><code>Bool ConvertVariableHWAddressTo802Address (Node* node, MacHWAddress* macHWAddr, Mac802Address* mac802Addr)</code></p>

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

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

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

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



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

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

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



# EXata 5.1 API Reference

## MAIN

This file contains some common definitions.

### Constant / Data Structure Summary

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

	This is a special addresses used in the MAC and network layers. It defines any source.
CONSTANT	<a href="#">ANY_IP</a>
	This is a special addresses used in the MAC and network layers. It defines any IP.
CONSTANT	<a href="#">ANY_INTERFACE</a>
	This is a special addresses used in the MAC and network layers. It defines any Interface.
CONSTANT	<a href="#">CPU_INTERFACE</a>
	This is a special addresses used in the MAC and network layers. It defines CPU Interface.
CONSTANT	<a href="#">INVALID_ADDRESS</a>
	It defines Invalid Address. Used only by mac/mac_802_11.c.
CONSTANT	<a href="#">MAX_STRING_LENGTH</a>
	Generic maximum length of a string. The maximum length of any line in the input file is 3x this value.
CONSTANT	<a href="#">BIG_STRING_LENGTH</a>
	maximum length of a string.
CONSTANT	<a href="#">MAX_CLOCK_STRING_LENGTH</a>
	Generic maximum length of a clock string.
CONSTANT	<a href="#">MAX_NW_PKT_SIZE</a>
	Defines the Maximum Network Packet Size which can handled by the physical network. In QualNet, its value is 2048. Packets larger than this will be fragmented by IP.
CONSTANT	<a href="#">MIN_NW_PKT_SIZE</a>
	Defines the Minimum Network Packet Size which can be handled by the physical network. In QualNet, its value is 40. Packets smaller than this will not have room for transport headers and most firewall-type devices will drop an initial fragment that does not contain enough data to hold the transport headers.
CONSTANT	<a href="#">MIN_IPv6_PKT_SIZE</a>
	Defines the Minimum Network Packet Size which can be handled by the IPv6 physical network. In QualNet, its value is 60. Packets smaller than this will not have room for transport headers and most firewall-type devices will drop an initial fragment that does not

	contain enough data to hold the transport headers. The additional space is to allow for IPv6's larger headers.
ENUMERATION	<a href="#">NetworkType</a>  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	<a href="#">in6_addr</a>  Describes the IPv6 address
STRUCT	<a href="#">AtmAddress</a>  Describes the ATM address
STRUCT	<a href="#">Address</a>  Describes the address structure which contains the interface address and network type

## Function / Macro Summary

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

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

UInt8	<a href="#">LshiftChar</a> (UInt8 x, UInt8 eposition)	Left shifts data where eposition determines the position of the last bit after the shift and (size-eposition) determines the number of bits to be shifted
UInt16	<a href="#">LshiftShort</a> (UInt16 x, UInt16 eposition)	Left shifts data where eposition determines the position of the last bit after the shift and (size-eposition) determines the number of bits to be shifted
UInt32	<a href="#">LshiftInt</a> (UInt32 x, int eposition)	Left shifts data where eposition determines the position of the last bit after the shift and (size-eposition) determines the number of bits to be shifted
UInt8	<a href="#">RshiftChar</a> (UInt8 x, UInt8 eposition)	Right shifts data where eposition determines the position of the last bit after the shift and (size-eposition) determines the number of bits to be shifted
UInt16	<a href="#">RshiftShort</a> (UInt16 x, UInt16 eposition)	Right shifts data where eposition determines the position of the last bit after the shift and (size-eposition) determines the number of bits to be shifted
UInt32	<a href="#">RshiftInt</a> (UInt32 x, int eposition)	Right shifts data where eposition determines the position of the last bit after the shift and (size-eposition) determines the number of bits to be shifted

## Constant / Data Structure Detail

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

	Length Field value for protocol IP TYPE
Constant	PROTOCOL_TYPE_ARP 0x0806  ARP type
Constant	ANY_DEST 0xffffffff  This is a special addresses used in the MAC and network layers. It defines any destination.
Constant	ANY_MAC802 0xffffffffffff  This is a special addresses used in the MAC and network layers. It defines any destination of six byte.
Constant	INVALID_802ADDRESS 0xfffffffffffffe  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	<code>BIG_STRING_LENGTH 512</code>  maximum length of a string.
Constant	<code>MAX_CLOCK_STRING_LENGTH 24</code>  Generic maximum length of a clock string.
Constant	<code>MAX_NW_PKT_SIZE 2048</code>  Defines the Maximum Network Packet Size which can be handled by the physical network. In QualNet, its value is 2048. Packets larger than this will be fragmented by IP.
Constant	<code>MIN_NW_PKT_SIZE 40</code>  Defines the Minimum Network Packet Size which can be handled by the physical network. In QualNet, its value is 40. Packets smaller than this will not have room for transport headers and most firewall-type devices will drop an initial fragment that does not contain enough data to hold the transport headers.
Constant	<code>MIN_IPv6_PKT_SIZE 60</code>  Defines the Minimum Network Packet Size which can be handled by the IPv6 physical network. In QualNet, its value is 60. Packets smaller than this will not have room for transport headers and most firewall-type devices will drop an initial fragment that does not contain enough data to hold the transport headers. The additional space is to allow for IPv6's larger headers.
Enumeration	<code>NetworkType</code>  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.
Structure	<code>in6_addr</code>  Describes the IPv6 address
Structure	<code>AtmAddress</code>  Describes the ATM address
Structure	<code>Address</code>

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

## Function / Macro Detail

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

address	<ul style="list-style-type: none"> <li>• <code>addr</code> - IPv4 interface address.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - NULL</li> </ul>
<b>SetIPv6AddressInfo</b>  Set IPv6 address and network type to generic address	<b>void SetIPv6AddressInfo (Address address, in6_addr addr)</b>  Parameters: <ul style="list-style-type: none"> <li>• <code>address</code> - generic address.</li> <li>• <code>addr</code> - IPv6 interface address.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - NULL</li> </ul>
<b>RoundToInt</b>  Round a float point number to an integer. This function tries to get consistent value on different platforms	<b>int RoundToInt (double x)</b>  Parameters: <ul style="list-style-type: none"> <li>• <code>x</code> - The float point number to be rounded</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>int</code> - Returns the rounded integer</li> </ul>
<b>MEM_malloc</b>  Allocates memory block of a given size.	<b>void* MEM_malloc (size_t size, char* filename, int lineno)</b>  Parameters: <ul style="list-style-type: none"> <li>• <code>size</code> - Size of the memory block to be allocated.</li> <li>• <code>filename</code> - Name of file allocating the memory</li> <li>• <code>lineno</code> - Line in the file where the API is called</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void*</code> - Returns the pointer of allocated memory otherwise NULL if allocation fails.</li> </ul>
<b>MEM_free</b>  Deallocates the memory in turn it calls free().	<b>void MEM_free (void* ptr)</b>  Parameters: <ul style="list-style-type: none"> <li>• <code>ptr</code> - Pointer of memory to be freed.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>maskChar</b>	<b>UInt8 maskChar (UInt8 sposition, UInt8 eposition)</b>

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

	<ul style="list-style-type: none"> <li>• <code>UInt16</code> - None</li> </ul>
<b>LshiftInt</b>	<p><code>UInt32 LshiftInt (UInt32 x, int eposition)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>x</code> - the data to be shifted</li> <li>• <code>eposition</code> - last bit position set to 1</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>UInt32</code> - None</li> </ul>
<b>RshiftChar</b>	<p><code>UInt8 RshiftChar (UInt8 x, UInt8 eposition)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>x</code> - the data to be shifted</li> <li>• <code>eposition</code> - last bit position set to 1</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>UInt8</code> - None</li> </ul>
<b>RshiftShort</b>	<p><code>UInt16 RshiftShort (UInt16 x, UInt16 eposition)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>x</code> - the data to be shifted</li> <li>• <code>eposition</code> - last bit position set to 1</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>UInt16</code> - None</li> </ul>
<b>RshiftInt</b>	<p><code>UInt32 RshiftInt (UInt32 x, int eposition)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>x</code> - the data to be shifted</li> <li>• <code>eposition</code> - last bit position set to 1</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>UInt32</code> - None</li> </ul>



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

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

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



# EXata 5.1 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	<a href="#">INVALID_MAPPING</a>
	Indicates Invalid Mapping
CONSTANT	<a href="#">MAX_INTERFACE_ADDRESSES</a>
	max no of addressees assigned to an interface
CONSTANT	<a href="#">NODE_HASH_SIZE</a>
	Defines node hash size. Hashes the nodeId using a mod NODE_HASH_SIZE hash.
STRUCT	<a href="#">NetworkProperty</a>
	Describes the property of a network.
STRUCT	<a href="#">AddressMappingType</a>
	Describes the type of address mapping.
STRUCT	<a href="#">AddressReverseMappingType</a>
	Describes the type of reverse address mapping.
STRUCT	<a href="#">SubnetListType</a>
	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	<a href="#">AddressMapType</a>

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

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

	<p>Set addr to any address of the same type as refAddr.</p>
void	<p><a href="#"><u>Address AddressCopy</u></a>(Address* dstAddress, Address* srcAddress)</p>
	<p>Copy srcAddress to dstAddress</p>
int	<p><a href="#"><u>Ipv6CompareAddr6</u></a>(in6_addr a, in6_addr b)</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>
BOOL	<p><a href="#"><u>Ipv6IsAddressInNetwork</u></a>(const in6_addr* globalAddr, unsigned int tla, unsigned int vla, unsigned int sla)</p>
	<p>Checks whether the address is in the same network. : if in the same network then returns TRUE, otherwise FALSE.</p>
BOOL	<p><a href="#"><u>Ipv6IsAddressInNetwork</u></a>(const in6_addr* globalAddr, const in6_addr* ipv6SubnetAddr, unsigned int prefixLenth)</p>
	<p>Checks whether the address is in the same network. : if in the same network then returns TRUE, otherwise FALSE.</p>
BOOL	<p><a href="#"><u>Ipv6CheckNetworkParams</u></a>(unsigned int tla tla, unsigned int nla nla, unsigned int sla sla)</p>
	<p>Checks network parameters (tla, nla, sla)</p>
void	<p><a href="#"><u>MAPPING_HashNodeId</u></a>(IdToNodePtrMap* hash, NodeAddress nodeId, Node* nodePtr)</p>
	<p>Hashes the nodeIds using a mod NODE_HASH_SIZE hash. This is not thread safe.</p>
Node*	<p><a href="#"><u>MAPPING_GetNodePtrFromHash</u></a>(IdToNodePtrMap* hash, NodeAddress nodeId)</p>
	<p>Retrieves the node pointer for nodeId from hash.</p>
AddressMapType*	<p><a href="#"><u>MAPPING_MallocAddressMap</u></a>( )</p>
	<p>Allocates memory block of size AddressMapType.</p>
void	<p><a href="#"><u>MAPPING_InitAddressMap</u></a>(AddressMapType* map)</p>
	<p>Initializes the AddressMapType structure.</p>
void	<p><a href="#"><u>MAPPING_BuildAddressMap</u></a>(const NodeInput* nodeInput, NodeAddress** nodeIdArrayPtr, AddressMapType* map)</p>
	<p>Builds the address map</p>
NodeAddress	<p><a href="#"><u>MAPPING_GetInterfaceAddressForSubnet</u></a>(Node* node, NodeAddress nodeId, NodeAddress subnetAddress, int numHostBits)</p>

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

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

	Get Network version IPv4/IPv6.
NetworkType	<a href="#">MAPPING_GetNetworkType</a> (const char* addrString)
	Identify network type from addrString.
void	<a href="#">MAPPING_GetIpv6InterfaceInfoForInterface</a> (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	<a href="#">MAPPING_GetIpv6GlobalAddress</a> (Node *node node, NodeId nodeId nodeId, unsigned int tla, unsigned int nla, unsigned int sla, in6_addr * addr6)
	Get IPV6 global address.
BOOL	<a href="#">MAPPING_GetIpv6GlobalAddressForInterface</a> (Node * node, NodeId nodeId, int interfaceIndex, in6_addr * addr6, BOOL isDeprecated)
	Get IPV6 global address for a node's nth interface.
void	<a href="#">MAPPING_CreateIpv6GlobalUnicastAddr</a> (unsigned int tla, unsigned int nla, unsigned int sla, int addressCounter, in6_addr* globalAddr)
	Create IPv6 Global Unicast Address from tla nla sla
void	<a href="#">MAPPING_CreateIpv6GlobalUnicastAddr</a> (AddressMapType * map, in6_addr IPv6subnetAddress, uunsigned int IPv6subnetPrefixLen, int addressCounter, in6_addr* globalAddr)
	Create IPv6 Global Unicast Address.
void	<a href="#">MAPPING_CreateIpv6LinkLocalAddr</a> (Node* node, Int32 interfaceId, in6_addr* globalAddr, in6_addr* linkLocalAddr, unsigned int subnetPrefixLen)
	Create IPv6 link local Address.
void	<a href="#">MAPPING_CreateIpv6SiteLocalAddr</a> (in6_addr* globalAddr, in6_addr* siteLocalAddr, unsigned short siteCounter, unsigned int subnetPrefixLen)
	Create IPv6 site local Address.
void	<a href="#">MAPPING_CreateIpv6MulticastAddr</a> (in6_addr* globalAddr, in6_addr* multicastAddr)
	Create ipv6 multicast address.
void	<a href="#">MAPPING_CreateIpv6SubnetAddr</a> (unsigned int tla, unsigned int nla, unsigned int sla, unsigned int* IPv6subnetPrefixLen, in6_addr* IPv6subnetAddress)

	<p>create subnet addr for IPV6 address.</p>
NodeId	<p>Get node id from Global Address.</p> <pre><a href="#">MAPPING_GetNodeIdFromGlobalAddr</a>(Node * node, in6_addr* globalAddr)</pre>
NodeId	<p>Get node id from Link layer Address.</p> <pre><a href="#">MAPPING_GetNodeIdFromLinkLayerAddr</a>(Node * node, NodeAddress linkLayerAddr)</pre>
NodeAddress	<p>Create IPv6 link layer Address.</p> <pre><a href="#">MAPPING_CreateIpv6LinkLayerAddr</a>(unsigned int nodeId, int interfaceId)</pre>
BOOL	<p>checks whether the ipv6 address is of this node.</p>
BOOL	<p>checks whether the node is in given range of : Addresses.</p> <pre><a href="#">MAPPING_IsNodeInThisIpRange</a>(Node* node, NodeId nodeId, NodeAddress startRange, NodeAddress endRange)</pre>
BOOL	<p>checks whether the ipv4 address is of this node.</p>
BOOL	<p>Get interface address for subnet using ipv6 addr.</p> <pre><a href="#">MAPPING_GetInterfaceAddressForSubnet</a>(Node* node, NodeId nodeId, in6_addr* ipv6SubnetAddr, unsigned int prefixLenth, in6_addr* ipv6InterfaceAddr, int* interfaceIndex)</pre>
BOOL	<p>Get interface address for subnet using ipv6 addr.</p> <pre><a href="#">MAPPING_GetInterfaceAddressForSubnet</a>(const AddressMapType* map, NodeId nodeId, in6_addr* ipv6SubnetAddr, unsigned int prefixLenth, in6_addr* ipv6InterfaceAddr, int* interfaceIndex)</pre>
BOOL	<p>Get interface address for subnet using tla nla sla.</p> <pre><a href="#">MAPPING_GetInterfaceAddressForSubnet</a>(Node* node node, NodeId nodeId nodeId, unsigned int tla, unsigned int nla, unsigned int sla, in6_addr* ipv6Addr, int* interfaceIndex)</pre>
BOOL	<p>Get interface address for subnet using tla nla sla.</p> <pre><a href="#">MAPPING_GetInterfaceAddressForSubnet</a>(const AddressMapType* map, NodeId nodeId, unsigned int tla, unsigned int nla, unsigned int sla, in6_addr* ipv6Addr, int* interfaceIndex)</pre>

int	<a href="#"><b>MAPPING_GetInterfaceFromLinkLayerAddress</b></a> (Node* node, const NodeAddress linkLayerAddr)  Get interface from link layer address.
int	<a href="#"><b>MAPPING_GetInterfaceIndexFromInterfaceAddress</b></a> (Node* node, Address interfaceAddress)  Get interface index from interface address.
BOOL	<a href="#"><b>MAPPING_GetIpv6GlobalAddress</b></a> (Node* node, NodeId nodeId, in6_addr subnetAddr, UInt32 prefixLen, in6_addr* addr6)  Get ipv6 global address
Address	<a href="#"><b>MAPPING_GetDefaultInterfaceAddressInfoFromNodeId</b></a> (Node *node node, NodeAddress nodeId nodeId, NetworkType networktype networktype)  Get default interface address based on network type
void	<a href="#"><b>Mapping_AutoCreateIPv6SubnetAddress</b></a> (NodeAddress ipAddress, subnetString)  Create IPv6 Testing Address Prefix (RFC 2471)from : ipv4 address.
NodeAddress	<a href="#"><b>MAPPING_GetSubnetAddressFromInterfaceAddress</b></a> (Node *node node, NodeAddress interfaceAddress)  Get subnet address from interface address.
BOOL	<a href="#"><b>MAPPING_GetSubnetAddressFromInterfaceAddress</b></a> (Node * node, in6_addr* ipv6InterfaceAddr, in6_addr* ipv6SubnetAddr)  Get ipv6 network Prefix from interface address.
BOOL	<a href="#"><b>MAPPING_GetPrefixLengthForInterfaceAddress</b></a> (Node* node, in6_addr* ipv6InterfaceAddr, unsigned int prefixLenth)  Get prefix length for interface address.
NetworkProtocolType	<a href="#"><b>MAPPING_GetNetworkProtocolTypeForNode</b></a> (NodeAddress nodeId, const NodeInput * nodeInput)  Get Network Protocol Type for the node.
NetworkType	<a href="#"><b>MAPPING_GetNetworkTypeFromInterface</b></a> (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 nodelds using a mod NODE_HASH_SIZE hash.
Structure	NetworkProperty  Describes the property of a network.
Structure	AddressMappingType  Describes the type of address mapping.
Structure	AddressReverseMappingType  Describes the type of reverse address mapping.
Structure	SubnetListType  Used to determine what the next address counter should be for each subnet address. This is needed to allow different SUBNET/LINK statements to declare the same subnet address.
Structure	AddressMapType  Describes the detailed information of Node ID <--> IP address mappings.
Structure	nodeIdToNodePtr  Describes the nodeld and corresponding nodePtr.

**Function / Macro Detail**

Function / Macro	Format
<b>MADDR6_SCOPE(a)</b>	Multicast Address Scope.
<b>IS_MULTIADDR6(a)</b>	Checks whether an address is multicast address.
<b>CLR_ADDR6(a)</b>	Set an address with 0 values.
<b>IS_CLR_ADDR6(a)</b>	Does an address have the value of 0 (Cleared).
<b>COPY_ADDR6(from, to)</b>	Copies from-ipv6 address to to-ipv6 address.
<b>SAME_ADDR6(a, b)</b>	Checks if a and b address is same address.
<b>IS_ANYADDR6(a)</b>	Checks whether the address is any address or not.
<b>IS_LOOPADDR6(a)</b>	Checks whether it is loopback address.
<b>CMP_ADDR6(a, b)</b>	Compaires two addresses.
<b>IS_IPV4ADDR6(a)</b>	Checks whether it is ipv4 address.
<b>IS_LOCALADDR6(a)</b>	Checks whether it is local address.
<b>IS_LINKLADDR6(a)</b>	Checks whether it is link local address.
<b>IS_SITELADDR6(a)</b>	Checks whether it is site local address.
<b>SAME_ADDR4(a, b)</b>	Checks whether IPv4 addresses match.
<b>IS_ANYADDR4(a)</b>	Checks whether IPv4 address is ANY_DEST.
<b>Address_IsSameAddress</b>	<b>BOOL Address_IsSameAddress (Address* addr1 addr1, Address* addr2 addr2)</b> Parameters: <ul style="list-style-type: none"> <li>• addr1 - Pointer to 1st address</li> </ul>
Check whether both addresses(i.e. addr1 and addr2) are	

same.	<ul style="list-style-type: none"> <li>• <code>addr2</code> - Pointer to 2nd address</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>BOOL</code> - None</li> </ul>
<b>Address_IsAnyAddress</b>  Check whether addr is any address of the same type	<p><code>BOOL Address_IsAnyAddress (Address* addr addr)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>addr</code> - Pointer to address</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>BOOL</code> - None</li> </ul>
<b>Address_IsMulticastAddress</b>  Check whether addr is a multicast address	<p><code>BOOL Address_IsMulticastAddress (Address* addr addr)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>addr</code> - Pointer to address</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>BOOL</code> - None</li> </ul>
<b>Address_IsSubnetBroadcastAddress</b>  Check whether addr is a subnet broadcast address	<p><code>BOOL Address_IsSubnetBroadcastAddress (Node* node node, Address* addr addr)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - pointer to node</li> <li>• <code>addr</code> - Pointer to address</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>BOOL</code> - None</li> </ul>
<b>Address_SetToAnyAddress</b>  Set addr to any address of the same type as refAddr.	<p><code>void Address_SetToAnyAddress (Address* addr addr, Address* refAddr refAddr)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>addr</code> - Pointer to address</li> <li>• <code>refAddr</code> - Pointer to refAddr</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>Address_AddressCopy</b>	<p><code>void Address_AddressCopy (Address* dstAddress, Address* srcAddress)</code></p> <p>Parameters:</p>

<p>Copy srcAddress to dstAddress</p>	<ul style="list-style-type: none"> <li>• <code>dstAddress</code> - Destination address</li> <li>• <code>srcAddress</code> - Source address</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - NULL</li> </ul>
<p><b>Ipv6CompareAddr6</b></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><b>int Ipv6CompareAddr6</b> (<code>in6_addr a, in6_addr b</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>a</code> - ipv6 address.</li> <li>• <code>b</code> - ipv6 address.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>int</code> - None</li> </ul>
<p><b>Ipv6IsAddressInNetwork</b></p> <p>Checks whether the address is in the same network. : if in the same network then returns TRUE, otherwise FALSE.</p>	<p><b>BOOL Ipv6IsAddressInNetwork</b> (<code>const in6_addr* globalAddr, unsigned int tla, unsigned int vla, unsigned int sla</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>globalAddr</code> - Pointer to ipv6 address.</li> <li>• <code>tla</code> - Top level ipv6 address.</li> <li>• <code>vla</code> - Next level ipv6 address.</li> <li>• <code>sla</code> - Site local ipv6 address.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>BOOL</code> - None</li> </ul>
<p><b>Ipv6IsAddressInNetwork</b></p> <p>Checks whether the address is in the same network. : if in the same network then returns TRUE, otherwise FALSE.</p>	<p><b>BOOL Ipv6IsAddressInNetwork</b> (<code>const in6_addr* globalAddr, const in6_addr* ipv6SubnetAddr, unsigned int prefixLenth</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>globalAddr</code> - Pointer to ipv6 address.</li> <li>• <code>ipv6SubnetAddr</code> - Pointer to ipv6 subnet address.</li> <li>• <code>prefixLenth</code> - prefix length of the address.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>BOOL</code> - TRUE if the address is in the same network, FALSE otherwise</li> </ul>
<p><b>Ipv6CheckNetworkParams</b></p>	<p><b>BOOL Ipv6CheckNetworkParams</b> (<code>unsigned int tla tla, unsigned int nla nla, unsigned int sla sla</code>)</p> <p>Parameters:</p>

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

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

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

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

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

	<ul style="list-style-type: none"> <li>• <code>interfaceAddress</code> - Interface address</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>int</code> - The interface index.</li> </ul>
<b>MAPPING_GetNodeInfoFromAtmNetInfo</b>	<p>Address <b>MAPPING_GetNodeInfoFromAtmNetInfo</b> (<code>unsigned int* index, unsigned int* genIndex</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>index</code> - return atm related interface index of a</li> <li>• <code>genIndex</code> - return generic interface index of a node.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>Address</code> - Return valid ATM Address related to Network information if genIndex is not equal to -1.</li> </ul>
<b>MAPPING_GetInterfaceIdForDestAddress</b>	<p><code>unsigned int MAPPING_GetInterfaceIdForDestAddress (Node* node, NodeId nodeId, NodeAddress destAddr)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - A pointer to node being initialized.</li> <li>• <code>nodeId</code> - Node ID</li> <li>• <code>destAddr</code> - Destination address.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>unsigned int</code> - None</li> </ul>
<b>MAPPING_GetSubnetMaskForDestAddress</b>	<p><code>NodeAddress MAPPING_GetSubnetMaskForDestAddress (Node* node, NodeId nodeId, NodeAddress destAddr)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - A pointer to node being initialized.</li> <li>• <code>nodeId</code> - Node ID</li> <li>• <code>destAddr</code> - Destination address.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>NodeAddress</code> - None</li> </ul>
<b>MAPPING_GetInterfaceAddrForNodeIdAndIntfId</b>	<p><code>NodeAddress MAPPING_GetInterfaceAddrForNodeIdAndIntfId (Node* node, NodeId nodeId, int intfId)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - The pointer to the node.</li> <li>• <code>nodeId</code> - Node ID</li> </ul>

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

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

	<p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>BOOL</code> - None</li> </ul>
<b>MAPPING_GetIpv6GlobalAddressForInterface</b>	<p><code>BOOL MAPPING_GetIpv6GlobalAddressForInterface (Node * node, NodeId nodeId, int interfaceIndex, in6_addr * addr6, BOOL isDeprecated)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - The node</li> <li>• <code>nodeId</code> - The node's id</li> <li>• <code>interfaceIndex</code> - The interface index.</li> <li>• <code>addr6</code> - The global IPv6 address.</li> <li>• <code>isDeprecated</code> - Return deprecated address (if valid)</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>BOOL</code> - None</li> </ul>
<b>MAPPING_CreateIpv6GlobalUnicastAddr</b>	<p><code>void MAPPING_CreateIpv6GlobalUnicastAddr (unsigned int tla, unsigned int nla, unsigned int sla, int addressCounter, in6_addr* globalAddr)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>tla</code> - Top level aggregation</li> <li>• <code>nla</code> - Next level aggregation</li> <li>• <code>sla</code> - Site level aggregation</li> <li>• <code>addressCounter</code> - The address counter.</li> <li>• <code>globalAddr</code> - The global IPv6 address.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>MAPPING_CreateIpv6GlobalUnicastAddr</b>	<p><code>void MAPPING_CreateIpv6GlobalUnicastAddr (AddressMapType * map, in6_addr IPv6subnetAddress, unsigned int IPv6subnetPrefixLen, int addressCounter, in6_addr* globalAddr)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>map</code> - The address map.</li> <li>• <code>IPv6subnetAddress</code> - The subnet address.</li> <li>• <code>IPv6subnetPrefixLen</code> - The prefix length.</li> <li>• <code>addressCounter</code> - The address counter.</li> </ul>

	<ul style="list-style-type: none"> <li>• <code>globalAddr</code> - The global IPv6 address.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>MAPPING_CreateIpv6LinkLocalAddr</b>	<p>Create IPv6 link local Address.</p> <p><code>void MAPPING_CreateIpv6LinkLocalAddr (Node* node, Int32 interfaceId, in6_addr* globalAddr, in6_addr* linkLocalAddr, unsigned int subnetPrefixLen)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - pointer to node structure</li> <li>• <code>interfaceId</code> - interface Id</li> <li>• <code>globalAddr</code> - The global IPv6 address.</li> <li>• <code>linkLocalAddr</code> - The subnet IPv6 address.</li> <li>• <code>subnetPrefixLen</code> - The subnet prefix length.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>MAPPING_CreateIpv6SiteLocalAddr</b>	<p>Create IPv6 site local Address.</p> <p><code>void MAPPING_CreateIpv6SiteLocalAddr (in6_addr* globalAddr, in6_addr* siteLocalAddr, unsigned short siteCounter, unsigned int subnetPrefixLen)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>globalAddr</code> - The global IPv6 address.</li> <li>• <code>siteLocalAddr</code> - The subnet IPv6 address.</li> <li>• <code>siteCounter</code> - The counter to use.</li> <li>• <code>subnetPrefixLen</code> - The subnet prefix length.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>MAPPING_CreateIpv6MulticastAddr</b>	<p>Create ipv6 multicast address.</p> <p><code>void MAPPING_CreateIpv6MulticastAddr (in6_addr* globalAddr, in6_addr* multicastAddr)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>globalAddr</code> - The global IPv6 address.</li> <li>• <code>multicastAddr</code> - The multicast IPv6 address.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>

<b>MAPPING_CreateIpv6SubnetAddr</b>  create subnet addr for IPV6 address.	<p>void <b>MAPPING_CreateIpv6SubnetAddr</b> (unsigned int tla, unsigned int nla, unsigned int sla, unsigned int* IPv6subnetPrefixLen, in6_addr* IPv6subnetAddress)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• tla - Top level aggregation.</li> <li>• nla - Next level aggregation.</li> <li>• sla - Site level aggregation.</li> <li>• IPv6subnetPrefixLen - The IPv6 prefix length.</li> <li>• IPv6subnetAddress - The IPv6 subnet address.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
Get node id from Global Address.	<p>NodeId <b>MAPPING_GetNodeIdFromGlobalAddr</b> (Node * node, in6_addr* globalAddr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - The node.</li> <li>• globalAddr - The global IPv6 address.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• NodeId - None</li> </ul>
Get node id from Link layer Address.	<p>NodeId <b>MAPPING_GetNodeIdFromLinkLayerAddr</b> (Node * node, NodeAddress linkLayerAddr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - The node.</li> <li>• linkLayerAddr - The link layer address.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• NodeId - None</li> </ul>
Create IPv6 link layer Address.	<p>NodeAddress <b>MAPPING_CreateIpv6LinkLayerAddr</b> (unsigned int nodeId, int interfaceId)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• nodeId - The node's id.</li> <li>• interfaceId - The interface id.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• NodeAddress - None</li> </ul>

<b>MAPPING_IsIpv6AddressOfThisNode</b>	<p>checks whether the ipv6 address is of this node.</p> <p><b>BOOL MAPPING_IsIpv6AddressOfThisNode</b> (Node* node, const NodeAddress nodeId, in6_addr* globalAddr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - The node id.</li> <li>• nodeId - The node's address.</li> <li>• globalAddr - The global IPv6 address.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• BOOL - None</li> </ul>
<b>MAPPING_IsNodeInThisIpRange</b>	<p>checks whether the node is in given range of : Addresses.</p> <p><b>BOOL MAPPING_IsNodeInThisIpRange</b> (Node* node, NodeId nodeId, NodeAddress startRange, NodeAddress endRange)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - The node.</li> <li>• nodeId - The node id.</li> <li>• startRange - The starting address.</li> <li>• endRange - The end address.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• BOOL - None</li> </ul>
<b>MAPPING_IsIpAddressOfThisNode</b>	<p>checks whether the ipv4 address is of this node.</p> <p><b>BOOL MAPPING_IsIpAddressOfThisNode</b> (Node* node, const NodeAddress nodeId, NodeAddress addr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - The node.</li> <li>• nodeId - The node id.</li> <li>• addr - The address.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• BOOL - None</li> </ul>
<b>MAPPING_GetInterfaceAddressForSubnet</b>	<p>Get interface address for subnet using ipv6 addr.</p> <p><b>BOOL MAPPING_GetInterfaceAddressForSubnet</b> (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"> <li>• node - The node.</li> <li>• nodeId - The node id.</li> <li>• ipv6SubnetAddr - The subnet address.</li> </ul>

	<ul style="list-style-type: none"> <li>• <code>prefixLenth</code> - The subnet prefix length.</li> <li>• <code>ipv6InterfaceAddr</code> - The ipv6 interface address.</li> <li>• <code>interfaceIndex</code> - The interface index.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>BOOL</code> - None</li> </ul>
<b>MAPPING_GetInterfaceAddressForSubnet</b>	<p>Get interface address for subnet using ipv6 addr.</p> <p><b>BOOL MAPPING_GetInterfaceAddressForSubnet</b> (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"> <li>• <code>map</code> - The address map.</li> <li>• <code>nodeId</code> - The node id.</li> <li>• <code>ipv6SubnetAddr</code> - The subnet address.</li> <li>• <code>prefixLenth</code> - The subnet prefix length.</li> <li>• <code>ipv6InterfaceAddr</code> - The ipv6 interface address.</li> <li>• <code>interfaceIndex</code> - The interface index.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>BOOL</code> - None</li> </ul>
<b>MAPPING_GetInterfaceAddressForSubnet</b>	<p>Get interface address for subnet using tla nla sla.</p> <p><b>BOOL MAPPING_GetInterfaceAddressForSubnet</b> (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"> <li>• <code>node</code> - The node.</li> <li>• <code>nodeId</code> - The node id.</li> <li>• <code>tla</code> - Top level aggregation.</li> <li>• <code>nla</code> - Next level aggregation.</li> <li>• <code>sla</code> - Site level aggregation.</li> <li>• <code>ipv6Addr</code> - The ipv6 interface address.</li> <li>• <code>interfaceIndex</code> - The interface index.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>BOOL</code> - None</li> </ul>

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

	<ul style="list-style-type: none"> <li><code>subnetAddr</code> - The subnet address.</li> <li><code>prefixLen</code> - The subnet prefix length.</li> <li><code>addr6</code> - The IPv6 address.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li><code>BOOL</code> - None</li> </ul>
<b>MAPPING_GetDefaultInterfaceAddressInfoFromNodeId</b>	<p>Address <b>MAPPING_GetDefaultInterfaceAddressInfoFromNodeId</b> (<code>Node *node node, NodeAddress nodeId nodeId, NetworkType networktype networktype</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li><code>node</code> - The node.</li> <li><code>nodeId</code> - The node id.</li> <li><code>networktype</code> - The network type.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li><code>Address</code> - None</li> </ul>
<b>Mapping_AutoCreateIPv6SubnetAddress</b>	<p>Create IPv6 Testing Address Prefix (RFC 2471)from : ipv4 address.</p> <p>void <b>Mapping_AutoCreateIPv6SubnetAddress</b> (<code>NodeAddress ipAddress, subnetString</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li><code>ipAddress</code> - The IPv4 address.</li> <li><code>subnetString</code> - <code>char*</code></li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li><code>void</code> - NONE</li> </ul>
<b>MAPPING_GetSubnetAddressFromInterfaceAddress</b>	<p>Get subnet address from interface address.</p> <p><code>NodeAddress MAPPING_GetSubnetAddressFromInterfaceAddress (Node *node node, NodeAddress interfaceAddress)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li><code>node</code> - The node address.</li> <li><code>interfaceAddress</code> - The interface address.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li><code>NodeAddress</code> - subnet address</li> </ul>
<b>MAPPING_GetSubnetAddressFromInterfaceAddress</b>	<p><code>BOOL MAPPING_GetSubnetAddressFromInterfaceAddress (Node * node, in6_addr* ipv6InterfaceAddr, in6_addr* ipv6SubnetAddr)</code></p> <p>Parameters:</p>

<p>Get ipv6 network Prefix from interface address.</p>	<ul style="list-style-type: none"> <li>• <code>node</code> - The node.</li> <li>• <code>ipv6InterfaceAddr</code> - The IPv6 interface address.</li> <li>• <code>ipv6SubnetAddr</code> - The subnet address pointer .</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>BOOL</code> - None</li> </ul>
<p><b>MAPPING_GetPrefixLengthForInterfaceAddress</b></p> <p>Get prefix length for interface address.</p>	<p><code>BOOL MAPPING_GetPrefixLengthForInterfaceAddress (Node* node, in6_addr* ipv6InterfaceAddr, unsigned int prefixLenth)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - The node.</li> <li>• <code>ipv6InterfaceAddr</code> - The IPV6 interface address.</li> <li>• <code>prefixLenth</code> - The interface prefix length.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>BOOL</code> - None</li> </ul>
<p><b>MAPPING_GetNetworkProtocolTypeForNode</b></p> <p>Get Network Protocol Type for the node.</p>	<p><code>NetworkProtocolType MAPPING_GetNetworkProtocolTypeForNode (NodeAddress nodeId, const NodeInput* nodeInput)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>nodeId</code> - The node id.</li> <li>• <code>nodeInput</code> - The node input file</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>NetworkProtocolType</code> - None</li> </ul>
<p><b>MAPPING_GetNetworkTypeFromInterface</b></p> <p>This function determines the network type of a particular interface of a node</p>	<p><code>NetworkType MAPPING_GetNetworkTypeFromInterface (Node* node, Int32 interfaceIndex)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to a node</li> <li>• <code>interfaceIndex</code> - interface index</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>NetworkType</code> - network type of an interface of a node</li> </ul>



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

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

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



# EXata 5.1 API Reference

## MEMORY

This file describes the memory management data structures and functions.

### Constant / Data Structure Summary

Type	Name
STRUCT	<a href="#">MemoryUsageData</a>  Defines the parameters collected by the memory system. Restricted to kernel use.

### Function / Macro Summary

Return Type	Summary
void	<a href="#">MEM_CreateThreadData()</a>  Creates partition-specific space for collecting memory usage statistics. This is used in threaded versions of QualNet, but not in distributed versions, currently.
void	<a href="#">MEM_InitializeThreadData(MemoryUsageData* data)</a>  Sets the partition-specific memory data for this partition.
void	<a href="#">MEM_PrintThreadData()</a>  Prints the partition-specific memory data.
void	<a href="#">MEM_ReportPartitionUsage(int partitionId, UInt32 totalAllocatedMemory, UInt32 totalFreedMemory, UInt32 totalPeakUsage)</a>  Prints out the total memory used by this partition.
void	<a href="#">MEM_ReportTotalUsage(UInt32 totalAllocatedMemory, UInt32 totalFreedMemory, UInt32 totalPeakUsage)</a>  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	MemoryUsageData
	Defines the parameters collected by the memory system. Restricted to kernel use.

## Function / Macro Detail

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

	<ul style="list-style-type: none"> <li>• <code>totalFreedMemory</code> - sum of all <code>MEM_free</code> calls</li> <li>• <code>totalPeakUsage</code> - peak usage of allocated memory</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>MEM_ReportTotalUsage</b>	<p><code>void MEM_ReportTotalUsage (UInt32 totalAllocatedMemory, UInt32 totalFreedMemory, UInt32 totalPeakUsage)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>totalAllocatedMemory</code> - sum of all <code>MEM_malloc</code> calls</li> <li>• <code>totalFreedMemory</code> - sum of all <code>MEM_free</code> calls</li> <li>• <code>totalPeakUsage</code> - peak usage of allocated memory</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>



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

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

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



# EXata 5.1 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	<a href="#">MSG_MAX_HDR_SIZE</a>  Maximum Header Size
CONSTANT	<a href="#">SMALL_INFO_SPACE_SIZE</a>  Size of small Info field. Should be larger than all commonly used info field data structures, especially PropTxInfo and PropRxInfo.
CONSTANT	<a href="#">MSG_PAYLOAD_LIST_MAX</a>  Maximum message payload list
CONSTANT	<a href="#">MAX_CACHED_PAYLOAD_SIZE</a>  Maximum cached payload size
CONSTANT	<a href="#">MSG_INFO_LIST_MAX</a>  Maximum message info list
CONSTANT	<a href="#">MAX_INFO_FIELDS</a>  Maximum number of info fields
CONSTANT	<a href="#">MAX_HEADERS</a>  Maximum number of headers
ENUMERATION	<a href="#">MessageInfoType</a>

	Type of information in the info field. One message can only have up to one info field with a specific info type.
STRUCT	<a href="#">MessageInfoHeader</a>  This is a structure which contains information about a info field.
STRUCT	<a href="#">Message</a>  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.

**Function / Macro Summary**

Return Type	Summary
void	<a href="#">MESSAGE_PrintMessage</a> (Node* node, Message* msg)
void	<a href="#">MESSAGE_Send</a> (Node* node, Message* msg, clocktype delay, bool isMT)  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.
void	<a href="#">MESSAGE_SendMT</a> (Node* node, Message* msg, clocktype delay)  Function call used to send a message from independent threads running within QualNet, for example those associated with external interfaces.
void	<a href="#">MESSAGE_RemoteSend</a> (Node* node, NodeId destNodeId, Message* msg, clocktype delay)  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.
void	<a href="#">MESSAGE_RouteReceivedRemoteEvent</a> (Node* node, Message* msg)  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.
void	<a href="#">MESSAGE_CancelSelfMsg</a> (Node* node, Message* msgToCancelPtr)  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.
Message*	<a href="#">MESSAGE_Alloc</a> (Node* node, 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*	<a href="#"><b>MESSAGE_Alloc</b></a> (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*	<a href="#"><b>MESSAGE_AllocMT</b></a> (PartitionData* partition, int layerType, int protocol, int eventType)
	Mutli-thread safe version of MESSAGE_Alloc for use by worker threads.
char*	<a href="#"><b>MESSAGE_InfoFieldAlloc</b></a> (Node* node, int infoSize)
	Allocate space for one "info" field
char*	<a href="#"><b>MESSAGE_InfoFieldAlloc</b></a> (PartitionData* partition, int infoSize)
	Allocate space for one "info" field
char*	<a href="#"><b>MESSAGE_InfoFieldAllocMT</b></a> (PartitionData* partition, int infoSize)
	Multi-thread safe version of MESSAGE_InfoFieldAlloc
void	<a href="#"><b>MESSAGE_InfoFieldFree</b></a> (Node* node, MessageInfoHeader* hdrPtr)
	Free space for one "info" field
char*	<a href="#"><b>MESSAGE_AddInfo</b></a> (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, <a href="#"><b>MESSAGE_ReturnInfo</b></a> function can be used to get a pointer to the allocated space for the info field in the message structure.
char*	<a href="#"><b>MESSAGE_AddInfo</b></a> (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, <a href="#"><b>MESSAGE_ReturnInfo</b></a> function can be used to get a pointer to the allocated space for the info field in the message structure.
void	<a href="#"><b>MESSAGE_RemoveInfo</b></a> (Node* node, Message* msg, unsigned short infoType)

	Remove one "info" field with given info type from the info array of the message.
char *	<p><a href="#">MESSAGE_InfoAlloc</a>(Node* node, 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>
char *	<p><a href="#">MESSAGE_InfoAlloc</a>(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><a href="#">MESSAGE_ReturnInfoSize</a>(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><a href="#">MESSAGE_ReturnInfoSize</a>(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><a href="#">MESSAGE_ReturnInfo</a>(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><a href="#">MESSAGE_CopyInfo</a>(Node* node, Message* dsgMsg, Message* srcMsg)</p> <p>Copy the "info" fields of the source message to the destination message.</p>
void	<p><a href="#">MESSAGE_CopyInfo</a>(Node* node, Message* dsgMsg, MessageInfoHeader* srcInfo)</p> <p>Copy the "info" fields of the source info header to the destination message.</p>
void	<p><a href="#">MESSAGE_FragmentPacket</a>(Node* node, Message* msg, int fragUnit, Message*** fragList, int* numFrags, 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><a href="#">MESSAGE_ReassemblePacket</a>(Node* node, Message** fragList, int numFrags, TraceProtocolType protocolType)</p> <p>Reassemble multiple fragments into one packet Note: All the fragments will be freed in this function.</p>
Message*	<p><a href="#">MESSAGE_PackMessage</a>(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*	<a href="#"><b>MESSAGE_UnpackMessage</b></a> (Node* node, Message* msg, bool copyInfo, bool freeOld)
	Unpack a super message to the original list of messages The list of messages were stored as payload of this super message.
void	<a href="#"><b>MESSAGE_PacketAlloc</b></a> (Node* node, Message* msg, int packetSize, TraceProtocolType originalProtocol)
	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.
void	<a href="#"><b>MESSAGE_PacketAlloc</b></a> (PartitionData* partition, Message* msg, int packetSize, TraceProtocolType originalProtocol, bool isMT)
	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.
void	<a href="#"><b>MESSAGE_AddHeader</b></a> (Node* node, Message* msg, int hdrSize, TraceProtocolType traceProtocol)
	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.
void	<a href="#"><b>MESSAGE_RemoveHeader</b></a> (Node* node, Message* msg, int hdrSize, TraceProtocolType traceProtocol)
	This function is called to remove a header from the packet. The "packetSize" variable in the message will be decreased by "hdrSize".
char*	<a href="#"><b>MESSAGE_ReturnHeader</b></a> (Message* msg, int header)
	This is kind of a hack so that MAC protocols (dot11) that need to peek at a packet that still has the PHY header can return the contents after the first (N) headers without first removing those headers.
void	<a href="#"><b>MESSAGE_ExpandPacket</b></a> (Node* node, Message* msg, int size)
	Expand packet by a specified size
void	<a href="#"><b>MESSAGE_ShrinkPacket</b></a> (Node* node, Message* msg, int size)
	This function is called to shrink packet by a specified size.
void	<a href="#"><b>MESSAGE_Free</b></a> (PartitionData* partition, Message* msg)
	When the message is no longer needed it can be freed. Firstly the "payload" and "info" fields of the message are freed. Then the message itself is freed. It is important to remember to free the message. Otherwise there will nasty memory leaks in the program.

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

int	<a href="#"><b>MESSAGE_ReturnFragSeqNum</b></a> (Message* msg, int fragmentNumber)
	Returns the sequence number of a particular fragments in the TCP packet.
int	<a href="#"><b>MESSAGE_ReturnFragSize</b></a> (Message* msg, int fragmentNumber)
	Returns the size of a particular fragment in the TCP packet.
int	<a href="#"><b>MESSAGE_ReturnFragNumInfos</b></a> (Message* msg, int fragmentNumber)
	Returns the number of info fields associated with a particular fragment in the TCP packet.
void	<a href="#"><b>MESSAGE_AppendInfo</b></a> (PartitionData* partitionData, Message* msg, int infosize, short infoType)
	Appends the "info" fields of the source message to the destination message.
void	<a href="#"><b>MESSAGE_AppendInfo</b></a> (Node* node, Message* msg, int infosize, short infoType)
	Appends the "info" fields of the source message to the destination message.
void	<a href="#"><b>MESSAGE_AppendInfo</b></a> (Node* node, Message* dsgMsg, MessageInfoHeader* srcInfo)
	Appends the "info" fields of the source message to the destination message.
void	<a href="#"><b>MESSAGE_AppendInfo</b></a> (Node* node, Message* dsgMsg, Message* srcMsg)
	Appends the "info" fields of the source message to the destination message.
size_t	<a href="#"><b>MESSAGE_SizeOf</b></a> ()
	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	<a href="#"><b>MESSAGE_FragmentPacket</b></a> (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*	<a href="#"><b>MESSAGE_ReassemblePacket</b></a> (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	<a href="#"><b>MESSAGE_SendAsEarlyAsPossible</b></a> (Node* node, Message* msg)

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

## Constant / Data Structure Detail

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

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

**Function / Macro Detail**

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

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

	<p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>Message*</code> - Pointer to allocated message structure</li> </ul>
<b>MESSAGE_Alloc</b>	<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><code>Message* MESSAGE_Alloc (PartitionData* partition, int layerType, int protocol, int eventType)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>partition</code> - partition that is allocating message</li> <li>• <code>layerType</code> - Layer type to be set for this message</li> <li>• <code>protocol</code> - Protocol to be set for this message</li> <li>• <code>eventType</code> - event type to be set for this message</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>Message*</code> - Pointer to allocated message structure</li> </ul>
<b>MESSAGE_AllocMT</b>	<p>Mutli-thread safe version of MESSAGE_Alloc for use by worker threads.</p> <p><code>Message* MESSAGE_AllocMT (PartitionData* partition, int layerType, int protocol, int eventType)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>partition</code> - partition that is allocating message</li> <li>• <code>layerType</code> - Layer type to be set for this message</li> <li>• <code>protocol</code> - Protocol to be set for this message</li> <li>• <code>eventType</code> - event type to be set for this message</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>Message*</code> - Pointer to allocated message structure</li> </ul>
<b>MESSAGE_InfoFieldAlloc</b>	<p>Allocate space for one "info" field</p> <p><code>char* MESSAGE_InfoFieldAlloc (Node* node, int infoSize)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - node which is allocating the space.</li> <li>• <code>infoSize</code> - size of the space to be allocated</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>char*</code> - pointer to the allocated space.</li> </ul>
<b>MESSAGE_InfoFieldAlloc</b>	<p>Allocate space for one "info" field</p> <p><code>char* MESSAGE_InfoFieldAlloc (PartitionData* partition, int infoSize)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>partition</code> - partition which is allocating the space.</li> </ul>

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

<p>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>	<ul style="list-style-type: none"> <li>• <code>infoSize</code> - size of the "info" field to be allocated</li> <li>• <code>infoType</code> - type of the "info" field to be allocated.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>char*</code> - Pointer to the added info field</li> </ul>
<p><b>MESSAGE_RemoveInfo</b></p> <p>Remove one "info" field with given info type from the info array of the message.</p>	<p><code>void MESSAGE_RemoveInfo (Node* node, Message* msg, unsigned short infoType)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - node which is removing info field.</li> <li>• <code>msg</code> - message for which "info" field</li> <li>• <code>infoType</code> - type of the "info" field to be removed.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - NULL</li> </ul>
<p><b>MESSAGE_InfoAlloc</b></p> <p>Allocate the default "info" field for the message. This function is similar to MESSAGE_AddInfo. The difference is that it assumes the type of the info field to be allocated is INFO_TYPE_DEFAULT.</p>	<p><code>char * MESSAGE_InfoAlloc (Node* node, Message* msg, int infoSize)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - node which is allocating the info field.</li> <li>• <code>msg</code> - message for which "info" field</li> <li>• <code>infoSize</code> - size of the "info" field to be allocated</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>char *</code> - None</li> </ul>
<p><b>MESSAGE_InfoAlloc</b></p> <p>Allocate the default "info" field for the message. This function is similar to MESSAGE_AddInfo. The difference is that it assumes the type of the info field to be allocated is INFO_TYPE_DEFAULT.</p>	<p><code>char * MESSAGE_InfoAlloc (PartitionData* partition, Message* msg, int infoSize)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>partition</code> - partition which is allocating the info field.</li> <li>• <code>msg</code> - message for which "info" field</li> <li>• <code>infoSize</code> - size of the "info" field to be allocated</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>char *</code> - None</li> </ul>
<p><b>MESSAGE_ReturnInfoSize</b></p>	<p><code>int MESSAGE_ReturnInfoSize (Message* msg, unsigned short infoType, int fragmentNumber)</code></p>

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

<p>header to the destination message.</p>	<ul style="list-style-type: none"> <li>• <code>dsgMsg</code> - Destination message</li> <li>• <code>srcInfo</code> - Info Header structure</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - NULL</li> </ul>
<p><b>MESSAGE_FragmentPacket</b></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* numFrags, TraceProtocolType protocolType)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - node which is fragmenting the packet</li> <li>• <code>msg</code> - The packet to be fragmented</li> <li>• <code>fragUnit</code> - The unit size for fragmenting the packet</li> <li>• <code>fragList</code> - A list of fragments created.</li> <li>• <code>numFrags</code> - Number of fragments in the fragment list.</li> <li>• <code>protocolType</code> - Protocol type for packet tracing.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - NULL</li> </ul>
<p><b>MESSAGE_ReassemblePacket</b></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 numFrags, TraceProtocolType protocolType)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - node which is assembling the packet</li> <li>• <code>fragList</code> - A list of fragments.</li> <li>• <code>numFrags</code> - Number of fragments in the fragment list.</li> <li>• <code>protocolType</code> - Protocol type for packet tracing.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>Message*</code> - The reassembled packet.</li> </ul>
<p><b>MESSAGE_PackMessage</b></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</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"> <li>• <code>node</code> - Pointer to node.</li> <li>• <code>msgList</code> - Pointer to a list of messages</li> <li>• <code>origProtocol</code> - Protocol allocating this packet</li> </ul>

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

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

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

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

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

create a TCP packet.	<p>Returns:</p> <ul style="list-style-type: none"> <li>• int - Number of Fragments. 0 if none.</li> </ul>
<b>MESSAGE_ReturnFragSeqNum</b>  Returns the sequence number of a particular fragments in the TCP packet.	<p>int <b>MESSAGE_ReturnFragSeqNum</b> (Message* msg, int fragmentNumber)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• msg - message for which "info" field</li> <li>• fragmentNumber - fragment location in the TCP message.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• int - Sequence number of the fragment. -1 if none.</li> </ul>
<b>MESSAGE_ReturnFragSize</b>  Returns the size of a particular fragment in the TCP packet.	<p>int <b>MESSAGE_ReturnFragSize</b> (Message* msg, int fragmentNumber)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• msg - message for which "info" field</li> <li>• fragmentNumber - fragment location in the TCP message.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• int - Sequence number of the fragment. 0 if none.</li> </ul>
<b>MESSAGE_ReturnFragNumInfos</b>  Returns the number of info fields associated with a particular fragment in the TCP packet.	<p>int <b>MESSAGE_ReturnFragNumInfos</b> (Message* msg, int fragmentNumber)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• msg - message for which "info" field</li> <li>• fragmentNumber - fragment location in the TCP message.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• int - Sequence number of the fragment. 0 if none.</li> </ul>
<b>MESSAGE_AppendInfo</b>  Appends the "info" fields of the source message to the destination message.	<p>void <b>MESSAGE_AppendInfo</b> (PartitionData* partitionData, Message* msg, int infosize, short infoType)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• partitionData - Partition which is copying the info fields</li> <li>• msg - Destination message</li> <li>• infosize - size of the info field</li> <li>• infoType - type of info field.</li> </ul> <p>Returns:</p>

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

	<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><b>int MESSAGE_FragmentPacket(Node* node, Message* msg, Node* fragmentedMsg, Node* remainingMsg, int fragUnit, TraceProtocolType protocolType, bool freeOriginalMsg)</b></p> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• <b>node</b> - node which is fragmenting the packet</li> <li>• <b>msg</b> - The packet to be fragmented</li> <li>• <b>fragmentedMsg</b> - First fragment</li> <li>• <b>remainingMsg</b> - Remaining packet</li> <li>• <b>fragUnit</b> - The unit size for fragmenting the packet</li> <li>• <b>protocolType</b> - Protocol type for packet tracing.</li> <li>• <b>freeOriginalMsg</b> - If TRUE, then original msg is set to NULL</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>• <b>BOOL</b> - TRUE if any fragment is created, FALSE otherwise</li> </ul>
<b>MESSAGE_ReassemblePacket</b> <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><b>Message* MESSAGE_ReassemblePacket (Node* node, Message* fragMsg1, Message* fragMsg2, TraceProtocolType protocolType)</b></p> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• <b>node</b> - node which is assembling the packet</li> <li>• <b>fragMsg1</b> - First fragment</li> <li>• <b>fragMsg2</b> - Second fragment</li> <li>• <b>protocolType</b> - Protocol type for packet tracing.</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>• <b>Message*</b> - The reassembled packet.</li> </ul>
<b>MESSAGE_SendAsEarlyAsPossible</b> <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><b>void MESSAGE_SendAsEarlyAsPossible (Node* node, Message* msg)</b></p> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• <b>node</b> - node which is sending message</li> <li>• <b>msg</b> - message to be delivered</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>• <b>void</b> - NULL</li> </ul>



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

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

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



# EXata 5.1 API Reference

## MOBILITY

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

### Constant / Data Structure Summary

Type	Name
CONSTANT	<a href="#">DEFAULT_DISTANCE_GRANULARITY</a>  Defines the default distance granularity
CONSTANT	<a href="#">NUM_NODE_PLACEMENT_TYPES</a>  Defines the number of node placement schemes
CONSTANT	<a href="#">NUM_MOBILITY_TYPES</a>  Defines the number of mobility models
CONSTANT	<a href="#">NUM_PAST_MOBILITY_EVENTS</a>  Number of past mobility models stored
ENUMERATION	<a href="#">NodePlacementType</a>  Specifies different node placement schemes
ENUMERATION	<a href="#">MobilityType</a>  Specifies different mobility models
STRUCT	<a href="#">MobilityHeap</a>  A Heap that determines the earliest time
STRUCT	<a href="#">MobilityElement</a>

	Defines all the element of mobility model.
STRUCT	<a href="#">MobilityRemainder</a>  A structure that defines the next states of the elements of mobility model.
STRUCT	<a href="#">MobilityData</a>  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	<a href="#">MOBILITY_InsertEvent</a> (MobilityHeap* heapPtr, Node* node)  Inserts an event.
void	<a href="#">MOBILITY_DeleteEvent</a> (MobilityHeap* heapPtr, Node* node)  Deletes an event.
void	<a href="#">MOBILITY_HeapFixDownEvent</a> (MobilityHeap* heapPtr, int i)  Inserts an event and sort out the heap downwards
void	<a href="#">MOBILITY_AllocateNodePositions</a> (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	<a href="#">MOBILITY_PreInitialize</a> (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	<a href="#">MOBILITY_PostInitialize</a> (Node* node, NodeInput* nodeInput)  Initializes variables in mobilityData not initialized by MOBILITY_PreInitialize().
void	<a href="#">MOBILITY_UpdatePathProfiles</a> (MobilityHeap* pathProfileHeap, clocktype nextEventTime, clocktype* upperBoundTime)

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

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

**Constant / Data Structure Detail**

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

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

## Function / Macro Detail

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

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

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

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

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

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

	<p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>MOBILITY_PostInitializePartition</b>	<p>void <b>MOBILITY_PostInitializePartition</b> (PartitionData* partitionData)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• partitionData - Pointer to the partition data</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>MOBILITY_NodePlacementFinalize</b>	<p>void <b>MOBILITY_NodePlacementFinalize</b> (PartitionData* partitionData)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• partitionData - Pointer to the partition data</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>MOBILITY_ChangeGroundNode</b>	<p>void <b>MOBILITY_ChangeGroundNode</b> (Node* node, BOOL before, BOOL after)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Pointer to node being initialized.</li> <li>• before - Orginal value for Ground-Node variable</li> <li>• after - new value for Ground-Node variable.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>MOBILITY_ChangePositionGranularity</b>	<p>void <b>MOBILITY_ChangePositionGranularity</b> (Node* node)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Pointer to node being initialized.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>



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

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

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



# EXata 5.1 API Reference

## MUTEX

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

### Function / Macro Summary

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

### Function / Macro Detail

Function / Macro	Format
<b>QNThreadLock</b>	<p>None <b>QNThreadLock</b> (QNThreadMutex mutex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>mutex</code> - Pointer to the Thread mutex to lock for this</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• None - None</li> </ul> <p>This constructor is used to begin a critical region.</p>
<b>QNPartitionLock</b>	<p>None <b>QNPartitionLock</b> (QNPartitionMutex mutex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>mutex</code> - Pointer to the Partition mutex to lock</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• None - None</li> </ul> <p>This constructor is used to begin a critical region.</p>



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

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

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



# EXata 5.1 API Reference

## NETWORK LAYER

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

### Constant / Data Structure Summary

Type	Name
CONSTANT	<a href="#">DEFAULT_IP_QUEUE_COUNT</a>
	Default number of output queue per interface
CONSTANT	<a href="#">DEFAULT_CPU_QUEUE_SIZE</a>
	Default size of CPU queue (in byte)
CONSTANT	<a href="#">DEFAULT_NETWORK_INPUT_QUEUE_SIZE</a>
	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	<a href="#">DEFAULT_NETWORK_OUTPUT_QUEUE_SIZE</a>
	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	<a href="#">DEFAULT_ETHERNET_MTU</a>
	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	<a href="#">IP_MAXPACKET</a>
	Maximum IP packet size
CONSTANT	<a href="#">NETWORK_IP_UNLIMITED_BACKPLANE_THROUGHPUT</a>

	Maximum throughput of backplane of network. <a href="#">NetworkIpBackplaneStatusType</a>
ENUMERATION	Status of backplane (either busy or idle) <a href="#">NetworkRoutingAdminDistanceType</a>
ENUMERATION	Administrative distance of different routing protocol <a href="#">NetworkRoutingProtocolType</a>
ENUMERATION	Enlisted different network/routing protocol <a href="#">ManagementReportType</a>
ENUMERATION	Type of management report message <a href="#">ManagementResponseType</a>
STRUCT	Type of management response message <a href="#">ManagementData</a>
STRUCT	Main data structure of network layer <a href="#">ManagementReport</a>
STRUCT	data structure of management report <a href="#">ManagementResponse</a>
	data structure of management response

**Function / Macro Summary**

Return Type	Summary
void	<a href="#">NETWORK_ManagementReport</a> (Node* node, int interfaceIndex, ManagementReport* report, ManagementReportResponse* resp)  Deliver a MAC management request to the NETWORK layer
void	<a href="#">NetworkGetInterfaceInfo()</a> (Node* node, int interfaceIndex, Address* address)

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

	data structure of management report
Structure	<p>ManagementResponse</p> <p>data structure of management response</p>

**Function / Macro Detail**

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

	<ul style="list-style-type: none"> <li>• <code>ipAddrString</code> - Pointer to string ipv6 address.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>NetworkIpGetInterfaceType</b>	<p>NetworkType <b>NetworkIpGetInterfaceType</b> (<code>Node* node, int interfaceIndex</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to node.</li> <li>• <code>interfaceIndex</code> - Interface index.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>NetworkType</code> - None</li> </ul>
<b>NETWORK_ReceivePacketFromMacLayer</b>	<p>void <b>NETWORK_ReceivePacketFromMacLayer</b> (<code>Node* node, Message* message, NodeAddress lastHopAddress, int interfaceIndex</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to node</li> <li>• <code>message</code> - Message received</li> <li>• <code>lastHopAddress</code> - last hop address</li> <li>• <code>interfaceIndex</code> - incoimg interface</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>NETWORK_Reset</b>	<p>void <b>NETWORK_Reset</b> (<code>Node* node, int interfaceIndex</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to node.</li> <li>• <code>interfaceIndex</code> - Interface index.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>NETWORK_AddResetFunctionList</b>	<p>void <b>NETWORK_AddResetFunctionList</b> (<code>Node* node, int interfaceIndex</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Pointer to node.</li> <li>• <code>interfaceIndex</code> - Interface index.</li> </ul>

Returns:

- void - None



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

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

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



# EXata 5.1 API Reference

## NODE

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

### Constant / Data Structure Summary

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

### Function / Macro Summary

Return Type	Summary
clocktyme	<p><a href="#">getNodeTime()</a></p> <p>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.</p>
void	<p><a href="#">NODE_CreateNode</a>(PartitionData* partitionData, NodeId nodeId, int index)</p> <p>Function used to allocate and initialize a node.</p>
void	<p><a href="#">NODE_ProcessEvent</a>(Node* node, Message* msg)</p> <p>Function used to call the appropriate layer to execute instructions for the message</p>

void	<a href="#">NODE_PrintLocation</a> (Node* node, int coordinateSystemType)
	Prints the node's three dimensional coordinates.
TerrainData*	<a href="#">NODE_GetTerrainPtr</a> (Node* node)
	Get terrainData pointer.

**Constant / Data Structure Detail**

Enumeration	<p>NodeGlobalIndex</p> <p>This enumeration contains indexes into the nodeGlobal array used for module data.</p>
Structure	<p>Node</p> <p>This struct includes all the information for a particular node. State information for each layer can be accessed from this structure.</p>
Structure	<p>NodePositions</p> <p>Contains information about the initial positions of nodes.</p>

**Function / Macro Detail**

Function / Macro	Format
<b>getNodeTime</b>	<p>clocktyme <b>getNodeTime</b> ()</p> <p>Parameters:</p> <p>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.</p> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>clocktyme</code> - The current time</li> </ul>
<b>NODE_CreateNode</b>	<p>void <b>NODE_CreateNode</b> (PartitionData* partitionData, NodeId nodeId, int index)</p> <p>Parameters:</p> <p>Function used to allocate and initialize a</p> <ul style="list-style-type: none"> <li>• <code>partitionData</code> - the partition that owns the node</li> </ul>

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





# EXata 5.1 API Reference

## PARALLEL

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

### Constant / Data Structure Summary

Type	Name
CONSTANT	<a href="#">MAX_THREADS</a>
	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	<a href="#">SynchronizationAlgorithm</a>
	Possible algorithms to use in the parallel runtime. Synchronous is used by default.
ENUMERATION	<a href="#">BarrierType</a>
	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	<a href="#">LookaheadLocator</a>
	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	<a href="#">EotHeapElement</a>
	Basic data structure for simplifying lookahead calculation.
STRUCT	<a href="#">LookaheadCalculator</a>
	Stores a heap of EOT elements to calculate lookahead.

### Function / Macro Summary

Return Type	Summary
LookaheadHandle	<p><a href="#"><code>PARALLEL_AllocateLookaheadHandle</code></a>(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><a href="#"><code>PARALLEL_AddLookaheadHandleToLookaheadCalculator</code></a>(Node* node, LookaheadHandle lookaheadHandle, clocktype eotOfNode)</p> <p>Adds a new LookaheadHandle to the lookahead calculator.</p>
void	<p><a href="#"><code>PARALLEL_SetLookaheadHandleEOT</code></a>(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><a href="#"><code>PARALLEL_RemoveLookaheadHandleFromLookaheadCalculator</code></a>(Node* node, LookaheadHandle lookaheadHandle, clocktype* eotOfNode)</p> <p>Removes a LookaheadHandle from the lookahead calculator.</p>
void	<p><a href="#"><code>PARALLEL_SetMinimumLookaheadForInterface</code></a>(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><a href="#"><code>PARALLEL_InitLookaheadCalculator</code></a>(LookaheadCalculator lookaheadCalculator)</p> <p>Initializes lookahead calculation. For kernel use only.</p>
int	<p><a href="#"><code>PARALLEL_AssignNodesToPartitions</code></a>(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><a href="#"><code>PARALLEL_GetPartitionForNode</code></a>(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><a href="#"><code>PARALLEL_InitializeParallelRuntime</code></a>(int numberOfThreads)</p> <p>Sets global variables and stuff. For kernel use only.</p>
void	<p><a href="#"><code>PARALLEL_CreatePartitionThreads</code></a>(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><a href="#"><code>PARALLEL_GetRemoteMessages</code></a>(PartitionData* partitionData)</p>
	<p>Collects all the messages received from other partitions. For kernel use only.</p>
void	<p><a href="#"><code>PARALLEL_GetRemoteMessagesAndBarrier</code></a>(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><a href="#"><code>PARALLEL_SendRemoteMessages</code></a>(Message* msgList, PartitionData* partitionData, int partitionId)</p>
	<p>Sends one or more messages to a remote partition. For kernel use only.</p>
	<p><a href="#"><code>PARALLEL_DeliverRemoteMessages</code></a>(PartitionData* partitionData)</p>
	<p>Delivers cached messages to all remote partitions. For kernel use only.</p>
void	<p><a href="#"><code>PARALLEL_SendRemoteMessagesOob</code></a>(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><a href="#"><code>PARALLEL_SendMessageToAllPartitions</code></a>(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><a href="#"><code>PARALLEL_SendRemoteLinkMessage</code></a>(Node* node, Message* msg, LinkData* link, clocktype txDelay)</p>
	<p>Sends one LINK message to a remote partition.</p>
void	<p><a href="#"><code>PARALLEL_UpdateSafeTime</code></a>(PartitionData* partitionData)</p>
	<p>A generic function for calculating the window of safe events For kernel use only.</p>
clocktype	<p><a href="#"><code>PARALLEL_ReturnEarliestGlobalEventTime</code></a>(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><a href="#"><code>PARALLEL_Exit</code></a>(PartitionData* partitionData)</p>
	<p>Exits from the parallel system, killing threads, etc. For kernel use only.</p>
void	<p><a href="#"><code>PARALLEL_SetProtocolIsNotEOTCapable</code></a>(Node* node)</p>

	<p>Currently, EOT can only be used if supported by all protocols running in the scenario. If any protocol is not capable, only the minimum lookahead is used.</p>
void	<p><a href="#"><code>PARALLEL_EnableDynamicMobility()</code></a></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>
void	<p><a href="#"><code>PARALLEL_SetGreedy(bool greedy)</code></a></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>
bool	<p><a href="#"><code>PARALLEL_IsGreedy()</code></a></p> <p>Checks whether SetGreedy has been called.</p>
void	<p><a href="#"><code>PARALLEL_Preflight(PartitionData* partitionData)</code></a></p> <p>Initializes parallel operation.</p>
void	<p><a href="#"><code>PARALLEL_ScheduleMessagesOnPartition(PartitionData* partitionData, Message* msgList, Message** oobMessage, bool* gotOobMessage, bool isMT)</code></a></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>
void	<p><a href="#"><code>PARALLEL_EndSimulation(PartitionData* partitionData)</code></a></p> <p>Shuts down the parallel engine, including whatever synchronization is required.</p>
void	<p><a href="#"><code>PARALLEL_BuildStatFile(int numPartitions, char* statFileName, char* experimentPrefix)</code></a></p> <p>Builds the final stat file when running in parallel node. Should only be called once from partition 0.</p>
void	<p><a href="#"><code>PARALLEL_NumberOfSynchronizations()</code></a></p> <p>Return the number of synchronizations performed per partition</p>
void	<p><a href="#"><code>PARALLEL_StartRealTimeThread(PartitionData* partitionData)</code></a></p> <p>Tells the kernel to use an independent thread to constantly update realtime.</p>

**Constant / Data Structure Detail**

Constant	<b>MAX_THREADS</b> 512  The maximum number of processes that can be used in parallel QualNet. Customers do not receive parallel.cpp, so cannot effectively change this value.
Enumeration	<b>SynchronizationAlgorithm</b>  Possible algorithms to use in the parallel runtime. Synchronous is used by default.
Enumeration	<b>BarrierType</b>  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.
Structure	<b>LookaheadLocator</b>  This struct is allows us to be able to remove from the LookaheadCalculator's heap. This way lookahead handles can request they be removed. Internally, as the heap re-heapifies these locators are updated.
Structure	<b>EotHeapElement</b>  Basic data structure for simplifying lookahead calculation.
Structure	<b>LookaheadCalculator</b>  Stores a heap of EOT elements to calculate lookahead.

**Function / Macro Detail**

Function / Macro	Format
<b>PARALLEL_AllocateLookaheadHandle</b>  Obtains a new lookahead handle that allows a protocol to indicate minimum delay values for output. This minimum delay is called EOT - earliest output time.	LookaheadHandle <b>PARALLEL_AllocateLookaheadHandle</b> (Node* node)  Parameters: <ul style="list-style-type: none"> <li>• node - the active node</li> </ul> Returns:

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

	<p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>PARALLEL_InitLookaheadCalculator</b>	<p>void <b>PARALLEL_InitLookaheadCalculator</b> (LookaheadCalculator lookaheadCalculator)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• lookaheadCalculator - the lookahead calculator</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>PARALLEL_AssignNodesToPartitions</b>	<p>int <b>PARALLEL_AssignNodesToPartitions</b> (int numNodes, int numberofPartitions, NodeInput* nodeInput, NodePosition* nodePos, AddressMapType* map)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• numNodes - the number of nodes</li> <li>• numberofPartitions - the number of partitions</li> <li>• nodeInput - the input configuration file</li> <li>• nodePos - the node positions</li> <li>• map - node ID &lt;--&gt; IP address mappings</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• int - the number of partitions used</li> </ul>
<b>PARALLEL_GetPartitionForNode</b>	<p>int <b>PARALLEL_GetPartitionForNode</b> (NodeId nodeId)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• nodeId - the node's ID</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• int - the partition to which the node is assigned</li> </ul>
<b>PARALLEL_InitializeParallelRuntime</b>	<p>void <b>PARALLEL_InitializeParallelRuntime</b> (int numberOfThreads)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• numberOfThreads - the number of processors to use.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>PARALLEL_CreatePartitionThreads</b>	<p>void <b>PARALLEL_CreatePartitionThreads</b> (int numberOfThreads, NodeInput* nodeInput,</p>

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

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

<b>PARALLEL_UpdateSafeTime</b>	<p>A generic function for calculating the window of safe events For kernel use only.</p>	<p>void <b>PARALLEL_UpdateSafeTime</b> (PartitionData* partitionData)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>partitionData - a pointer to the partition</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>void - None</li> </ul>
<b>PARALLEL_ReturnEarliestGlobalEventTime</b>	<p>Returns the earliest global event time. Required for interfacing to time-sensitive external programs. For kernel use only.</p>	<p>clocktype <b>PARALLEL_ReturnEarliestGlobalEventTime</b> (PartitionData* partitionData)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>partitionData - a pointer to the partition</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>clocktype - the time of the earliest event across all partitions</li> </ul>
<b>PARALLEL_Exit</b>	<p>Exits from the parallel system, killing threads, etc. For kernel use only.</p>	<p>void <b>PARALLEL_Exit</b> (PartitionData* partitionData)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>partitionData - a pointer to the partition</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>void - None</li> </ul>
<b>PARALLEL_SetProtocolIsNotEOTCapable</b>	<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 <b>PARALLEL_SetProtocolIsNotEOTCapable</b> (Node* node)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>node - the node's data</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>void - None</li> </ul>
<b>PARALLEL_EnableDynamicMobility</b>	<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 <b>PARALLEL_EnableDynamicMobility</b> ()</p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> <li>void - None</li> </ul>
<b>PARALLEL_SetGreedy</b>	<p>Tells the kernel to use spin locks on barriers if true, or to use blocking</p>	<p>void <b>PARALLEL_SetGreedy</b> (bool greedy)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>greedy - should it be greedy or not?</li> </ul>

<p>barriers otherwise. In greedy mode, the Simulator needs a dedicated CPU per partition.</p>	<p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<p><b>PARALLEL_IsGreedy</b></p> <p>Checks whether SetGreedy has been called.</p>	<p><code>bool PARALLEL_IsGreedy ()</code></p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>bool</code> - true if greedy mode is enabled.</li> </ul>
<p><b>PARALLEL_Preflight</b></p> <p>Initializes parallel operation.</p>	<p><code>void PARALLEL_Preflight (PartitionData* partitionData)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>partitionData</code> - the partition to initialize.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<p><b>PARALLEL_ScheduleMessagesOnPartition</b></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><code>void PARALLEL_ScheduleMessagesOnPartition (PartitionData* partitionData, Message* msgList, Message** oobMessage, bool* gotOobMessage, bool isMT)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>partitionData</code> - the partition.</li> <li>• <code>msgList</code> - a list of normal simulation messages.</li> <li>• <code>oobMessage</code> - an out of bounds message.</li> <li>• <code>gotOobMessage</code> - returns true if Oob response is received</li> <li>• <code>isMT</code> - is this called from a worker thread</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<p><b>PARALLEL_EndSimulation</b></p> <p>Shuts down the parallel engine, including whatever synchronization is required.</p>	<p><code>void PARALLEL_EndSimulation (PartitionData* partitionData)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>partitionData</code> - the partition to terminate.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
	<p><b>PARALLEL_BuildStatFile</b> (<code>int numPartitions, char* statFileName, char* experimentPrefix</code>)</p> <p>Parameters:</p>

	<p>Builds the final stat file when running in parallel node. Should only be called once from partition 0.</p> <ul style="list-style-type: none"> <li>• numPartitions - number of partitions</li> <li>• statFileName - name of stat file</li> <li>• experimentPrefix - experiment prefix</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>PARALLEL_NumberOfSynchronizations</b>  Return the number of synchronizations performed per partition	<p><b>void PARALLEL_NumberOfSynchronizations ()</b></p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>PARALLEL_StartRealTimeThread</b>  Tells the kernel to use an independent thread to constantly update realtime.	<p><b>void PARALLEL_StartRealTimeThread (PartitionData* partitionData)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• partitionData - a pointer to the partition</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>



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

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

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



# EXata 5.1 API Reference

## PARTITION

This file contains declarations of some functions for partition threads.

### Constant / Data Structure Summary

Type	Name
CONSTANT	<a href="#">NUM_SIM_TIME_STATUS_PRINTS</a>  The number of percentage complete statements to print
CONSTANT	<a href="#">COMMUNICATION_ID_INVALID</a>  A default uninitialized communication ID.
CONSTANT	<a href="#">COMMUNICATION_DELAY_REAL_TIME</a>  A value to indicate real time interpartition communication
ENUMERATION	<a href="#">PartitionGlobalDataIndex</a>  This enumeration contains indexes into the PartitionGlobalData array used for module data.
STRUCT	<a href="#">SubnetMemberData</a>  Data structure containing interfaceIndex and Node* for a node in a single subnet
STRUCT	<a href="#">PartitionSubnetMemberList</a>  Data structure containing member data info for all nodes in a single subnet
STRUCT	<a href="#">PartitionSubnetData</a>  Data structure containing subnet member data for all subnets
STRUCT	<a href="#">PartitionData</a>

	Contains global information for this partition.
STRUCT	<p><a href="#">SimulationProperties</a></p> <p>Global properties of the simulation for all partitions.</p>

**Function / Macro Summary**

Return Type	Summary
clocktype	<p><a href="#">getGlobalTime()</a></p> <p>Returns the simulation time at a global level For the current time of a node, use Node::getTime PartitionData::getGlobalTime should only be used for timing at the partition or global level.</p>
void	<p><a href="#">PARTITION_GetTerrainPtr</a>(PartitionData* partitionData)</p> <p>Inline function used to get terrainData pointer.</p>
void	<p><a href="#">PARTITION_CreateEmptyPartition</a>(int partitionId, int numPartitions)</p> <p>Function used to allocate and perform initialization of an empty partition data structure.</p>
void	<p><a href="#">PARTITION_InitializePartition</a>(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>Function used to initialize a partition.</p>
void	<p><a href="#">PARTITION_InitializeNodes</a>(PartitionData* partitionData)</p> <p>Function used to allocate and initialize the nodes on a partition.</p>
void	<p><a href="#">PARTITION_Finalize</a>(PartitionData* partitionData)</p> <p>Finalizes the nodes on the partition.</p>
void	<p><a href="#">PARTITION_ProcessPartition</a>(PartitionData* partitionData)</p> <p>Creates and initializes the nodes, then processes events on this partition.</p>
void	<p><a href="#">PARTITION_ProcessSendMT</a>(PartitionData* partitionData)</p> <p>Messages sent by worker threads outside of the main simulation event loop MUST call MESSAGE_SendMT(). This function then is</p>

	<p>the other half - where the multi-thread messages are properly added to the event list.</p>
BOOL	<p><a href="#"><b>PARTITION_ReturnNodePointer</b></a>(PartitionData* partitionData, Node** node, NodeId nodeId, BOOL remoteOK)</p> <p>Returns a pointer to the node or NULL if the node is not on this partition. If remoteOK is TRUE, returns a pointer to this partition's proxy for a remote node if the node does not belong to this partition. This feature should be used with great care, as the proxy is incomplete. Returns TRUE if the node was successfully found.</p>
void	<p><a href="#"><b>PARTITION_NodeExists</b></a>(PartitionData* partitionData, NodeId nodeId)</p> <p>Determines whether the node ID exists in the scenario. Must follow node creation.</p>
void	<p><a href="#"><b>PARTITION_PrintRunTimeStats</b></a>(PartitionData* partitionData)</p> <p>If dynamic statistics reporting is enabled, generates statistics for enabled layers.</p>
void	<p><a href="#"><b>PARTITION_SchedulePartitionEvent</b></a>(PartitionData* partitionData, Message* msg, clocktype eventTime, bool scheduleBeforeNodes)</p> <p>Schedules a generic partition-level event.</p>
void	<p><a href="#"><b>PARTITION_HandlePartitionEvent</b></a>(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><a href="#"><b>PARTITION_ClientStateSet</b></a>(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><a href="#"><b>PARTITION_ClientStateFind</b></a>(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><a href="#"><b>PARTITION_COMMUNICATION_RegisterCommunicator</b></a>(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><a href="#"><b>PARTITION_COMMUNICATION_FindCommunicator</b></a>(PartitionData* partitionData, std::string name)</p> <p>Locate an already registered commincator.</p>
void	<p><a href="#"><b>PARTITION_COMMUNICATION_SendToPartition</b></a>(PartitionData* partitionData, int partitionId, Message* msg, clocktype delay)</p>

## PARTITION

	Transmit a message to a partition.
void	<a href="#">PARTITION_COMMUNICATION_SendToAllPartitions</a> (PartitionData* partitionData, Message* msg, clocktype delay)
	Transmit a message to all partitions.
std	<a href="#">IO_Return_Qualnet_Directory()</a>
	This will return in a string the current directory qualnet is executing from
boolean true/false if file exists	<a href="#">IO_SourceFileExists()</a>
	This will return a boolean true if file exists, and false if not
std	<a href="#">IO_CheckSourceLibrary()</a>
	This will return in a string the formatted yes/no line for whether the fingerprint file exists for given library
std	<a href="#">IO_ReturnSourceAndCompiledLibraries()</a>
	This will return in a string a list of libraries currently compiled into product as well as those which have source code available.
std	<a href="#">IO_ReturnExpirationDateFromLicenseFeature()</a>
	This will return in a string a list of libraries currently compiled into product as well as those which have source code available.
std	<a href="#">IO_ReturnExpirationDateFromNumericalDate()</a>
	This will return in a string the expiration date of the library
std	<a href="#">IO_ReturnExpirationDateFromNumericalDate()</a>
	This will return in a string the expiration date of the library
UInt64 containing the date	<a href="#">IO_ParseFlexLMDate()</a>
	Parse a FlexLM date in a platform safe way
std	<a href="#">IO_ReturnStatusMessageFromLibraryInfo()</a>
	This will return in a string the status message for the library used with the -print_libraries option
std	<a href="#">IO_ReturnStatusMessageFromLibraryInfo()</a>

	<p>This will return in a string the library name from its index :: because flexlm won't allow std strucsts in main.cpp :: but main.cpp is the only place flex structs are allowed</p> <p><a href="#"><u>PARTITION_ShowProgress( )</u></a></p>
	Print standard QualNet progress log

**Constant / Data Structure Detail**

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

	Contains global information for this partition.
Structure	<p>SimulationProperties</p> <p>Global properties of the simulation for all partitions.</p>

**Function / Macro Detail**

Function / Macro	Format
<b>getGlobalTime</b>	<p>clocktype <b>getGlobalTime ()</b></p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>clocktype</code> - The current global simulation time</li> </ul>
>Returns the simulation time at a global level For the current time of a node, use <code>Node::getNodeTime</code> PartitionData::getGlobalTime should only be used for timing at the partition or global level.	
<b>PARTITION_GetTerrainPtr</b>	<p>void <b>PARTITION_GetTerrainPtr</b> (PartitionData* <code>partitionData</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>partitionData</code> - pointer to <code>partitionData</code></li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
Inline function used to get <code>terrainData</code> pointer.	
<b>PARTITION_CreateEmptyPartition</b>	<p>void <b>PARTITION_CreateEmptyPartition</b> (int <code>partitionId</code>, int <code>numPartitions</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>partitionId</code> - the partition ID, used for parallel</li> <li>• <code>numPartitions</code> - for parallel</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
Function used to allocate and perform initilaization of of an empty partition data structure.	
<b>PARTITION_InitializePartition</b>	<p>void <b>PARTITION_InitializePartition</b> (PartitionData* <code>partitionData</code>, TerrainData* <code>terrainData</code>, clocktype <code>maxSimClock</code>, double <code>startRealTime</code>, int <code>numNodes</code>, BOOL <code>traceEnabled</code>, AddressMapType* <code>addressMapPtr</code>, NodePositions* <code>nodePositions</code>, NodeInput* <code>nodeInput</code>, int <code>seedVal</code>, int* <code>nodePlacementTypeCounts</code>, char* <code>experimentPrefix</code>, clocktype <code>startSimClock</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>partitionData</code> - an empty partition data structure</li> <li>• <code>terrainData</code> - dimensions, terrain database, etc.</li> </ul>
Function used to initialize a partition.	

	<ul style="list-style-type: none"> <li>• maxSimClock - length of the scenario</li> <li>• startRealTime - for synchronizing with the realtime</li> <li>• numNodes - number of nodes in the simulation</li> <li>• traceEnabled - is packet tracing enabled?</li> <li>• addressMapPtr - contains Node ID &lt;--&gt; IP address mappings</li> <li>• nodePositions - initial node locations and partition assignments</li> <li>• nodeInput - contains all the input parameters</li> <li>• seedVal - the global random seed</li> <li>• nodePlacementTypeCounts - gives information about node placement</li> <li>• experimentPrefix - the experiment name</li> <li>• startSimClock - the simulation starting time</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>PARTITION_InitializeNodes</b>	<p>Function used to allocate and initialize the nodes on a partition.</p> <p>void <b>PARTITION_InitializeNodes</b> (PartitionData* partitionData)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• partitionData - an pre-initialized partition data structure</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>PARTITION_Finalize</b>	<p>Finalizes the nodes on the partition.</p> <p>void <b>PARTITION_Finalize</b> (PartitionData* partitionData)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• partitionData - an pre-initialized partition data structure</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>PARTITION_ProcessPartition</b>	<p>Creates and initializes the nodes, then processes events on this partition.</p> <p>void <b>PARTITION_ProcessPartition</b> (PartitionData* partitionData)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• partitionData - an pre-initialized partition data structure</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>

<b>PARTITION_ProcessSendMT</b>  Messages sent by worker threads outside of the main simulation event loop MUST call MESSAGE_SendMT(). This function then is the other half - where the multi-thread messages are properly added to the event list.	<p>void <b>PARTITION_ProcessSendMT</b> (PartitionData* partitionData)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>partitionData - an pre-initialized partition data structure</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>void - None</li> </ul>
<b>PARTITION_ReturnNodePointer</b>  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>BOOL <b>PARTITION_ReturnNodePointer</b> (PartitionData* partitionData, Node** node, NodeId nodeId, BOOL remoteOK)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>partitionData - an pre-initialized partition data structure</li> <li>node - for returning the node pointer</li> <li>nodeId - the node's ID</li> <li>remoteOK - is it ok to return a pointer to proxy node?</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>BOOL - returns TRUE if the node was successfully found</li> </ul>
<b>PARTITION_NodeExists</b>  Determines whether the node ID exists in the scenario. Must follow node creation.	<p>void <b>PARTITION_NodeExists</b> (PartitionData* partitionData, NodeId nodeId)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>partitionData - an pre-initialized partition data structure</li> <li>nodeId - the node's ID</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>void - None</li> </ul>
<b>PARTITION_PrintRunTimeStats</b>  If dynamic statistics reporting is enabled, generates statistics for enabled layers.	<p>void <b>PARTITION_PrintRunTimeStats</b> (PartitionData* partitionData)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>partitionData - an pre-initialized partition data structure</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>void - None</li> </ul>
<b>PARTITION_SchedulePartitionEvent</b>  Schedules a generic partition-level event.	<p>void <b>PARTITION_SchedulePartitionEvent</b> (PartitionData* partitionData, Message* msg, clocktype eventTime, bool scheduleBeforeNodes)</p> <p>Parameters:</p>

	<ul style="list-style-type: none"> <li>partitionData - an pre-initialized partition data structure</li> <li>msg - an event</li> <li>eventTime - the time the event should occur</li> <li>scheduleBeforeNodes - process event before or after node events</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>void - None</li> </ul>
<b>PARTITION_HandlePartitionEvent</b>	<p>void <b>PARTITION_HandlePartitionEvent</b> (PartitionData* partitionData, Message* msg)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>partitionData - an pre-initialized partition data structure</li> <li>msg - an event</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>void - None</li> </ul>
<b>PARTITION_ClientStateSet</b>	<p>void <b>PARTITION_ClientStateSet</b> (PartitionData* partitionData, const char* stateName, void* clientState)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>partitionData - an pre-initialized partition data structure</li> <li>stateName - Name used to locate this client state</li> <li>clientState - Pointer to whatever data-structure the</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>void - None</li> </ul>
<b>PARTITION_ClientStateFind</b>	<p>void* <b>PARTITION_ClientStateFind</b> (PartitionData* partitionData, const char* stateName)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>partitionData - an pre-initialized partition data structure</li> <li>stateName - Name used to locate this client state</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>void* - returns the client state</li> </ul>
<b>PARTITION_COMMUNICATION_RegisterCommunicator</b>	<p>CommunicatorId <b>PARTITION_COMMUNICATION_RegisterCommunicator</b> (PartitionData* partitionData, const char* name, PartitionCommunicationHandler handler)</p> <p>Parameters:</p>

<p>invoked to receive callbacks when messages are with the id are sent.</p>	<ul style="list-style-type: none"> <li>• <code>partitionData</code> - an pre-initialized partition data structure</li> <li>• <code>name</code> - Your name for this type of message.</li> <li>• <code>handler</code> - Function</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>CommunicatorId</code> - used to later when calling MESSAGE_Alloc().</li> </ul>
<p><b>PARTITION_COMMUNICATION_FindCommunicator</b></p> <p>Locate an already registered commincator.</p>	<p>CommunicatorId <b>PARTITION_COMMUNICATION_FindCommunicator</b> (PartitionData* <code>partitionData</code>, std <code>name</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>partitionData</code> - an pre-initialized partition data structure</li> <li>• <code>name</code> - string</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>CommunicatorId</code> - found communicator Id or COMMUNICATION_ID_INVALID if not found.</li> </ul>
<p><b>PARTITION_COMMUNICATION_SendToPartition</b></p> <p>Transmit a message to a partition.</p>	<p>void <b>PARTITION_COMMUNICATION_SendToPartition</b> (PartitionData* <code>partitionData</code>, int <code>partitionId</code>, Message* <code>msg</code>, clocktype <code>delay</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>partitionData</code> - an pre-initialized partition data structure</li> <li>• <code>partitionId</code> - partition to send the message to</li> <li>• <code>msg</code> - Message to send. You are required to follow</li> <li>• <code>delay</code> - When the message should execute. Special delay</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<p><b>PARTITION_COMMUNICATION_SendToAllPartitions</b></p> <p>Transmit a message to all partitions.</p>	<p>void <b>PARTITION_COMMUNICATION_SendToAllPartitions</b> (PartitionData* <code>partitionData</code>, Message* <code>msg</code>, clocktype <code>delay</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>partitionData</code> - an pre-initialized partition data structure</li> <li>• <code>msg</code> - Message to send. You are required to follow</li> <li>• <code>delay</code> - When the message should execute. Special delay</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>

<b>IO_Return_Qualnet_Directory</b>	<p>This will return in a string the current directory qualnet is executing from</p>	<p><b>std IO_Return_Qualnet_Directory ()</b></p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> <li>• std - string containing current qualnet directory</li> </ul>
<b>IO_SourceFileExists</b>	<p>This will return a boolean true if file exists, and false if not</p>	<p>boolean true/false if file exists <b>IO_SourceFileExists ()</b></p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> <li>• boolean true/false if file exists - None</li> </ul>
<b>IO_CheckSourceLibrary</b>	<p>This will return in a string the formatted yes/no line for whether the fingerprint file exists for given library</p>	<p><b>std IO_CheckSourceLibrary ()</b></p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> <li>• std - string containing list of libraries</li> </ul>
<b>IO_ReturnSourceAndCompiledLibraries</b>	<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><b>std IO_ReturnSourceAndCompiledLibraries ()</b></p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> <li>• std - string containing list of libraries</li> </ul>
<b>IO_ReturnExpirationDateFromLicenseFeature</b>	<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><b>std IO_ReturnExpirationDateFromLicenseFeature ()</b></p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> <li>• std - string containing expiration date for this feature</li> </ul>
<b>IO_ReturnExpirationDateFromNumericalDate</b>	<p>This will return in a string the expiration date of the library</p>	<p><b>std IO_ReturnExpirationDateFromNumericalDate ()</b></p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> <li>• std - string containing expiration date for this feature</li> </ul>
<b>IO_ReturnExpirationDateFromNumericalDate</b>	<p>This will return in a string the expiration date of the library</p>	<p><b>std IO_ReturnExpirationDateFromNumericalDate ()</b></p> <p>Parameters:</p> <p>Returns:</p>

	<code>std</code> - string containing expiration date for this feature
<b>IO_ParseFlexLMDate</b>  Parse a FlexLM date in a platform safe way	UInt64 containing the date <b>IO_ParseFlexLMDate ()</b>  Parameters:  Returns: <ul style="list-style-type: none"><li>• UInt64 containing the date - None</li></ul>
<b>IO_ReturnStatusMessageFromLibraryInfo</b>  This will return in a string the status message for the library used with the -print_libraries option	<code>std IO_ReturnStatusMessageFromLibraryInfo ()</code>  Parameters:  Returns: <ul style="list-style-type: none"><li>• <code>std</code> - string containing status message for library</li></ul>
<b>IO_ReturnStatusMessageFromLibraryInfo</b>  This will return in a string the library name from its index :: because flexlm won't allow std strucsts in main.cpp :: but main.cpp is the only place flex structs are allowed	<code>std IO_ReturnStatusMessageFromLibraryInfo ()</code>  Parameters:  Returns: <ul style="list-style-type: none"><li>• <code>std</code> - string containing library name</li></ul>
<b>PARTITION_ShowProgress</b>  Print standard QualNet progress log	<b>PARTITION_ShowProgress ()</b>  Parameters:  Returns: <ul style="list-style-type: none"><li>• -</li></ul>



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

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

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



# EXata 5.1 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	<a href="#">PHY_DEFAULT_NOISE_FACTOR</a> Default noise factor in physical medium
CONSTANT	<a href="#">PHY_DEFAULT_TEMPERATURE</a> Default temperature of physical medium.
CONSTANT	<a href="#">PHY_DEFAULT_MIN_PCOM_VALUE</a> Default minimum pcom value threshold
CONSTANT	<a href="#">PHY_DEFAULT_SYNC_COLLISION_WINDOW</a> Default minimum pcom value threshold
ENUMERATION	<a href="#">PhyModel</a> Different phy types supported.
ENUMERATION	<a href="#">PhyRxModel</a> Different types of packet reception model
STRUCT	<a href="#">PhyBerEntry</a> SNR/BER curve entry
STRUCT	<a href="#">PhyBerTable</a>

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

**Function / Macro Summary**

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

## PHYSICAL LAYER

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

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

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

## PHYSICAL LAYER

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

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

## PHYSICAL LAYER

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

**Constant / Data Structure Detail**

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

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

**Function / Macro Detail**

Function / Macro	Format
<b>PHY_GlobalBerInit</b>	<pre>void <b>PHY_GlobalBerInit</b> (const NodeInput* nodeInput)</pre> <p>Parameters:</p>

<p>Pre-load all the BER files.</p>	<p><code>nodeInput</code> - structure containing contents of input file</p> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<p><b>PHY_GetSnrBerTableByName</b></p> <p>Get a pointer to a specific BER table.</p>	<p><code>void PHY_GetSnrBerTableByName (char* tableName)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>tableName</code> - name of the BER file</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<p><b>PHY_GetSnrBerTableIndex</b></p> <p>Get a index of BER table used by PHY.</p>	<p><code>int PHY_GetSnrBerTableIndex (Node* node, int phyIndex)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Node pointer</li> <li>• <code>phyIndex</code> - interface Index</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>int</code> - None</li> </ul>
<p><b>PHY_SetSnrBerTableIndex</b></p> <p>Set index of BER table to be used by PHY.</p>	<p><code>int PHY_SetSnrBerTableIndex (Node* node, int phyIndex)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Node pointer</li> <li>• <code>phyIndex</code> - interface Index</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>int</code> - None</li> </ul>
<p><b>PHY_GlobalPerInit</b></p> <p>Pre-load all the PER files.</p>	<p><code>void PHY_GlobalPerInit (const NodeInput* nodeInput)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>nodeInput</code> - structure containing contents of input file</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<p><b>PHY_GetSnrPerTableByName</b></p> <p>Get a pointer to a specific PER table.</p>	<p><code>void PHY_GetSnrPerTableByName (char* tableName)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>tableName</code> - name of the PER file</li> </ul>

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

	<ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>PHY_Finalize</b>	<p>void <b>PHY_Finalize</b> (Node * node)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - node for which results are to be collected</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>PHY_ProcessEvent</b>	<p>void <b>PHY_ProcessEvent</b> (Node* node, Message* msg)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - node which received the message</li> <li>• msg - message received by the layer</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>PHY_GetStatus</b>	<p>PhyStatusType <b>PHY_GetStatus</b> (Node * node, int phyNum)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - node for which stats are to be collected</li> <li>• phyNum - interface for which stats are to be collected</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• PhyStatusType - status of interface.</li> </ul>
<b>PHY_SetTransmitPower</b>	<p>void <b>PHY_SetTransmitPower</b> (Node * node, int phyIndex, double newTxPower_mW)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - node for which transmit power is to be set</li> <li>• phyIndex - interface for which transmit power is to be set</li> <li>• newTxPower_mW - transmit power(mW)</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>PHY_SetRxSNRThreshold</b>	<p>void <b>PHY_SetRxSNRThreshold</b> (Node * node, int phyIndex, double snr)</p> <p>Parameters:</p>

Sets the Radio's Rx SNR Threshold	<ul style="list-style-type: none"> <li>• <code>node</code> - node for which transmit power is to be set</li> <li>• <code>phyIndex</code> - interface for which transmit power is to be set</li> <li>• <code>snr</code> - threshold value to be set</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>PHY_SetDataRate</b>	<p><code>void PHY_SetDataRate (Node * node, int phyIndex, Int64 dataRate)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - node for which transmit power is to be set</li> <li>• <code>phyIndex</code> - interface for which transmit power is to be set</li> <li>• <code>dataRate</code> - dataRate value to be set</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>PHY_SetTxDataRate</b>	<p><code>void PHY_SetTxDataRate (Node * node, int phyIndex, Int64 dataRate)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - node for which transmit power is to be set</li> <li>• <code>phyIndex</code> - interface for which transmit power is to be set</li> <li>• <code>dataRate</code> - dataRate value to be set</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>PHY_SetRxDataRate</b>	<p><code>void PHY_SetRxDataRate (Node * node, int phyIndex, Int64 dataRate)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - node for which transmit power is to be set</li> <li>• <code>phyIndex</code> - interface for which transmit power is to be set</li> <li>• <code>dataRate</code> - dataRate value to be set</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
For radios that support different Tx and Rx data rates, this will set the Rx Data Rate. For others, it will call <code>PHY_SetDataRate</code> .	
For radios that support different Tx and Rx data rates, this will set the Rx Data Rate. For others, it will call <code>PHY_SetDataRate</code> .	
<b>PHY_GetTransmitPower</b>	<p><code>void PHY_GetTransmitPower (Node * node, int phyIndex, double* txPower_mW)</code></p> <p>Parameters:</p>

<p>Gets the Radio's transmit power in mW.</p>	<ul style="list-style-type: none"> <li>• <code>node</code> - Node that is</li> <li>• <code>phyIndex</code> - interface index.</li> <li>• <code>txPower_mW</code> - transmit power(mW)</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<p><b>PHY_GetTransmissionDelay</b></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 <b>PHY_GetTransmissionDelay</b> (<code>Node * node, int phyIndex, int size</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - node pointer to node</li> <li>• <code>phyIndex</code> - interface index</li> <li>• <code>size</code> - size of the frame in bytes</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>clocktype</code> - transmission delay.</li> </ul>
<p><b>PHY_GetTransmissionDuration</b></p> <p>Get transmission duration of a structured signal fragment.</p>	<p>clocktype <b>PHY_GetTransmissionDuration</b> (<code>Node * node, int phyIndex, int dataRateIndex, int size</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - node pointer to node</li> <li>• <code>phyIndex</code> - interface index.</li> <li>• <code>dataRateIndex</code> - data rate.</li> <li>• <code>size</code> - size of frame in bytes.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>clocktype</code> - transmission duration</li> </ul>
<p><b>PHY_GetFrameModel</b></p> <p>Get Physical Model</p>	<p>PhyModel <b>PHY_GetFrameModel</b> (<code>Node * node, int phyNum</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - node pointer to node</li> <li>• <code>phyNum</code> - interface index</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>PhyModel</code> - Physical Model</li> </ul>
<p><b>PHY_GetAntennaModelType</b></p>	<p>PhyModel <b>PHY_GetAntennaModelType</b> (<code>Node * node, int phyNum</code>)</p>

	<p><b>Get Antenna Model type</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - node pointer to node</li> <li>• phyNum - interface index</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• PhyModel - Physical Model</li> </ul>
<p><b>PHY_StartTransmittingSignal</b></p> <p>Starts transmitting a packet.</p>	<p>void <b>PHY_StartTransmittingSignal</b> (Node * node, int phyNum, Message* msg, BOOL useMacLayerSpecifiedDelay, clocktype delayUntilAirborne)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - node pointer to node</li> <li>• phyNum - interface index</li> <li>• msg - packet to be sent</li> <li>• useMacLayerSpecifiedDelay - use delay specified by MAC</li> <li>• delayUntilAirborne - delay until airborne</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<p><b>PHY_StartTransmittingSignal</b></p> <p>Starts transmitting a packet. Function is being overloaded</p>	<p>void <b>PHY_StartTransmittingSignal</b> (Node * node, int phyNum, Message* msg, clocktype duration, BOOL useMacLayerSpecifiedDelay, clocktype delayUntilAirborne)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - node pointer to node</li> <li>• phyNum - interface index</li> <li>• msg - packet to be sent</li> <li>• duration - specified transmission delay</li> <li>• useMacLayerSpecifiedDelay - use delay specified by MAC</li> <li>• delayUntilAirborne - delay until airborne</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<p><b>PHY_StartTransmittingSignal</b></p> <p>Starts transmitting a packet.</p>	<p>void <b>PHY_StartTransmittingSignal</b> (Node * node, int phyNum, Message* msg, int bitSize, BOOL useMacLayerSpecifiedDelay, clocktype delayUntilAirborne)</p> <p>Parameters:</p>

	<ul style="list-style-type: none"> <li>• node - node pointer to node</li> <li>• phyNum - interface index</li> <li>• msg - packet to be sent</li> <li>• bitSize - specified size of the packet in bits</li> <li>• useMacLayerSpecifiedDelay - use delay specified by MAC</li> <li>• delayUntilAirborne - delay until airborne</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>PHY_SignalArrivalFromChannel</b>	<p>void <b>PHY_SignalArrivalFromChannel</b> (Node * node, int phyIndex, int channelIndex, PropRxInfo* propRxInfo)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - node pointer to node</li> <li>• phyIndex - interface index</li> <li>• channelIndex - channel index</li> <li>• propRxInfo - information on the arrived signal</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
Called when a new signal arrives	
<b>PHY_SignalEndFromChannel</b>	<p>void <b>PHY_SignalEndFromChannel</b> (Node * node, int phyIndex, int channelIndex, PropRxInfo* propRxInfo)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - node pointer to node</li> <li>• phyIndex - interface index</li> <li>• channelIndex - channel index</li> <li>• propRxInfo - information on the arrived signal</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
Called when the current signal ends	
<b>PHY_GetTxDataRate</b>	<p>Int64 <b>PHY_GetTxDataRate</b> (Node * node, int phyIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - node pointer to node</li> <li>• phyIndex - interface index</li> </ul>
Get transmission data rate	

	<p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>Int64</code> - None</li> </ul>
<b>PHY_GetRxDataRate</b>	<p><code>Int64 PHY_GetRxDataRate (Node * node, int phyIndex)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - node pointer to node</li> <li>• <code>phyIndex</code> - interface index</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>Int64</code> - None</li> </ul>
<b>PHY_GetTxDataRateType</b>	<p><code>int PHY_GetTxDataRateType (Node * node, int phyIndex)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - node pointer to node</li> <li>• <code>phyIndex</code> - interface index</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>int</code> - None</li> </ul>
<b>PHY_GetRxDataRateType</b>	<p><code>int PHY_GetRxDataRateType (Node * node, int phyIndex)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - node pointer to node</li> <li>• <code>phyIndex</code> - interface index</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>int</code> - None</li> </ul>
<b>PHY_SetTxDataRateType</b>	<p><code>void PHY_SetTxDataRateType (Node * node, int phyIndex, int dataRateType)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - node pointer to node</li> <li>• <code>phyIndex</code> - interface index</li> <li>• <code>dataRateType</code> - rate of data</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>

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

<p>Get the highest transmission data rate type for broadcast</p>	<ul style="list-style-type: none"> <li>• node - node pointer to node</li> <li>• phyIndex - interface index</li> <li>• dataRateType - rate of data</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<p><b>PHY_SetHighestTxDataRateTypeForBC</b></p> <p>Set the highest transmission data rate type for broadcast</p>	<p><b>void PHY_SetHighestTxDataRateTypeForBC (Node* node, int phyIndex)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - node pointer to node</li> <li>• phyIndex - interface index</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<p><b>PHY_ComputeSINR</b></p> <p>Compute SINR</p>	<p><b>double PHY_ComputeSINR (PhyData * phyData, double * signalPower_mW, double* interferencePower_mW, int bandwidth)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• phyData - PHY layer data</li> <li>• signalPower_mW - Signal power</li> <li>• interferencePower_mW - Interference power</li> <li>• bandwidth - Bandwidth</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• double - Signal to Interference and Noise Ratio</li> </ul>
<p><b>PHY_SignalInterference</b></p> <p>Compute Power from the desired signal and interference</p>	<p><b>void PHY_SignalInterference (Node* node, int phyIndex, int channelIndex, Message * msg, double * signalPower_mW, double* interferencePower_mW)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Node that is being</li> <li>• phyIndex - interface number</li> <li>• channelIndex - channel index</li> <li>• msg - message including desired signal</li> <li>• signalPower_mW - power from the desired signal</li> </ul>

	<ul style="list-style-type: none"> <li>• <code>interferencePower_mW</code> - power from interfering signals</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>PHY_BER</b>	<p><code>double PHY_BER (PhyData * phyData, int berTableIndex, double sinr)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>phyData</code> - PHY layer data</li> <li>• <code>berTableIndex</code> - index for BER tables</li> <li>• <code>sinr</code> - Signal to Interference and Noise Ratio</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>double</code> - Bit Error Rate</li> </ul>
<b>PHY_SER</b>	<p><code>double PHY_SER (PhyData * phyData, int perTableIndex, double sinr)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>phyData</code> - PHY layer data</li> <li>• <code>perTableIndex</code> - index for SER tables</li> <li>• <code>sinr</code> - Signal to Interference and Noise Ratio</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>double</code> - Packet Error Rate</li> </ul>
<b>PHY_StopListeningToChannel</b>	<p><code>void PHY_StopListeningToChannel (Node* node, int phyIndex, int channelIndex)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Node that is being</li> <li>• <code>phyIndex</code> - interface number</li> <li>• <code>channelIndex</code> - channel index</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>PHY_CanListenToChannel</b>	<p><code>BOOL PHY_CanListenToChannel (Node* node, int phyIndex, int channelIndex)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Node that is being</li> <li>• <code>phyIndex</code> - interface number</li> </ul>
Check if it can listen to the channel	

	<ul style="list-style-type: none"> <li>• <code>channelIndex</code> - channel index</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>BOOL</code> - None</li> </ul>
<b>PHY_IsListeningToChannel</b>	<p><code>BOOL PHY_IsListeningToChannel (Node* node, int phyIndex, int channelIndex)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Node that is being</li> <li>• <code>phyIndex</code> - interface number</li> <li>• <code>channelIndex</code> - channel index</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>BOOL</code> - None</li> </ul>
<b>PHY_SetTransmissionChannel</b>	<p><code>void PHY_SetTransmissionChannel (Node* node, int phyIndex, int channelIndex)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Node that is being</li> <li>• <code>phyIndex</code> - interface number</li> <li>• <code>channelIndex</code> - channel index</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>PHY_GetTransmissionChannel</b>	<p><code>void PHY_GetTransmissionChannel (Node* node, int phyIndex, int channelIndex)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Node that is being</li> <li>• <code>phyIndex</code> - interface number</li> <li>• <code>channelIndex</code> - channel index</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>PHY_MediumIsIdle</b>	<p><code>BOOL PHY_MediumIsIdle (Node* node, int phyNum)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Node that is being</li> </ul>

	<ul style="list-style-type: none"> <li>• phyNum - interface number</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• BOOL - None</li> </ul>
<b>PHY_MediumIsIdleInDirection</b>	<p>BOOL <b>PHY_MediumIsIdleInDirection</b> (Node* node, int phyNum, double azimuth)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Node that is being</li> <li>• phyNum - interface number</li> <li>• azimuth - azimuth (in degrees)</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• BOOL - None</li> </ul>
<b>PHY_SetSensingDirection</b>	<p>void <b>PHY_SetSensingDirection</b> (Node* node, int phyNum, double azimuth)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Node that is being</li> <li>• phyNum - interface number</li> <li>• azimuth - azimuth (in degrees)</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>PHY_StartTransmittingSignalDirectionally</b>	<p>void <b>PHY_StartTransmittingSignalDirectionally</b> (Node* node, int phyNum, Message* msg, BOOL useMacLayerSpecifiedDelay, clocktype delayUntilAirborne, double directionAzimuth)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Node that is</li> <li>• phyNum - interface number</li> <li>• msg - signal to transmit</li> <li>• useMacLayerSpecifiedDelay - use delay specified by MAC</li> <li>• delayUntilAirborne - delay until airborne</li> <li>• directionAzimuth - azimuth to transmit the signal</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>

<b>PHY_LockAntennaDirection</b>	<p><b>void PHY_LockAntennaDirection (Node* node, int phyNum)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Node that is being</li> <li>• phyNum - interface number</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>PHY_UnlockAntennaDirection</b>	<p><b>void PHY_UnlockAntennaDirection (Node* node, int phyNum)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Node that is being</li> <li>• phyNum - interface number</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>PHY_GetLastSignalsAngleOfArrival</b>	<p><b>double PHY_GetLastSignalsAngleOfArrival (Node* node, int phyNum)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Node that is being</li> <li>• phyNum - interface number</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• double - AOA</li> </ul>
<b>PHY_TerminateCurrentReceive</b>	<p><b>void PHY_TerminateCurrentReceive (Node* node, int phyNum, const BOOL terminateOnlyOnReceiveError, BOOL* receiveError, clocktype* endSignalTime)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Node pointer that the</li> <li>• phyNum - interface number</li> <li>• terminateOnlyOnReceiveError - terminate only when</li> <li>• receiveError - if error happened</li> <li>• endSignalTime - end of signal</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>

<b>PHY_PropagationRange</b>  Calculates an estimated radio range for the PHY. Supports only TWO-RAY and FREE-SPACE.	<p>double <b>PHY_PropagationRange</b> (Node* txnode, Node* node, int txInterfaceIndex, int interfaceIndex, int channelIndex, BOOL printAll)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>txnode - the Tx node of interest</li> <li>node - the Rx node of interest</li> <li>txInterfaceIndex - the interface for the TX node</li> <li>interfaceIndex - the interface for the Rx node</li> <li>channelIndex - the index of the channel</li> <li>printAll - if TRUE, prints the range for all data</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>double - the range in meters</li> </ul>
<b>ENERGY_Init</b>  This function declares energy model variables and initializes them. Moreover, the function reads energy model specifications and configures the parameters which are configurable.	<p>void <b>ENERGY_Init</b> (Node* node, const int phyIndex, NodeInput* nodeInput)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>node - the node of interest.</li> <li>phyIndex - the PHY index.</li> <li>nodeInput - the node input.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>void - None</li> </ul>
<b>ENERGY_PrintStats</b>  To print the statistic of Energy Model.	<p>void <b>ENERGY_PrintStats</b> (Node* node, const int phyIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>node - the node of interest.</li> <li>phyIndex - the PHY index.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>void - None</li> </ul>
<b>Phy_ReportStatusToEnergyModel</b>  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	<p>void <b>Phy_ReportStatusToEnergyModel</b> (Node* node, const int phyIndex, PhyStatusType prevStatus, PhyStatusType newStatus)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>node - the node of interest.</li> <li>phyIndex - the PHY index.</li> </ul>

<p>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>	<ul style="list-style-type: none"> <li>• <code>prevStatus</code> - the the previous status.</li> <li>• <code>newStatus</code> - the the new status.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<p><b>Generic_UpdateCurrentLoad</b></p> <p>To update the current load of generic energy model.</p>	<p><code>void Generic_UpdateCurrentLoad (Node* node, const int phyIndex)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - the node of interest.</li> <li>• <code>phyIndex</code> - the PHY index.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<p><b>PHY_NotificationOfPacketDrop</b></p> <p>To Notify the StatsDB module and other modules of the packet dropping event.</p>	<p><code>void PHY_NotificationOfPacketDrop (Node* node, int phyIndex, int channelIndex, const Message* msg, const string&amp; dropType, double rxPower_mW, double interferencePower_mW, double passloss_dB)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - the node of interest.</li> <li>• <code>phyIndex</code> - the PHY index.</li> <li>• <code>channelIndex</code> - the channelIndex</li> <li>• <code>msg</code> - The dropped message</li> <li>• <code>dropType</code> - the reason for the drop</li> <li>• <code>rxPower_mW</code> - receving power of the signal</li> <li>• <code>interferencePower_mW</code> - interference power of the signal</li> <li>• <code>passloss_dB</code> - pathloss value of the signal</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<p><b>PHY_NotificationOfSignalReceived</b></p> <p>To Notify the StatsDB module and other modules of the signal received event .</p>	<p><code>void PHY_NotificationOfSignalReceived (Node* node, int phyIndex, int channelIndex, const Message* msg, double rxPower_mW, double interferencePower_mW, double passloss_dB, int controlSize)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - the node of interest.</li> <li>• <code>phyIndex</code> - the PHY index.</li> </ul>

	<ul style="list-style-type: none"> <li>• <code>channelIndex</code> - the channelIndex</li> <li>• <code>msg</code> - The dropped message</li> <li>• <code>rxPower_mW</code> - receiving power of the signal</li> <li>• <code>interferencePower_mW</code> - interference power of the signal</li> <li>• <code>passloss_dB</code> - pathloss value of the signal</li> <li>• <code>controlSize</code> - size of control header</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>PHY_GetSteeringAngle</b>	<p>void <b>PHY_GetSteeringAngle</b> (Node* node, int phyIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - node being used</li> <li>• <code>phyIndex</code> - physical to be initialized</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>PHY_GetBandwidth</b>	<p>double <b>PHY_GetBandwidth</b> (Node* node, int phyIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - The node of interest.</li> <li>• <code>phyIndex</code> - The PHY index.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>double</code> - The bandwidth</li> </ul>
<b>PHY_GetFrequency</b>	<p>double <b>PHY_GetFrequency</b> (Node* node, int channelIndex)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - The node of interest.</li> <li>• <code>channelIndex</code> - Index of the propagation channel</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>double</code> - The frequency</li> </ul>
<b>PHY_GetPhyModel</b>	<p>PhyModel <b>PHY_GetPhyModel</b> (Node* node, int phyIndex)</p> <p>Parameters:</p>

	<p>To get the PhyModel for the node.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - The node of interest.</li> <li>• <code>phyIndex</code> - The PHY index.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>PhyModel</code> - The PhyModel</li> </ul>
<b>PHY_GetPhyModel</b>	<p><code>std PHY_GetPhyModel (Node* node, int phyIndex)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - The node of interest.</li> <li>• <code>phyIndex</code> - The PHY index.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>std</code> - string</li> </ul>
<b>PHY_isSignalFeatureMatchReceiverPhyModel</b>	<p><code>BOOL PHY_isSignalFeatureMatchReceiverPhyModel (Node* node, int phyIndex)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - The node of interest.</li> <li>• <code>phyIndex</code> - The PHY index.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>BOOL</code> - if the signal feature matches the receiver's phyModel</li> </ul>
<b>PHY_ComputeInbandPower</b>	<p><code>double PHY_ComputeInbandPower (double signalPower_mW, double signalFrequency, double signalBandwidth, double rxFrequency, double rxBandwidth)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>signalPower_mW</code> - The signal power in mW.</li> <li>• <code>signalFrequency</code> - The signal frequency in Hz.</li> <li>• <code>signalBandwidth</code> - The signal bandwidth in Hz</li> <li>• <code>rxFrequency</code> - The receiver frequency in Hz</li> <li>• <code>rxBandwidth</code> - The receiver bandwidth in Hz</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>double</code> - The inband signal power in mW</li> </ul>
<b>PHY_InterferenceArrivalFromChannel</b>	<p><code>void PHY_InterferenceArrivalFromChannel (Node * node, int phyIndex, int channelIndex, PropRxInfo* propRxInfo, double sigPower_mW)</code></p>

<p>Called when a interference signal arrives</p>	<p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• node - node pointer to node</li> <li>• phyIndex - interface index</li> <li>• channelIndex - channel index</li> <li>• propRxInfo - information on the arrived signal</li> <li>• sigPower_mW - The inband interference power in mW</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<p><b>PHY_InterferenceEndFromChannel</b></p> <p>Called when a interference signal ends</p>	<p><b>void <code>PHY_InterferenceEndFromChannel</code></b> (Node * node, int phyIndex, int channelIndex, PropRxInfo* propRxInfo, double sigPower_mW)</p> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• node - node pointer to node</li> <li>• phyIndex - interface index</li> <li>• channelIndex - channel index</li> <li>• propRxInfo - information on the arrived signal</li> <li>• sigPower_mW - The inband interference power in mW</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<p><b>PHY_GetChannelName</b></p> <p>To get the name of the channel.</p>	<p><b>std <code>PHY_GetChannelName</code></b> (Node* node, int channelIndex)</p> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• node - The node of interest.</li> <li>• channelIndex - Index of the propagation channel</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>• std - string</li> </ul>
<p><b>PHY_GetChannelIndexForChannelName</b></p> <p>To get the channel index.</p>	<p><b>Int32 <code>PHY_GetChannelIndexForChannelName</code></b> (Node* node, std channelName)</p> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• node - The node of interest.</li> <li>• channelName - string</li> </ul>

	<p>Returns:</p> <ul style="list-style-type: none"> <li><code>Int32</code> - Channel index.</li> </ul>
<b>PHY_ChannelNameExists</b>	<p><code>BOOL PHY_ChannelNameExists (Node* node)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li><code>node</code> - The node of interest.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li><code>BOOL</code> - TRUE if <code>channelName</code> is valid False if channel name is invalid</li> </ul>



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

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

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



# EXata 5.1 API Reference

## PROPAGATION

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

### Constant / Data Structure Summary

Type	Name
CONSTANT	<a href="#">BOLTZMANN_CONSTANT</a>  Boltzmann constant
CONSTANT	<a href="#">NEGATIVE_PATHLOSS_dB</a>  Path loss in dB (used as an invalid value)
CONSTANT	<a href="#">SPEED_OF_LIGHT</a>  Defines the value of speed of light
CONSTANT	<a href="#">PROP_DEFAULT_PROPAGATION_LIMIT_dBm</a>  Default value for propagation limit.
CONSTANT	<a href="#">PROP_DEFAULT_SHADOWING_MEAN_dB</a>  Default mean value for shadowing in dB
CONSTANT	<a href="#">MAX_NUM_ELEVATION_SAMPLES</a>  Maximum number of sample would be taken.
CONSTANT	<a href="#">PROP_DEFAULT_BANDWIDTH_FACTOR</a>  The bandwidth factor that is used to get the half sum bandwidth.
ENUMERATION	<a href="#">PathlossModel</a>

## PROPAGATION

	Different type of path loss. <a href="#">ShadowingModel</a>
ENUMERATION	Different type of shadowing used. <a href="#">FadingModel</a>
ENUMERATION	Different type of fading used. <a href="#">propagationEnvironment</a>
ENUMERATION	Different type of propagation environment. <a href="#">LoSIndicator</a>
ENUMERATION	Indicated if the path is Line of sight OR non-Line of sight <a href="#">SuburbanTerrainType</a>
ENUMERATION	Terrain types for Suburban-foliage model <a href="#">IndoorLinkType</a>
ENUMERATION	Link types for Indoor model <a href="#">LinkType</a>
STRUCT	Link types for model <a href="#">PropPathProfile</a>
STRUCT	Structure that keeps track of all propertice of a path. <a href="#">PropChannel</a>
STRUCT	structure of a channel. <a href="#">PropProfile</a>
STRUCT	Main structure of propagation profile <a href="#">PropData</a>
	Main structure of propagation data.

## PROPAGATION

STRUCT	<a href="#">PropTxInfo</a>
	This structure is used for fields related to channel layer information that need to be sent with a message.
STRUCT	<a href="#">PropRxInfo</a>
	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	<a href="#">PROP_NumberChannels(node)</a>  Get the number of channel.
MACRO	<a href="#">PROP_ChannelWavelength(node, channelIndex)</a>  Get wavelength of channel having index channelIndex
void	<a href="#">PROP_GlobalInit</a> (PartitionData* partitionData, NodeInput* nodeInput)  Initialization function for propagation This function is called from each partition, not from each node
void	<a href="#">PROP_PartitionInit</a> (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	<a href="#">PROP_Init</a> (Node* node, int channelIndex, NodeInput* nodeInput)  Initialization function for propagation functions. This function is called from each node.
void	<a href="#">PROP_ProcessEvent</a> (Node* node, Message* msg)  To receive message.
void	<a href="#">PROP_Finalize</a> (Node* node)  To collect various result.
double	<a href="#">PROP_PathlossFreeSpace</a> (double distance, double wavelength)

## PROPAGATION

	Calculates pathloss using free space model.
double	<a href="#">PROP_PathlossTwoRay</a> (double distance, double wavelength, float txAntennaHeight, float rxAntennaHeight)  To calculate path loss of a channel.
double	<a href="#">PROP_PathlossOpar</a> (double distance, double OverlappingDistance, double frequency, ObstructionType obstructiontype)  Calculates extra path attenuation using opar model.
void	<a href="#">PROP_CalculatePathloss</a> (Node* node, NodeId txNodeId, NodeId rxNodeId, int channelIndex, double wavelength, float txAntennaHeight, float rxAntennaHeight, PropPathProfile* pathProfile, bool forBinning)  To calculate path loss of a channel.
void	<a href="#">PROP_CalculateFading</a> (PropTxInfo* propTxInfo, Node* node2, int channelIndex, clocktype currentTime, float* fading_dB, double* channelReal, double* channelImag)  To calculate fading between two node.
BOOL	<a href="#">PROP_CalculateRxPowerAndPropagationDelay</a> (Message* msg, int channelIndex, PropChannel* propChannel, PropTxInfo* propTxInfo, Node* txNode, Node* rxNode, PropPathProfile* pathProfile)  This function will be called by QualNet wireless propagation code to calculate rxPower and prop delay for a specific signal from a specific tx node to a specific rx node.
BOOL	<a href="#">PROP_CalculateRxPowerAndPropagationDelay</a> (Message* msg, int channelIndex, PropChannel* propChannel, PropTxInfo* propTxInfo, Node* txNode, Node* rxNode, PropPathProfile* pathProfile)  This function will be called by QualNet wireless propagation code to calculate rxPower and prop delay for a specific signal from a specific tx node to a specific rx node.
void	<a href="#">PROP_MotionObtainfadingStretchingFactor</a> (PropTxInfo* propTxInfo, Node* receiver, int channelIndex)  Get a stretching factor for fast moving objects.
void	<a href="#">PROP_UpdatePathProfiles</a> (Node* node)  UpdatePathProfiles
void	<a href="#">PROP_ReleaseSignal</a> (Node* node, Message* msg, int phyIndex, int channelIndex, float txPower_dBm, clocktype duration, clocktype delayUntilAirborne)  Release (transmit) the signal
void	<a href="#">PROP_SubscribeChannel</a> (Node* node, int phyIndex, int channelIndex)

## PROPAGATION

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

	Set channel wavelength from profile for PropChannel.
double	<a href="#">PROP_SetChannelDopplerFrequency</a> (Node* node, int channelIndex)
void	Get channel doppler freq from profile for PropChannel. <a href="#">PROP_SetChannelDopplerFrequency</a> (Node* node, int channelIndex, double channelDopplerFrequency)
BOOL	Set channel doppler freq from profile for PropChannel. <a href="#">PROP_FrequencyOverlap</a> (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
<b>PROP_NumberChannels(node)</b>	Get the number of channel.
<b>PROP_ChannelWavelength(node, channelIndex)</b>	Get wavelength of channel having index channelIndex
<b>PROP_GlobalInit</b>  Initialization function for propagation This function is called from each partition, not from each node	void <b>PROP_GlobalInit</b> (PartitionData* partitionData, NodeInput* nodeInput)  Parameters: <ul style="list-style-type: none"><li>• partitionData - structure shared among nodes</li><li>• nodeInput - structure containing contents of input file</li></ul> Returns: <ul style="list-style-type: none"><li>• void - None</li></ul>
<b>PROP_PartitionInit</b>	void <b>PROP_PartitionInit</b> (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><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>partitionData - structure shared among nodes</li> <li>nodeInput - structure containing contents of input file</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>void - None</li> </ul>
<b>PROP_Init</b>	<p>Initialization function for propagation functions. This function is called from each node.</p> <p><b>void PROP_Init (Node* node, int channelIndex, NodeInput* nodeInput)</b></p> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>node - node being initialized.</li> <li>channelIndex - channel being initialized.</li> <li>nodeInput - structure containing contents of input file</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>void - None</li> </ul>
<b>PROP_ProcessEvent</b>	<p>To receive message.</p> <p><b>void PROP_ProcessEvent (Node* node, Message* msg)</b></p> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>node - Node that is</li> <li>msg - message received by the layer</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>void - None</li> </ul>
<b>PROP_Finalize</b>	<p>To collect various result.</p> <p><b>void PROP_Finalize (Node* node)</b></p> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>node - node for which results are to be collected</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>void - None</li> </ul>
<b>PROP_PathlossFreeSpace</b>	<p>Calculates pathloss using free space model.</p> <p><b>double PROP_PathlossFreeSpace (double distance, double wavelength)</b></p> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>distance - distance (meters) between two nodes</li> <li>wavelength - wavelength used for propagation.</li> </ul> <p><b>Returns:</b></p>

	<ul style="list-style-type: none"> <li>• <code>double</code> - pathloss in db</li> </ul>
<b>PROP_PathlossTwoRay</b>	<p>double <b>PROP_PathlossTwoRay</b> (double distance, double wavelength, float txAntennaHeight, float rxAntennaHeight)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>distance</code> - distance (meters) between two nodes</li> <li>• <code>wavelength</code> - wavelength used for propagation.</li> <li>• <code>txAntennaHeight</code> - transmitting antenna hight.</li> <li>• <code>rxAntennaHeight</code> - receiving antenna hight.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>double</code> - pathloss in db</li> </ul>
<b>PROP_PathlossOpar</b>	<p>double <b>PROP_PathlossOpar</b> (double distance, double OverlappingDistance, double frequency, ObstructionType obstructiontype)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>distance</code> - distance (meters) between two nodes</li> <li>• <code>OverlappingDistance</code> - overlapping distance</li> <li>• <code>frequency</code> - frequency used for propagation.</li> <li>• <code>obstructiontype</code> - obstruction type</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>double</code> - extra path attenuation in db</li> </ul>
<b>PROP_CalculatePathloss</b>	<p>void <b>PROP_CalculatePathloss</b> (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"> <li>• <code>node</code> - Node that is</li> <li>• <code>txNodeId</code> - including for debugging</li> <li>• <code>rxNodeId</code> - including for debugging</li> <li>• <code>channelIndex</code> - channel number.</li> <li>• <code>wavelength</code> - wavelength used for propagation.</li> <li>• <code>txAntennaHeight</code> - transmitting antenna hight.</li> <li>• <code>rxAntennaHeight</code> - receiving antenna hight.</li> </ul>

	<ul style="list-style-type: none"> <li>• <code>pathProfile</code> - characteristics of path.</li> <li>• <code>forBinning</code> - disables some features to support</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>PROP_CalculateFading</b>	<p>To calculate fading between two node.</p> <p><code>void PROP_CalculateFading (PropTxInfo* propTxInfo, Node* node2, int channelIndex, clocktype currentTime, float* fading_dB, double* channelReal, double* channelImag)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>propTxInfo</code> - Information about the transmitter</li> <li>• <code>node2</code> - receiver</li> <li>• <code>channelIndex</code> - channel number</li> <li>• <code>currentTime</code> - current simulation time</li> <li>• <code>fading_dB</code> - calculated fading store here.</li> <li>• <code>channelReal</code> - for cooperative comm</li> <li>• <code>channelImag</code> - for cooperative comm</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>PROP_CalculateRxPowerAndPropagationDelay</b>	<p>This function will be called by QualNet wireless propagation code to calculate rxPower and prop delay for a specific signal from a specific tx node to a specific rx node.</p> <p><code>BOOL PROP_CalculateRxPowerAndPropagationDelay (Message* msg, int channelIndex, PropChannel* propChannel, PropTxInfo* propTxInfo, Node* txNode, Node* rxNode, PropPathProfile* pathProfile)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>msg</code> - Signal to be propagated</li> <li>• <code>channelIndex</code> - Channel that the signal is propagated</li> <li>• <code>propChannel</code> - Info of the propagation channel</li> <li>• <code>propTxInfo</code> - Transmission parameers of the tx node</li> <li>• <code>txNode</code> - Point to the Tx node</li> <li>• <code>rxNode</code> - Point to the Rx node</li> <li>• <code>pathProfile</code> - For returning results</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>BOOL</code> - If FALSE, indicate the two nodes cannot comm TRUE means two nodes can communicate</li> </ul>
<b>PROP_CalculateRxPowerAndPropagationDelay</b>	<code>BOOL PROP_CalculateRxPowerAndPropagationDelay (Message* msg, int channelIndex, PropChannel* propChannel,</code>

	<p>This function will be called by QualNet wireless propagation code to calculate rxPower and prop delay for a specific signal from a specific tx node to a specific rx node.</p> <p>PropTxInfo* propTxInfo, Node* txNode, Node* rxNode, PropPathProfile* pathProfile)</p> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• msg - Signal to be propagated</li> <li>• channelIndex - Channel that the signal is propagated</li> <li>• propChannel - Info of the propagation channel</li> <li>• propTxInfo - Transmission parameters of the tx node</li> <li>• txNode - Point to the Tx node</li> <li>• rxNode - Point to the Rx node</li> <li>• pathProfile - For returning results</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>• BOOL - If FALSE, indicate the two nodes cannot comm TRUE means two nodes can communicate</li> </ul>
<b>PROP_MotionObtainfadingStretchingFactor</b>  Get a stretching factor for fast moving objects.	void <b>PROP_MotionObtainfadingStretchingFactor</b> (PropTxInfo* propTxInfo, Node* receiver, int channelIndex) <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• propTxInfo - Transmitter information</li> <li>• receiver - Receiver node.</li> <li>• channelIndex - channel number</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>PROP_UpdatePathProfiles</b>  UpdatePathProfiles	void <b>PROP_UpdatePathProfiles</b> (Node* node) <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• node - Node that is</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>PROP_ReleaseSignal</b>  Release (transmit) the signal	void <b>PROP_ReleaseSignal</b> (Node* node, Message* msg, int phyIndex, int channelIndex, float txPower_dBm, clocktype duration, clocktype delayUntilAirborne) <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• node - Node that is</li> <li>• msg - Signal to be transmitted</li> </ul>

	<ul style="list-style-type: none"> <li>• <code>phyIndex</code> - PHY data index</li> <li>• <code>channelIndex</code> - channel index</li> <li>• <code>txPower_dBm</code> - transmitting power</li> <li>• <code>duration</code> - transmission duration</li> <li>• <code>delayUntilAirborne</code> - delay until airborne</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>PROP_SubscribeChannel</b>	<p><code>void PROP_SubscribeChannel (Node* node, int phyIndex, int channelIndex)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Node that is</li> <li>• <code>phyIndex</code> - interface index</li> <li>• <code>channelIndex</code> - channel index</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>PROP_UnsubscribeChannel</b>	<p><code>void PROP_UnsubscribeChannel (Node* node, int phyIndex, int channelIndex)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Node that is</li> <li>• <code>phyIndex</code> - interface index</li> <li>• <code>channelIndex</code> - channel index</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>PROP_UnreferenceSignal</b>	<p><code>void PROP_UnreferenceSignal (Node* node, Message* msg)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - Node that is</li> <li>• <code>msg</code> - Signal to be unreferenced</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>PROP_CalculateInterNodePathLossOnChannel</b>	<code>void PROP_CalculateInterNodePathLossOnChannel (Node* node, int channelIndex, int* numNodesOnChannel,</code>

	<p>Calculate inter-node pathloss, distance values between all the nodes on a given channel</p> <p><b>NodeAddress* nodeIdList, float** pathloss_dB, float** distance)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - any valid node</li> <li>• channelIndex - selected channel instance</li> <li>• numNodesOnChannel - number of nodes using this channel</li> <li>• nodeIdList - list of (numNodesOnChannel) nodeIds</li> <li>• pathloss_dB - 2D pathloss array for nodes in</li> <li>• distance - 2D array of inter-node distances</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>PROP_CalculatePropagationDelay</b>	<p>Calculate the wireless propagation delay for the given distance and propagation speed.</p> <p><b>clocktype PROP_CalculatePropagationDelay (double distance, double propSpeed, PartitionData* partitionData, int channelIndex, int coordinateSystemType, Coordinates* fromPosition, Coordinates* toPosition)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• distance - Propagation distance</li> <li>• propSpeed - Propagation speed</li> <li>• partitionData - Partition data</li> <li>• channelIndex - Channel index or -1 for p2p</li> <li>• coordinateSystemType - Coordinate system type</li> <li>• fromPosition - Source position</li> <li>• toPosition - Destination position</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• 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.</li> </ul>
<b>PROP_Reset</b>	<p>Reset previous channel remove/add node to propChannel for signal delivery, in propagation_private.</p> <p><b>void PROP_Reset (Node* node, int phyIndex, char* newChannelListenable)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Node that is being instantiated in</li> <li>• phyIndex - interface index</li> <li>• newChannelListenable - new channel</li> </ul>

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

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

Check if there is frequency overlap between signal and receiver node.

Parameters:

- `txNode` - the Tx node
- `rxNode` - the Rx node
- `txChannelIndex` - the Tx channel index
- `rxChannelIndex` - the Rx channel index
- `txPhyIndex` - the PHY index for the Tx node.
- `rxPhyIndex` - the PHY index for the Rx node.

Returns:

- `BOOL` - if there is frequency overlap



Contact [EXata Support](#) for questions pertaining to the EXata API Reference. This document is confidential and proprietary.

It may not be reproduced or distributed without the expressed written consent of [SCALABLE Network Technologies](#).

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

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



# EXata 5.1 API Reference

## QUEUES

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

### Constant / Data Structure Summary

Type	Name
CONSTANT	<a href="#">DEQUEUE_NEXT_PACKET</a>
	Denotes position of packet in the queue for dequeue operation
CONSTANT	<a href="#">ALL_PRIORITIES</a>
	This macro is used to specify that queue and scheduler operations not consider priority value of queue or packet
CONSTANT	<a href="#">QOS_DEFAULT_INTERFACE_OBSERVATION_INTERVAL</a>
	This macro is used to specify the interface observation interval for Qos Routing. Ref.(Qospf.h see <a href="#">QOSPF_DEFAULT_INTERFACE_OBSERVATION_INTERVAL</a> )
CONSTANT	<a href="#">STATISTICS_RESOLUTION</a>
	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	<a href="#">DEFAULT_QUEUE_DELAY_WEIGHT_FACTOR</a>
	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	<a href="#">PACKET_ARRAY_INFO_FIELD_SIZE</a>
	The Queue structure will store a field of data in addition to the Message itself, with a maximum size of this value
ENUMERATION	<a href="#">QueueBehavior</a>

	This enumeration is used by both queues and schedulers to determine the queue behavior.
ENUMERATION	<a href="#">QueueOperation</a>
STRUCT	This enumeration is used by both queues and schedulers to determine the operation of the retrieve functions.
STRUCT	<a href="#">PacketArrayEntry</a>  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.  <a href="#">QueueAgeInfo</a>  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 Summary**

Return Type	Summary
void	<a href="#">Queue</a> (Message* msg, const void* infoField, BOOL* QueueIsFull, const clocktype currentTime, const double serviceTag)  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.
BOOL	<a href="#">Queue</a> (Message** msg, const int index, const QueueOperation operation, const clocktype currentTime, double* serviceTag)  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.
BOOL	<a href="#">Queue</a> ( )  This function prototype returns a Boolean value of true if the array of stored messages is empty, false otherwise.
int	<a href="#">Queue</a> ( )  This function prototype returns the number of bytes stored in the array.
int	<a href="#">Queue</a> ( )  This function prototype returns free space in number of bytes in the queue.
int	<a href="#">Queue</a> ( )

		This function prototype returns the number of Messages stored in the packetArray.
int	<a href="#">Queue()</a>	
void	<a href="#">Queue(double serviceTag)</a>	This function prototype returns the size of the Queue
int	<a href="#">Queue(Queue* oldQueue)</a>	Set the service tag of the queue
void		This function is proposed to replicate the state of the queue, as if it had been the operative queue all along. If there are packets in the existing queue, they are transferred one-by-one into the new queue. This can result in additional drops of packets that had previously been stored. This function returns the number of additional drops.
void	<a href="#">Queue(BOOL suspend)</a>	
void	<a href="#">Queue(int* qDelayVal, int* totalTransmissionVal, const clocktype currentTime, BOOL isResetTotalTransmissionVal)</a>	This function is proposed to identify and tag misbehaved queue at the interface, so that they can be punished.
int	<a href="#">Queue()</a>	
clocktype	<a href="#">Queue()</a>	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.
clocktype	<a href="#">Queue()</a>	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.
void	<a href="#">Queue(clocktype currentTime)</a>	
clocktype	<a href="#">Queue()</a>	This function prototype resets the current period statistics variables, and sets the currentPeriodStartTime to the currentTime.

	This function prototype returns the currentPeriodStartTime.  void <b>Queue</b> (Node* node, const char* layer, const int interfaceIndex, const int instanceId, const char* invokingProtocol, const char* splStatStr)
void	This function prototype outputs the final statistics for this queue. The layer, protocol, interfaceAddress, and instanceId parameters are given to IO_PrintStat with each of the queue's statistics.  <b>Queue</b> (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
<b>Queue</b>  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 <b>Queue</b> (Message* msg, const void* infoField, BOOL* QueueIsFull, const clocktype currentTime, const double serviceTag)  Parameters: <ul style="list-style-type: none"><li>• msg - Pointer to Message structure</li><li>• infoField - The infoField parameter</li><li>• QueueIsFull - returns Queue occupancy status</li><li>• currentTime - Current Simulation time</li><li>• serviceTag - ServiceTag</li></ul> Returns: <ul style="list-style-type: none"><li>• void - Null</li></ul>

<b>Queue</b>	<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><b>BOOL Queue</b> (Message** msg, const int index, const QueueOperation operation, const clocktype currentTime, double* serviceTag)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>msg</code> - The retrieved msg</li> <li>• <code>index</code> - The position of the packet in the queue</li> <li>• <code>operation</code> - The retrieval mode</li> <li>• <code>currentTime</code> - Current Simulation time</li> <li>• <code>serviceTag</code> - ServiceTag = NULL</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>BOOL</code> - TRUE or FALSE</li> </ul>
<b>Queue</b>	<p>This function prototype returns a Boolean value of true if the array of stored messages is empty, false otherwise.</p>	<p><b>BOOL Queue ()</b></p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>BOOL</code> - TRUE or FALSE</li> </ul>
<b>Queue</b>	<p>This function prototype returns the number of bytes stored in the array.</p>	<p><b>int Queue ()</b></p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>int</code> - Integer</li> </ul>
<b>Queue</b>	<p>This function prototype returns free space in number of bytes in the queue.</p>	<p><b>int Queue ()</b></p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>int</code> - number of bytes free.</li> </ul>
<b>Queue</b>	<p>This function prototype returns the number of Messages stored in the packetArray.</p>	<p><b>int Queue ()</b></p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>int</code> - Integer</li> </ul>
<b>Queue</b>		<p><b>int Queue ()</b></p> <p>Parameters:</p>

This function prototype returns the size of the Queue	Returns: <ul style="list-style-type: none"><li>• int - Integer</li></ul>
<b>Queue</b>  Set the service tag of the queue	void <b>Queue</b> (double serviceTag)  Parameters: <ul style="list-style-type: none"><li>• serviceTag - the value of the service tag</li></ul> Returns: <ul style="list-style-type: none"><li>• void - NULL</li></ul>
<b>Queue</b>  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.	int <b>Queue</b> (Queue* oldQueue)  Parameters: <ul style="list-style-type: none"><li>• oldQueue - Old queue pointer</li></ul> Returns: <ul style="list-style-type: none"><li>• int - Old packetArray</li></ul>
<b>Queue</b>  This function is proposed to identify and tag misbehaved queue at the interface, so that they can be punished.	void <b>Queue</b> (BOOL suspend)  Parameters: <ul style="list-style-type: none"><li>• suspend - The queue status</li></ul> Returns: <ul style="list-style-type: none"><li>• void - Null</li></ul>
<b>Queue</b>  This function is proposed for qos information update for Qos Routings like Qospf.	void <b>Queue</b> (int* qDelayVal, int* totalTransmissionVal, const clocktype currentTime, BOOL isResetTotalTransmissionVal)  Parameters: <ul style="list-style-type: none"><li>• qDelayVal - Returning qDelay value</li><li>• totalTransmissionVal - Returning totalTransmission value</li><li>• currentTime - Current simulation time</li><li>• isResetTotalTransmissionVal - Default false</li></ul> Returns: <ul style="list-style-type: none"><li>• void - Null</li></ul>
<b>Queue</b>	int <b>Queue</b> ()

	<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>	<p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> <li>• int - Integer</li> </ul>
<b>Queue</b>		<p>clocktype <b>Queue</b> ()</p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> <li>• clocktype - Utilize Time.</li> </ul>
<b>Queue</b>	<p>This function prototype returns the queue utilization, or the amount of time that the queue is nonempty, during a given period. This period starts at the beginning of the simulation, and restarts whenever the queue resetPeriod function is called.</p>	<p>clocktype <b>Queue</b> ()</p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> <li>• clocktype - Queue Delays.</li> </ul>
<b>Queue</b>	<p>This function prototype returns the average time a packet spends in the queue, during a given period. This period starts at the beginning of the simulation, and restarts whenever the QueueResetPeriodFunctionType function is called.</p>	<p>void <b>Queue</b> (clocktype currentTime)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• currentTime - Current simulation time.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - Null</li> </ul>
<b>Queue</b>	<p>This function prototype resets the current period statistics variables, and sets the currentPeriodStartTime to the currentTime.</p>	<p>clocktype <b>Queue</b> ()</p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> <li>• clocktype - Current period start time.</li> </ul>
<b>Queue</b>	<p>This function prototype outputs the final statistics for this queue. The layer, protocol, interfaceAddress, and instanceId parameters</p>	<p>void <b>Queue</b> (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"> <li>• node - Pointer to Node structure</li> </ul>

	<p>are given to IO_PrintStat with each of the queue's statistics.</p> <ul style="list-style-type: none"> <li>• <b>layer</b> - The layer string</li> <li>• <b>interfaceIndex</b> - The interface index</li> <li>• <b>instanceId</b> - Instance Ids</li> <li>• <b>invokingProtocol</b> - The protocol string</li> <li>• <b>splStatStr</b> - Special string for stat print</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <b>void</b> - Null</li> </ul>
<b>Queue</b> <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 configInfo. Some parameters includes default values, to prevent breaking existing models. [Uses: vide Pseudo code]</p>	<pre>void Queue (Node* node, const char queueTypeString[], const int queueSize, const int interfaceIndex, const int queueNumber, const int infoFieldSize, const BOOL enableQueueStat, const BOOL showQueueInGui, const clocktype currentTime, const void* configInfo)</pre> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <b>node</b> - Node pointer</li> <li>• <b>queueTypeString[]</b> - Queue type string</li> <li>• <b>queueSize</b> - Queue size in bytes</li> <li>• <b>interfaceIndex</b> - used to set random seed</li> <li>• <b>queueNumber</b> - used to set random seed</li> <li>• <b>infoFieldSize</b> - Default infoFieldSize = 0</li> <li>• <b>enableQueueStat</b> - Default enableQueueStat = false</li> <li>• <b>showQueueInGui</b> - If want to show this Queue in GUI</li> <li>• <b>currentTime</b> - Current simulation time</li> <li>• <b>configInfo</b> - pointer to a structure that contains</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <b>void</b> - Null</li> </ul>



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

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



# EXata 5.1 API Reference

## RANDOM NUMBERS

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

### Constant / Data Structure Summary

Type	Name
ENUMERATION	<a href="#">RandomDistributionType</a>  Random function types
ENUMERATION	<a href="#">RandomDataType</a>  Used for parsing input strings.
STRUCT	<a href="#">ValueProbabilityPair</a>  Stores one data point in a user defined distribution.
STRUCT	<a href="#">ArbitraryDistribution</a>  Stores a user defined distribution.

### Function / Macro Summary

Return Type	Summary
void	<a href="#">RANDOM_SetSeed</a> (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	<a href="#">RANDOM_urand</a> (RandomSeed seed)  Returns a uniform distribution in [0.0 .. 1.0]

## RANDOM NUMBERS

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

## RANDOM NUMBERS

void	<a href="#">RandomDistribution.setDistributionGeneralPareto</a> (T val1, T val2, double alpha)
	Sets the distribution to general Pareto distribution
void	<a href="#">RandomDistribution.setDistributionParetoUntruncated</a> (double alpha)
	Sets the distribution to the truncated Pareto distribution
void	<a href="#">RandomDistribution.setDistributionDeterministic</a> (T val)
	The distribution will always return val.
void	<a href="#">RandomDistribution.setDistributionNull</a> ()
	The distribution will return 0. This is used for initialization.
int	<a href="#">RandomDistribution.setDistribution</a> (char* inputString, char* printStr, RandomDataType dataType)
	Sets the distribution by parsing string input.
T	<a href="#">RandomDistribution.getRandomNumber</a> (RandomSeed seed, Node* node)
	These two functions return the next random number from the defined distribution.
void	<a href="#">RandomDistribution.setSeed</a> (UInt32 globalSeed, UInt32 nodeId, UInt32 protocolId, UInt32 instanceId)
	Calls RANDOM_SetSeed on the member seed.
void	<a href="#">RandomDistribution.setSeed</a> (RandomSeed seed)
	Copies the parameter seed into the member seed.

## Constant / Data Structure Detail

Random function types	
Enumeration	RandomDistributionType
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
<b>RANDOM_SetSeed</b>  Chooses from a set of pre-defined independent random seeds. The parameter names here are recommended invariants for use in selecting seeds, but other values could be used instead.	<p>void <b>RANDOM_SetSeed</b> (RandomSeed seed, UInt32 globalSeed, UInt32 nodeId, UInt32 protocolId, UInt32 instanceId)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>seed</code> - the seed to be set.</li> <li>• <code>globalSeed</code> - the scenario's global seed, i.e. SEED in the</li> <li>• <code>nodeId</code> - the node's ID</li> <li>• <code>protocolId</code> - the protocol number, as defined in the layer</li> <li>• <code>instanceId</code> - the instance of this protocol, often the</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>RANDOM_erand</b>  Returns a uniform distribution in [0.0 .. 1.0]	<p>double <b>RANDOM_erand</b> (RandomSeed seed)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>seed</code> - the seed for this random stream.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>double</code> - a random number</li> </ul>
<b>RANDOM_jrand</b>  Returns an integer uniformly distributed between -2^31 and 2^31.	<p>Int32 <b>RANDOM_jrand</b> (RandomSeed seed)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>seed</code> - the seed for this random stream.</li> </ul> <p>Returns:</p>

	<ul style="list-style-type: none"> <li>• <code>Int32</code> - a random number</li> </ul>
<b>RANDOM_nrand</b>	<p>Int32 <b>RANDOM_nrand</b> (RandomSeed seed)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>seed</code> - the seed for this random stream.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>Int32</code> - a random number</li> </ul>
<b>RANDOM_LoadUserDistributions</b>	<p>void <b>RANDOM_LoadUserDistributions</b> (NodeInput* nodeInput)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>nodeInput</code> - the .config file</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>RandomDistribution.init</b>	<p>void <b>RandomDistribution.init</b> ()</p> <p>Parameters:</p> <p>Initializes the random distribution</p> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>RandomDistribution.setDistributionUniform</b>	<p>void <b>RandomDistribution.setDistributionUniform</b> (T min, T max)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>min</code> - the low end of the range</li> <li>• <code>max</code> - the high end of the range</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>RandomDistribution.setDistributionUniformInteger</b>	<p>void <b>RandomDistribution.setDistributionUniformInteger</b> (T min, T max)</p> <p>Parameters:</p> <p>This one gives [min, max] for integers.</p> <ul style="list-style-type: none"> <li>• <code>min</code> - the low end of the range</li> <li>• <code>max</code> - the high end of the range</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>

<b>RandomDistribution.setDistributionExponential</b>	<p>void <b>RandomDistribution.setDistributionExponential</b> (T mean)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• mean - the mean value of the distribution</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>RandomDistribution.setDistributionGaussian</b>	<p>void <b>RandomDistribution.setDistributionGaussian</b> (T sigma)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• sigma - the sigma value</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>RandomDistribution.setDistributionGaussianInt</b>	<p>void <b>RandomDistribution.setDistributionGaussianInt</b> (T sigma)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• sigma - the sigma value</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>RandomDistribution.setDistributionPareto</b>	<p>void <b>RandomDistribution.setDistributionPareto</b> (T val1, T val2, double alpha)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• val1 - the low end of the range</li> <li>• val2 - the high end of the range</li> <li>• alpha - the alpha value</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>RandomDistribution.setDistributionPareto4</b>	<p>void <b>RandomDistribution.setDistributionPareto4</b> (T val1, T val2, T val3, double alpha)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• val1 - the minimum value of Pareto distribution</li> <li>• val2 - the low end of the range</li> <li>• val3 - the high end of the range</li> <li>• alpha - the alpha value</li> </ul>

	<p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>RandomDistribution.setDistributionGeneralPareto</b>	<p>void <b>RandomDistribution.setDistributionGeneralPareto</b> (T val1, T val2, double alpha)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• val1 - the low end of the range</li> <li>• val2 - the high end of the range</li> <li>• alpha - the alpha value</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>RandomDistribution.setDistributionParetoUntruncated</b>	<p>void <b>RandomDistribution.setDistributionParetoUntruncated</b> (double alpha)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• alpha - the alpha value</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>RandomDistribution.setDistributionDeterministic</b>	<p>void <b>RandomDistribution.setDistributionDeterministic</b> (T val)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• val - the value to return</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>RandomDistribution.setDistributionNull</b>	<p>void <b>RandomDistribution.setDistributionNull</b> ()</p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<b>RandomDistribution.setDistribution</b>	<p>int <b>RandomDistribution.setDistribution</b> (char* inputString, char* printStr, RandomDataType dataType)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• inputString - the input string, typically from a line</li> <li>• printStr - usually the name of the calling</li> </ul>

	<ul style="list-style-type: none"> <li>• <code>dataType</code> - the data type of the template class is</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>int</code> - returns the number of tokens read from the input string</li> </ul>
<b>RandomDistribution.getRandomNumber</b>	<p>T <b>RandomDistribution.getRandomNumber</b> (RandomSeed seed, Node* node)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>seed</code> - when the seed parameter is present, it is used in</li> <li>• <code>node</code> - the node parameter is required to look up user</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>T</code> - the random value</li> </ul>
<b>RandomDistribution.setSeed</b>	<p>void <b>RandomDistribution.setSeed</b> (UInt32 globalSeed, UInt32 nodeId, UInt32 protocolId, UInt32 instanceId)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>globalSeed</code> - the scenario's global seed, i.e. SEED in the</li> <li>• <code>nodeId</code> - the node's ID</li> <li>• <code>protocolId</code> - the protocol number, as defined in the layer</li> <li>• <code>instanceId</code> - the instance of this protocol, often the</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<b>RandomDistribution.setSeed</b>	<p>void <b>RandomDistribution.setSeed</b> (RandomSeed seed)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>seed</code> - an already initialized seed.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>



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

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



# EXata 5.1 API Reference

## SCEDULERS

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

### Constant / Data Structure Summary

Type	Name
CONSTANT	<a href="#">DEFAULT_QUEUE_COUNT</a>  Default number of queue per interface
STRUCT	<a href="#">QueueData</a>  This structure contains pointers to queue structures, default function behaviors, and statistics for the scheduler

### Function / Macro Summary

Return Type	Summary
QueueData*	<a href="#">Scheduler</a> (int priority)  Returns pointer to QueueData associated with the queue. this is a Private
int	<a href="#">Scheduler</a> ( )  Returns number of queues under this Scheduler
int	<a href="#">Scheduler</a> (int queueIndex)  Returns Priority for the queues under this Scheduler
BOOL	<a href="#">Scheduler</a> (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

## SCHEDULERS

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

## Constant / Data Structure Detail

Constant	DEFAULT_QUEUE_COUNT 3  Default number of queue per interface
Structure	QueueData

**Function / Macro Detail**

Function / Macro	Format
<b>Scheduler</b>  Returns pointer to QueueData associated with the queue. this is a Private	<p><b>QueueData* Scheduler (int priority)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <b>priority</b> - Queue priority</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <b>QueueData*</b> - Pointer of queue</li> </ul>
<b>Scheduler</b>  Returns number of queues under this Scheduler	<p><b>int Scheduler ()</b></p> <p>Parameters:</p> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <b>int</b> - Number of queue.</li> </ul>
<b>Scheduler</b>  Returns Priority for the queues under this Scheduler	<p><b>int Scheduler (int queueIndex)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <b>queueIndex</b> - Queue index</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <b>int</b> - Return priority of a queue</li> </ul>
<b>Scheduler</b>  Returns a Boolean value of TRUE if the array of stored messages in each queue that the scheduler controls are empty, and FALSE otherwise	<p><b>BOOL Scheduler (const int priority)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <b>priority</b> - Priority of a queue</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <b>BOOL</b> - TRUE or FALSE</li> </ul>
<b>Scheduler</b>  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><b>BOOL Scheduler (const int priority)</b></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <b>priority</b> - Priority of a queue</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <b>BOOL</b></li> </ul>

- TRUE or FALSE	
<b>Scheduler</b>  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>int <b>Scheduler</b> (const int priority)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• priority - Priority of a queue</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• int - Bytes in queue is used.</li> </ul>
<b>Scheduler</b>  This function enable Qos monitoring for all queues that the scheduler controls.	<p>void <b>Scheduler</b> (int queueIndex, int* qDelayVal, int* totalTransmissionVal, const clocktype currentTime, BOOL isResetTotalTransmissionVal)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• queueIndex - Queue index</li> <li>• qDelayVal - Queue delay</li> <li>• totalTransmissionVal - Transmission value</li> <li>• currentTime - Current simulation time</li> <li>• isResetTotalTransmissionVal - Total Transmission is set or not</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - Null</li> </ul>
<b>Scheduler</b>  This function enable data collection for performance study of schedulers.	<p>void <b>Scheduler</b> (int priority, int packetSize, const clocktype currentTime)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• priority - Priority of the queue</li> <li>• packetSize - Size of packet</li> <li>• currentTime - Current simulation time</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - Null</li> </ul>
<b>Scheduler</b>  This function invokes queue finalization.	<p>void <b>Scheduler</b> (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"> <li>• node - Pointer to Node structure</li> <li>• layer - The layer string</li> <li>• interfaceIndex - Interface Index</li> </ul>

	<ul style="list-style-type: none"> <li>• <code>instanceId</code> - Instance Ids</li> <li>• <code>invokingProtocol</code> - The protocol string</li> <li>• <code>splStatStr</code> - Special string for stat print</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - Null</li> </ul>
<b>SCHEDULER_Setup</b>	<p>This function runs the generic and then algorithm-specific scheduler initialization routine.</p> <p>void <b>SCHEDULER_Setup</b> (Scheduler** scheduler, const char schedulerTypeString[], BOOL enableSchedulerStat, const char* graphDataStr)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>scheduler</code> - Pointer of pointer to Scheduler class</li> <li>• <code>schedulerTypeString[]</code> - Scheduler Type string</li> <li>• <code>enableSchedulerStat</code> - Scheduler Statistics is set YES or NO</li> <li>• <code>graphDataStr</code> - Scheduler's graph statistics is set or not</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - Null</li> </ul>
<b>GenericPacketClassifier</b>	<p>Classify a packet for a specific queue</p> <p>int <b>GenericPacketClassifier</b> (Scheduler* scheduler, int pktPriority)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>scheduler</code> - Pointer to a Scheduler class.</li> <li>• <code>pktPriority</code> - Incoming packet's priority</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>int</code> - Integer.</li> </ul>



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

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





# EXata 5.1 API Reference

## SLIDING-WINDOW

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

### Constant / Data Structure Summary

Type	Name
STRUCT	<a href="#"><u>MsTmWin</u></a>  sliding time window averager structure

### Function / Macro Summary

Return Type	Summary
void	<a href="#"><u>MsTmWinInit</u></a> (MsTmWin* pWin, clocktype sSize, int nSlot, double weight, clocktype theTime)  initialize time sliding window with the given parameters
void	<a href="#"><u>MsTmWinInit</u></a> (MsTmWin* pWin, clocktype sSize, int nSlot, double weight, clocktype theTime)  resets time sliding window with the given parameters
void	<a href="#"><u>MsTmWinNewData</u></a> (MsTmWin* pWin, double data, clocktype theTime)  updates time sliding window with the given new data
clocktype	<a href="#"><u>MsTmWinWinSize</u></a> (MsTmWin* pWin, clocktype theTime)  returns the window size
double	<a href="#"><u>MsTmWinSum</u></a> (MsTmWin* pWin, clocktype theTime)  computes the data sum of the window
double	<a href="#"><u>MsTmWinAvg</u></a> (MsTmWin* pWin, clocktype theTime)

	computes the data average of the window <a href="#">MsTmWinTotalAvg</a> (MsTmWin* pWin, clocktype theTime)
double	computes the total data sum <a href="#">MsTmWinTotalSum</a> (MsTmWin* pWin, clocktype theTime)

**Constant / Data Structure Detail**

Structure	MsTmWin
	sliding time window averager structure

**Function / Macro Detail**

Function / Macro	Format
<b>MsTmWinInit</b>  initialize time sliding window with the given parameters	void <b>MsTmWinInit</b> (MsTmWin* pWin, clocktype sSize, int nSlot, double weight, clocktype theTime)  Parameters: <ul style="list-style-type: none"><li>• pWin - pointer to the time sliding window</li><li>• sSize - sliding window slot size</li><li>• nSlot - sliding window number of slots</li><li>• weight - weight for average computation</li><li>• theTime - the current time</li></ul> Returns: <ul style="list-style-type: none"><li>• void - None</li></ul>
<b>MsTmWinInit</b>  resets time sliding window with the given	void <b>MsTmWinInit</b> (MsTmWin* pWin, clocktype sSize, int nSlot, double weight, clocktype theTime)  Parameters: <ul style="list-style-type: none"><li>• pWin - pointer to the time sliding window</li></ul>

<p>parameters</p>	<ul style="list-style-type: none"> <li>• <code>sSize</code> - sliding window slot size</li> <li>• <code>nSlot</code> - sliding window number of slots</li> <li>• <code>weight</code> - weight for average computation</li> <li>• <code>theTime</code> - the current time</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<p><b>MsTmWinNewData</b></p> <p>updates time sliding window with the given new data</p>	<p>void <b>MsTmWinNewData</b> (<code>MsTmWin* pWin</code>, <code>double data</code>, <code>clocktype theTime</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>pWin</code> - pointer to the time sliding window</li> <li>• <code>data</code> - new data</li> <li>• <code>theTime</code> - the current time</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - None</li> </ul>
<p><b>MsTmWinWinSize</b></p> <p>returns the window size</p>	<p><code>clocktype MsTmWinWinSize</code> (<code>MsTmWin* pWin</code>, <code>clocktype theTime</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>pWin</code> - pointer to the time sliding window</li> <li>• <code>theTime</code> - the current time</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>clocktype</code> - the window size based on the current time</li> </ul>
<p><b>MsTmWinSum</b></p> <p>computes the data sum of the window</p>	<p><code>double MsTmWinSum</code> (<code>MsTmWin* pWin</code>, <code>clocktype theTime</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>pWin</code> - pointer to the time sliding window</li> <li>• <code>theTime</code> - the current time</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>double</code> - the data sum of the window</li> </ul>
<p><b>MsTmWinAvg</b></p>	<p><code>double MsTmWinAvg</code> (<code>MsTmWin* pWin</code>, <code>clocktype theTime</code>)</p> <p>Parameters:</p>

<p>computes the data average of the window</p>	<ul style="list-style-type: none"> <li>• <code>pWin</code> - pointer to the time sliding window</li> <li>• <code>theTime</code> - the current time</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>double</code> - the data average of the window</li> </ul>
<p><b>MsTmWinTotalSum</b></p> <p>computes the total data sum</p>	<p><code>double MsTmWinTotalSum (MsTmWin* pWin, clocktype theTime)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>pWin</code> - pointer to the time sliding window</li> <li>• <code>theTime</code> - the current time</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>double</code> - the total data sum</li> </ul>
<p><b>MsTmWinTotalAvg</b></p> <p>computes the total data average</p>	<p><code>double MsTmWinTotalAvg (MsTmWin* pWin, clocktype theTime)</code></p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>pWin</code> - pointer to the time sliding window</li> <li>• <code>theTime</code> - the current time</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>double</code> - the total data average</li> </ul>



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

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

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



# EXata 5.1 API Reference

## TRACE

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

### Constant / Data Structure Summary

Type	Name
CONSTANT	<a href="#">MAX_TRACE_LENGTH</a>  Buffer for an XML trace record.
CONSTANT	<a href="#">TRACE_STRING_LENGTH</a>  Generic maximum length of a string. The maximum length of any line in the input file is 3x this value.
ENUMERATION	<a href="#">TraceDirectionType</a>  Different direction of packet tracing
ENUMERATION	<a href="#">PacketActionType</a>  Different types of action on packet
ENUMERATION	<a href="#">PacketDirection</a>  Direction of packet with respect to the node
ENUMERATION	<a href="#">TraceLayerType</a>  Keeps track of which layer is being traced.
ENUMERATION	<a href="#">TraceIncludedHeadersType</a>  Specifies if included headers are output.
ENUMERATION	<a href="#">PacketActionCommentType</a>

	Gives specific comments on the packet action here packet drop.
ENUMERATION	<a href="#">TraceProtocolType</a>
STRUCT	Enlisting all the possible traces <a href="#">TraceData</a>
STRUCT	Keeps track of which protocol is being traced. <a href="#">PktQueue</a>
STRUCT	Gives details of the packet queue <a href="#">ActionData</a>
	Keeps track of protocol action

## Function / Macro Summary

Return Type	Summary
void	<a href="#">TRACE_Initialize</a> (Node* node, const NodeInput* nodeInput)  Initialize necessary trace information before simulation starts.
BOOL	<a href="#">TRACE_IsTraceAll</a> (Node* node)  Determine if TRACE-ALL is enabled from configuration file.
void	<a href="#">TRACE_PrintTrace</a> (Node* node, Message* message, TraceLayerType layerType, PacketDirection pktDirection, ActionData* actionData)  Print trace information to file. To be used with Tracer.
void	<a href="#">TRACE_PrintTrace</a> (Node* node, Message* message, TraceLayerType layerType, PacketDirection pktDirection, ActionData* actionData, NetworkType netType)  Print trace information to file. To be used with Tracer.
void	<a href="#">TRACE_EnableTraceXML</a> (Node* node, TraceProtocolType protocol, char* protocolName, TracePrintXMLFn xmlPrintFn, BOOL writeMap)  Enable XML trace for a particular protocol.
void	<a href="#">TRACE_EnableTraceXML</a> (Node* node, TraceProtocolType protocol, char* protocolName, TracePrintXMLFn xmlPrintFn,

	<code>BOOL writeMap)</code>
void	Enable XML trace for a particular protocol. <a href="#"><code>TRACE_DisableTraceXML</code></a> (Node* node, TraceProtocolType protocol, char* protocolName, BOOL writeMap)
void	Disable XML trace for a particular protocol. <a href="#"><code>TRACE_WriteToBufferXML</code></a> (Node* node, char* buf)
void	Write trace information to a buffer, which will then be printed to a file. <a href="#"><code>TRACE_WriteTraceHeader</code></a> (FILE* fp)
void	Write trace header information to the partition's trace file <a href="#"><code>TRACE_WriteXMLTraceTail</code></a> (FILE* fp)
	Write trace tail information to the partition's trace file

**Constant / Data Structure Detail**

Constant	<code>MAX_TRACE_LENGTH</code> (4090)  Buffer for an XML trace record.
Constant	<code>TRACE_STRING_LENGTH</code> 400  Generic maximum length of a string. The maximum length of any line in the input file is 3x this value.
Enumeration	<code>TraceDirectionType</code>  Different direction of packet tracing
Enumeration	<code>PacketActionType</code>  Different types of action on packet
Enumeration	<code>PacketDirection</code>

	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
<b>TRACE_Initialize</b>  Initialize necessary trace information before simulation starts.	void <b>TRACE_Initialize</b> (Node* node, const NodeInput* nodeInput)  Parameters: <ul style="list-style-type: none"><li>• node - this node</li><li>• nodeInput - access to configuration file</li></ul>

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

	<p>Enable XML trace for a particular protocol.</p> <p><b>TracePrintXMLFn</b> <code>xmlPrintFn</code>, <code>BOOL writeMap</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - this node</li> <li>• <code>protocol</code> - protocol to enable trace for</li> <li>• <code>protocolName</code> - name of protocol</li> <li>• <code>xmlPrintFn</code> - callback function</li> <li>• <code>writeMap</code> - flag to print protocol ID map</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - NULL</li> </ul>
<b>TRACE_EnableTraceXML</b>	<p>Enable XML trace for a particular protocol.</p> <p><b>void TRACE_EnableTraceXML</b> (<code>Node* node</code>, <code>TraceProtocolType protocol</code>, <code>char* protocolName</code>, <code>TracePrintXMLFn xmlPrintFn</code>, <code>BOOL writeMap</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - this node</li> <li>• <code>protocol</code> - protocol to enable trace for</li> <li>• <code>protocolName</code> - name of protocol</li> <li>• <code>xmlPrintFn</code> - callback function</li> <li>• <code>writeMap</code> - flag to print protocol ID map</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - NULL</li> </ul>
<b>TRACE_DisableTraceXML</b>	<p>Disable XML trace for a particular protocol.</p> <p><b>void TRACE_DisableTraceXML</b> (<code>Node* node</code>, <code>TraceProtocolType protocol</code>, <code>char* protocolName</code>, <code>BOOL writeMap</code>)</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>node</code> - this node</li> <li>• <code>protocol</code> - protocol to enable trace for</li> <li>• <code>protocolName</code> - name of protocol</li> <li>• <code>writeMap</code> - flag to print protocol ID map</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• <code>void</code> - NULL</li> </ul>
<b>TRACE_WriteToBufferXML</b>	<p><b>void TRACE_WriteToBufferXML</b> (<code>Node* node</code>, <code>char* buf</code>)</p>

	<p>Write trace information to a buffer, which will then be printed to a file.</p> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• node - This node.</li> <li>• buf - Content to print to trace file.</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>• void - NULL</li> </ul>
<b>TRACE_WriteTraceHeader</b>  <p>Write trace header information to the partition's trace file</p>	<p><b>void TRACE_WriteTraceHeader (FILE* fp)</b></p> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• fp - pointer to the trace file.</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>• void - NULL</li> </ul>
<b>TRACE_WriteXMLTraceTail</b>  <p>Write trace tail information to the partition's trace file</p>	<p><b>void TRACE_WriteXMLTraceTail (FILE* fp)</b></p> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• fp - pointer to the trace file.</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>• void - NULL</li> </ul>



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

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

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



# EXata 5.1 API Reference

## TRANSPORT LAYER

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

### Constant / Data Structure Summary

Type	Name
CONSTANT	<a href="#">TRANSPORT_DELAY</a>  Delay to process a packet in transport layer
ENUMERATION	<a href="#">TransportProtocol</a>  Enlisting different transport layer protocol
STRUCT	<a href="#">TransportData</a>  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 [EXata Support](#) for questions pertaining to the EXata API Reference. This document is confidential and proprietary.

It may not be reproduced or distributed without the expressed written consent of  
[SCALABLE Network Technologies](#).



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

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



# EXata 5.1 API Reference

## USER

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

### Constant / Data Structure Summary

Type	Name
CONSTANT	<a href="#">USER_PHONE_STARTUP_DELAY</a>  Delay from a cellphone is powered on until it can start working.
CONSTANT	<a href="#">USER_INCREASE_DISSATISFACTION</a>  The step value that the user dissatisfaction degree is increased each time.
CONSTANT	<a href="#">USER_CDECREASE_DISSATISFACTION</a>  The step value that the user dissatisfaction degree is decreased each time.
CONSTANT	<a href="#">USER_DEFAULT_STATUS_START_TIME</a>  Defines the default user status start time
ENUMERATION	<a href="#">UserApplicationStatus</a>  Status of an user application session.
STRUCT	<a href="#">UserAppInfo</a>  Data structure stores information of one user application session.
STRUCT	<a href="#">UserStatus</a>  Data structure stores statuses of a user
STRUCT	<a href="#">struct_user_str</a>

**Function / Macro Summary**

<b>Return Type</b>	<b>Summary</b>
void	<p><a href="#"><u>USER_HandleCallUpdate</u></a>(Node* node, UserApplicationStatus appStatus)</p> <p>Reaction to the status change of an application session</p>
void	<p><a href="#"><u>USER_HandleUserLayerEvent</u></a>(Node* node, Message* msg)</p> <p>Handle messages and events for user layer</p>
void	<p><a href="#"><u>USER_SetTrafficPattern</u></a>(Node* node)</p> <p>Set a user's traffic pattern based on its profile.</p>
void	<p><a href="#"><u>USER_SetApplicationArrival</u></a>(Node* node)</p> <p>Schedule an application arrival time.</p>

**Constant / Data Structure Detail**

Constant	<p>USER_PHONE_STARTUP_DELAY 5S</p> <p>Delay from a cellphone is powered on until it can start working.</p>
Constant	<p>USER_INCREASE_DISSATISFACTION 0.1</p> <p>The step value that the user dissatisfaction degree is increased each time.</p>
Constant	<p>USER_CECREASE_DISSATISFACTION -0.1</p> <p>The step value that the user dissatisfaction degree is decreased each time.</p>
Constant	<p>USER_DEFAULT_STATUS_START_TIME 10S</p>

	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
<b>USER_HandleCallUpdate</b>  Reaction to the status change of an application session	void <b>USER_HandleCallUpdate</b> (Node* node, UserApplicationStatus appStatus)  Parameters: <ul style="list-style-type: none"><li>• node - Pointer to node.</li><li>• appStatus - New status of the app session</li></ul> Returns: <ul style="list-style-type: none"><li>• void - NULL</li></ul>
<b>USER_HandleUserLayerEvent</b>  Handle messages and events for user layer	void <b>USER_HandleUserLayerEvent</b> (Node* node, Message* msg)  Parameters: <ul style="list-style-type: none"><li>• node - Pointer to node.</li><li>• msg - The event</li></ul> Returns: <ul style="list-style-type: none"><li>• void - NULL</li></ul>
<b>USER_SetTrafficPattern</b>	void <b>USER_SetTrafficPattern</b> (Node* node)

Set a user's traffic pattern based on its profile.	<p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Pointer to node.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - NULL</li> </ul>
<b>USER_SetApplicationArrival</b>  Schedule an application arrival time.	<b>void USER_SetApplicationArrival (Node* node)</b>  <p>Parameters:</p> <ul style="list-style-type: none"> <li>• node - Pointer to node.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• void - NULL</li> </ul>



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

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

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



# EXata 5.1 API Reference

## WALLCLOCK

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

### Function / Macro Summary

Return Type	Summary
BOOL	<p><a href="#">WallClock( void)</a></p> <p>This method returns true if the WallClock is currently in the paused state.</p>
double	<p><a href="#">WallClock( )</a></p> <p>Return the real time multiple</p>
void	<p><a href="#">WallClock( void)</a></p> <p>Pausing of the WallClock can be disabled by any external interface ambassador. Permission to pause is all or nothing, so if any external interface disables pause, no pausing is allowed. As an example, a simulation using IPNE and HLA is run. If the IPNE code disables pausing, then HLA won't be able to pause the WallClock or in other words the wall clock's value for time just keeps running.</p>
void	<p><a href="#">WallClock( void)</a></p> <p>Allows pausing of the WallClock</p>
double	<p><a href="#">WallClock( void)</a></p> <p>Get the amount of time, in seconds, spent paused.</p>

### Function / Macro Detail

Function / Macro	Format
<b>WallClock</b>	BOOL <b>WallClock</b> ( void)

	<p>This method returns true if the WallClock is currently in the paused state.</p> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• void - None</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>• BOOL - TRUE or FALSE</li> </ul>
<b>WallClock</b>	<p><b>double WallClock ()</b></p> <p><b>Parameters:</b></p> <p>Return the real time multiple</p> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>• double - None</li> </ul>
<p><b>WallClock</b></p> <p>Pausing of the WallClock can be disabled by any external interface ambassador. Permission to pause is all or nothing, so if any external interface disables pause, no pausing is allowed. As an example, a simulation using IPNE and HLA is run. If the IPNE code disables pausing, then HLA won't be able to pause the WallClock or in other words the wall clock's value for time just keeps running.</p>	<p><b>void WallClock ( void)</b></p> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• void - None</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<p><b>WallClock</b></p> <p>Allows pausing of the WallClock</p>	<p><b>void WallClock ( void)</b></p> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• void - None</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>• void - None</li> </ul>
<p><b>WallClock</b></p> <p>Get the amount of time, in seconds, spent paused.</p>	<p><b>double WallClock ( void)</b></p> <p><b>Parameters:</b></p> <ul style="list-style-type: none"> <li>• void - None</li> </ul> <p><b>Returns:</b></p> <ul style="list-style-type: none"> <li>• double - The amount of time paused, in seconds.</li> </ul>



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

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

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