



IEEE 1588 Protocol Stack for Microsoft Windows®

Product Documentation

Real-Time Systems GmbH
Gartenstrasse 33
88212 Ravensburg
Germany

www.real-time-systems.com

info@real-time-systems.com

Document: 000612238

Table of Contents

- 1 Notices.....4**
- 2 Introduction.....5**
 - 2.1 Important Facts.....5
 - 2.1.1 Ready to go Versions:.....5
 - 2.1.2 Portability:.....5
 - 2.1.3 Interoperability:.....5
 - 2.2 Definition.....6
 - 2.3 Product Features.....6
 - 2.4 Background.....6
 - 2.5 Benefits.....7
 - 2.6 About Real-Time Systems GmbH.....8
 - 2.7 Technical Support.....8
- 3 Technical overview.....8**
 - 3.1 How it works.....9
 - 3.1.1 Basic operation.....9
 - 3.1.2 Message types.....9
 - 3.1.3 Example transaction.....10
 - 3.1.4 Boundary clocks.....11
 - 3.1.5 Literature.....11
- 4 Overview of the RTS IEEE 1588 Windows Version.....12**
- 5 Installation of the RTS IEEE 1588 Windows Version.....12**
 - 5.1 32-Bit Windows.....12
 - 5.2 64-Bit Windows.....14
- 6 Uninstall of the RTS IEEE 1588 Windows Version.....15**
- 7 Installer Requirements.....15**
- 8 Use of the RTS IEEE 1588 Windows Version.....15**
 - 8.1 Settings.....17
 - 8.1.1 Protocol Version.....17
 - 8.1.2 Slave Only.....17
 - 8.1.3 Master Selection.....17
 - 8.1.4 Stratum.....18
 - 8.1.5 Clock Class, Priority 1, Priority 2.....18
 - 8.1.6 Announce Interval.....18
 - 8.1.7 Synch Interval.....18
 - 8.1.8 Number of Switches.....18
 - 8.1.9 Domain.....18
 - 8.1.10 Synchronize Windows Time to PTP Time.....18
 - 8.1.11 International Atomic Time Correction.....18
 - 8.1.12 IPV6.....18
 - 8.1.13 DSCP.....18
 - 8.1.14 Verbosity Level.....19
 - 8.1.14.1 Verbose Output.....19
 - 8.2 Interface Selection.....23

8.3Open Log Directory.....	23
<u>9API Description of the RTS IEEE 1588 Windows Version.....</u>	<u>23</u>
9.1Time-Representation Data Structure.....	23
9.2Enumerations	24
9.3Service Control Functions.....	24
9.3.1ieee1588ServiceStop.....	24
9.3.2ieee1588ServiceStart.....	24
9.3.3ieee1588ServiceGetState.....	25
9.4PTP Specific Functions.....	25
9.4.1ieee1588GetAdapters.....	25
9.4.2ieee1588GetTime.....	25
9.4.3ieee1588GetTimeEx.....	25
9.4.4ieee1588GetGmMac.....	26
9.4.5ieee1588GetState.....	26
9.4.6ieee1588SetInboundLatency.....	26
9.4.7ieee1588SetOutboundLatency.....	26
9.4.8ieee1588SetInitialTime.....	27
9.5Callback Functions.....	27
9.5.1ieee1588SetTimerCallback.....	27
9.5.2ieee1588FreeTimerCallback.....	27
9.6GPIO Functions.....	28
9.6.1ieee1588ConfigureGpioPeriodicPulse.....	28
9.7Configuration Functions.....	28
9.7.1ieee1588ConfigReadParameter.....	28
9.7.2ieee1588ConfigWriteParameter.....	28
9.7.3ieee1588ConfigGetInterface.....	29
9.7.4ieee1588ConfigSetInterface.....	29
9.7.5ieee1588ConfigGetMasterMacAddress.....	29
9.7.6ieee1588ConfigSetMasterMacAddress.....	30
<u>10Utilities.....</u>	<u>30</u>
10.1PTPManager.....	30
10.2PTPv2Browser.....	32
10.3Wireshark.....	32
10.3.1Protocol related Wireshark Identifiers.....	33
<u>11END OF DOCUMENT.....</u>	<u>34</u>

1 Notices

All Rights Reserved: Neither this document nor excerpts therefrom may be reproduced, transmitted, or conveyed to third parties by any means whatsoever without the express permission of Real-Time Systems GmbH. At the time of publication, the information in this document was carefully verified and found to be correct. Nonetheless, it cannot be ruled out that discrepancies may arise in the course of time. This document will be reviewed at reasonable intervals to assure that subsequent editions reflect the current product status.

Trademarks: [Real-Time Systems GmbH](#) is a registered trademark of Real-Time Systems GmbH. All other product and company names herein may be trademarks of their respective owners. Wind River, VxWorks and Workbench are registered trademarks of Wind River Systems; Alameda, California; www.windriver.com. Intel and Pentium are registered trademarks of Intel Corporation; Santa Clara, California; www.intel.com. Windows, Microsoft Windows and MS Windows are registered trademarks of Microsoft Corporation; Redmond, Washington; www.microsoft.com.

Undocumented Features: While a complex product such as the one described herein may contain undocumented features, such features are not considered to be part of the product and their functionality is therefore not subject to any form of support or guarantee.

Information: Real-Time Systems' 1588 PTP protocol stack is based on PTPd. In accord with BSD-style licensing conditions, the following copyright notice - Copyright (c) 2005 Kendall Correll - applies to all program files that comprise PTPd, just as if the copyright text were included in each PTPd file.

2 Introduction

Real-Time Systems' IEEE 1588 Protocol Stack is a modular, software product that implements the full capabilities of the IEEE 1588 Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. This standard, often called Precision Time Protocol or PTP, fully automates the synchronization of multiple clocks in a distributed system. Units that participate in such a system must be linked together by Ethernet.

While there is no limitation to the kind of applications that could benefit from this highly-accurate clock synchronization technique, a few that naturally come to mind are: measurement, instrumentation, motion control and process automation.

This handbook presents enough information to give decision-makers some general insight into the formal methods used by the 1588 protocol. Developers, however, who may require more specific information, should consult the reference materials listed elsewhere in this manual. 3.1.5 Literature

In this manual, Real-Time Systems GmbH is often abbreviated as RTS

2.1 Important Facts

Even though this product is designed for the Microsoft Windows Operating Systems in conjunction with the Intel Gigabit Ethernet Controller Intel 82574L, there are different implementations available in binary format for product evaluation. This product has been used in a lot of different product environments with different operating systems and on different hardware platforms.

2.1.1 Ready to go Versions:

Real Time-Systems GmbH partners with a lot of companies on the embedded market therefor the stack has been ported to different hardware platforms and operating systems. The following table shows the ready to go compatibility list:

CPU	Timestamping	Operating System
Intel x86	Software Realtek NIC 8139	VxWorks
Intel x86	Intel E1000 (PCI-E) (82574L and I350)	Windows XP/7/Server
Intel x86	Intel E1000 (PCI-E) (82574L)	VxWorks
Intel x86	Intel E1000 (PCI-E) (82574L)	QNX Neutrino
I.MX35 (MSC EXM32)	National PHY DP83640	QNX Neutrino

2.1.2 Portability:

The stack has been used in several products where hardware platforms are project specific and not available for other customers. See below the list of ports done by RTS.

CPU	Timestamping	Operating System
PPC	Software (Board Specific NIC)	VxWorks
PPC	Freescale PowerQUICC	ThreadX
PPC	FPGA based	QNX Neutrino

2.1.3 Interoperability:

The RTS 1588 Software Stack adheres strictly to the IEEE 1588 Standard. Therefore, any number of nodes outfitted with the RTS 1588 Software Stack will work in harmony with other diverse network

nodes—such as, switches, routers and master clocks—provided all run in full accordance with the IEEE 1588 standard.

2.2 Definition

The IEEE 1588 Precision Time Protocol embodies a software technique to synchronize distributed real-time clocks in a packet-based multicast network (usually Ethernet).

2.3 Product Features

- Supports V1 and V2
- Full Master / Slave Implementation
- Small Footprint
- Hardware supported IEEE1588 via Intel 82574L, I350
- Highly accurate: +/- 3 Micro Seconds in the 82574L and I350 Chip
- Allows to synchronize Windows System Time at +/- 1 ms
- Access to Real-Time Clock through a rich API
- Full logging capabilities
- Intuitive Graphical User Interface for all Settings
- Easy Installation / Automated Setup
- IPV6 Support

2.4 Background

Many real-time and embedded systems use two or more physically independent, computer-based subsystems, or nodes, connected together in a local area network (LAN), to fulfill the system's overall purpose. To act in a harmonious fashion, that is, to perform interdependent work, the various nodes must accurately synchronize to one another.

Synchronization is a fundamental requirement for many kinds of distributed systems. To achieve it, three basic techniques may be used: message-based, periodic, or time-based synchronization techniques.

In message-based synchronization, a central node sends messages to other nodes, which, upon receiving a message, carry out a related directive. In periodic synchronization, a time-raster is broadcast throughout the system, causing nodes to carry out specific functions in accord with the raster-ticks. Time-based synchronization, the third and most modern method, causes a clock in each participating network node to enter into synchronicity with a master clock; each node then performs tasks in accord with its own local synchronized clock's time.

While the first two methods are especially susceptible to jitter or delay and have no relationship to absolute time, clock time-based systems overcome both objections. Until recently, however, most time-based solutions have been proprietary to a specific company or product.

Now, the Institute of Electrical and Electronics Engineers - IEEE has developed a standardized clock-synchronizing protocol that may be used in virtually all multicasting (packet-based) networks such as Ethernet. The IEEE 1588 standard defines a 'Precision Clock Synchronization Protocol for Network and Control Systems', also known as Precision Time Protocol or PTP for short.

This low-overhead protocol synchronizes multiple clocks in a distributed system either to one another or to a real-world clock such as GPS. While providing a high degree of interoperability, it has significant advantages over older methods.

Consider the following characteristics:

- Time: The time base is absolute time, the same as real-world time provided by the master.
- Components: Uses conventional, inexpensive multi-source components, integral to nearly all LANs.
- Independent: The protocol is defined to be independent of networking topology and technology, provided it is a multi-cast network like Ethernet.
- Fault tolerant: The protocol uses an algorithm to dynamically compensate for propagation delays caused by hubs, switches and repeaters.
- Self-configuring: An identical protocol in each unit routinely analyzes network topology and configures itself in an optimal fashion. This autonomous process eliminates the need for administrative services.
- Automatic: Since each node automatically exchanges PTP protocol messages with the others, participating nodes have to do no more than load and start the protocol.
- Hot-plugging: Units may at any time be added to or removed from a running system.
- Master time: No central time authority is required because the system nominates a master clock automatically. If a GPS clock or other high-accuracy clock is present in the system, it will be used as a grandmaster clock.
- Precision: The protocol provides a high degree of accuracy. (+/- 3 Micro Seconds on Intel 82574L and I350). Additionally the Windows System time may be synchronized to about 1ms accuracy.

2.5 Benefits

- Standards: Because Real-Time Systems' implementation completely conforms to the IEEE 1588 standard, any system that uses it automatically attains interoperability with other systems or devices that conform to the standard.
- Accuracy: The IEEE 1588 protocol provides a significant improvement in synchronization accuracy over other network time protocols.
- Administration-free: A network synchronized by the PTP protocol is so flexible that it will recover and function when powered-up or powered-down (in whole or in part), even when nodes are added or removed.
- Availability: Now.
- Savings / Investment protection: Real-Time Systems' PTP protocol stack is a vigorously maintained, standard product that deploys on virtually all standard host computers that use conventional Ethernet components.
- Determinism: Since the Ethernet and IP permit collisions, they are not conducive to deterministic operations in a distributed system. The PTP protocol, on the other hand, enforces determinism by freeing the real-time applications from network constraints. When all operations on distributed nodes are based on local, highly-synchronized clocks, the deterministic behavior of the overall system is assured.
- Low-cost hardware: To successfully synchronize the clocks in a network, Real-Time System's PTP implementation for Windows requires only that nodes be equipped with standard network interface cards (NIC) based on Intel 82574L and I350.

2.6 About Real-Time Systems GmbH

Real-Time Systems GmbH is a leading supplier of products and solutions for operating system virtualization and precision time synchronization in the embedded and real-time markets. Besides marketing its own products, Real-Time Systems also offers consulting and engineering services to their customers.

Real-Time Systems GmbH is privately held and maintains its corporate headquarters in Ravensburg, Germany with partners in Europe, USA and Asia.

Real-Time Systems GmbH -
the partner you've been looking for

Real-Time Systems GmbH
Gartenstrasse 33
88212 Ravensburg, Germany
E-mail inquiries: info@real-time-systems.com
Web: www.real-time-systems.com

2.7 Technical Support

Software updates, help of a knowledgeable software engineer via telephone or e-mail, and e-mail announcements regarding this and related products are available.

E-mail inquiries: ieee1588@real-time-systems.com

3 Technical overview

The 1588 PTP protocol, firmly grounded in topological mathematics and computer science, is too complex to present in this manual. For those interested in details, the references listed in Literature 3.1.5 provide a wealth of information. The information offered in this manual is only intended to give someone not already familiar with the protocol a reasonable notion of what it entails.

The following terms are used in dealing with PTP systems:

- **Network device:** Network or non-terminal device – such as: router, switch or repeater – that has more than one network (NIC) connection BUT no 1588 PTP capability.
- **Boundary clock:** A network device, usually a network switch or router with IEEE 1588 capability. In the PTP protocol, the boundary clock is a branching element, spanning two or more subnets. Boundary clocks propagate the best master clock time into subnets.
- **Master clock:** One clock in a subnet, automatically nominated by the best master clock algorithm, establishes the time standard to which all other clocks in that subnet are subordinated.
- **Best Master Clock (BMC):** The PTP protocol uses a complex algorithm to automatically determine which clock in a subnet will serve as the master clock. If two or more clocks appear to be equally good, the algorithm makes certain that all but one enter the slave state.
- **Slave clock:** Clocks in nodes that subordinate themselves to a master clock and synchronize with the master time. Note: In Real-Time Systems' implementation, users may optionally force nodes of their choice to always be 'slaves' via a parameter in the start function. Refer to `ieee1588Start`. If all nodes except one are forced to be slaves, the remaining clock is certain to be nominated as master clock.
- **Grandmaster clock:** The PTP protocol establishes a tree-like hierarchy of all clocks participating in a synchronized network. A single clock, determined to be at the root of this hierarchy, is called the grandmaster. If a GPS or other high-precision clock is present in the system, it will usually be nominated as grandmaster. If a grandmaster clock runs with UTC (Coordinated Universal Time), then all clocks in the network will be synchronized to UTC.
- **Ordinary clock:** An ordinary clock is a PTP clock in an end node that is connected to its subnet by a single network interface card (NIC).

- **Hardware clock:** A hardware clock such as GPS, radio receiver, or a high-frequency oscillator-based clock may be included in a PTP-managed system. Such a clock will usually be nominated as a grandmaster clock. If there is no time difference between hardware clocks – for example, when there is more than one GPS clock in a network – the PTP protocol may logically partition the network so that there are multiple hierarchical trees. The tree structure imposed by the PTP protocol is not dependent on the physical network structure; it logically overlays the physical network.
- **Software clocks:** Real-Time Systems' PTP protocol implements one software clock at each node. It is not required to have a hardware clock anywhere in the system. <<check>>
- **Overhead:** Approximately every 2 seconds (a parameterized value), nodes exchange PTP messages that resynchronize and correct subordinated clocks. At such times, a change in the master clock can also be automatically initiated.
- **Precision:** Delays and jitter are periodically measured. Data is collected by a routine exchange of messages which is then used to correct slave clocks, thus keeping all system clocks in very close harmony. As delivered, the Real-Time Systems' IEEE 1588 stack will synchronize the various clocks to an accuracy of ± 3 μ sec or better in the 82574L and I350 clock.
- **Spanning tree:** The PTP protocol establishes a logical hierarchy of clocks in a network. This logical tree structure may or may not agree with the actual physical layout of the network. If there are two or more GPS clocks in the network, the protocol may logically segment the network so that a GPS clock will be the grandmaster in each such segment. Such a network would be synchronized to UTC time.
- **Timestamp:** Users' application programs have access (via user-callable functions) to the software clock's timestamp function.
- **Jitter / Skew:** Due to signal propagation times and buffering, the more devices (e.g.: repeater, hub, switch) that lie on a path to an ordinary clock (end of branch) the greater will be both jitter and skew. The less often PTP messages are exchanged, the greater the skew among clocks is likely to be. The frequency of message exchange may be specified by the user. Other (non-protocol) network traffic can also increase skew.
- **Local networks:** It is not practical to operate the PTP protocol over the Internet. The protocol is limited to multicasting networks, such as Ethernet.
- **Anomalies:** In the RTS IEEE 1588 protocol, provision has been made to prevent isolated anomalies from adversely affecting the system's accuracy. If, for example, the PTP software discovers a synchronizing timestamp with an unreasonable value (70 μ sec beyond expectations), it will not use it in the time-correction algorithm.

3.1 How it works

3.1.1 Basic operation

While the 1588 PTP protocol is a complex state machine realized in software (refer to Chapter 3.1.5 Literature), the following list of basic tasks should give the reader a notion of what the protocol does. The protocol must...

1. Explore the system to establish boundaries and communications paths.
2. Nominate a master clock.
3. Build a master-slave hierarchy.
4. Start up in an orderly fashion, and, in a running system, reconfigure itself, as necessary.
5. Regularly broadcast master-clock data so that subordinated clocks can synchronize to and remain synchronized with their master clocks.
6. Provide application programs with access to PTP parameters, including timestamps.

3.1.2 Message types

Within a 1588 PTP protocol system, all communication is accomplished using just five message types.

- Sync Event message
- Follow_Up General message, including timing information

- Delay_Req Event message
- Delay_Resp General message, including timing information
- Management General message (provided access to PTP parameters)

Sync - Issued by a clock in the **Master** state, **Sync** messages contain clock characterization information and the estimated local time of transmission.

Follow Up - Issued by a clock in the **Master** state, **Follow Up** messages are logically associated with the preceding **Sync** message. They contain the precise time of transmission of the **Sync** message, which will be used for calculations in precision-alignment algorithms.

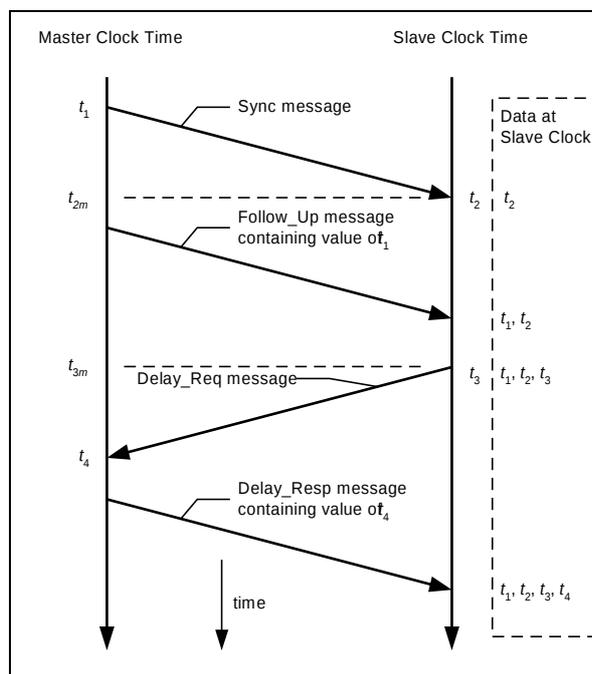
Delay Req - Issued by clocks in the **Slave** state, **Delay Req** messages contain the local time of transmission from the slave. The slave notes the exact time it sends this message; upon receipt, the master clock notes it as well.

Delay Resp - Issued by a clock in the **Master** state, **Delay Resp** messages are logically associated with the preceding **Delay Req** message. It contains the time that a **Delay Req** message was received at the master. The slave node records the precise time this message is received, and, in connection with the known time that it had sent the **Delay Req**, uses the difference to calculate the actual latency time.

Management - Management messages that are multicast remain within a subnet. Boundary clocks do, however, forward management messages into other subnets. Management messages convey configuration information to all nodes in a system.

3.1.3 Example transaction

The following account should give the reader an intuitive idea of how a typical data exchange between a master and a slave node is used to establish clock synchronicity. It is meant to give the reader a feeling for how the protocol works. This example transaction is for two-step clocks using the request-response delay mechanism.



- The master clock transmits a **Sync** message to the slave clocks. The **Sync** message does not contain time information. Upon sending the **Sync**, the master clock timestamps the outgoing packet and records this information (local timestamp t_1).
- When a slave clock receives the **Sync** message, it uses its local clock to generate timestamp (t_2). This timestamp is used to remember the **Sync** message's time of arrival.
- Afterwards the master clock packs timestamp t_1 of the prior sent **Sync** message in a **Follow Up** message and sends this to the slave clocks.

- Each slave clock, upon receiving the **Follow_Up** message, transmits a **Delay_Req** to the master clock. Upon sending the **Delay_Req**, the slave clock timestamps the outgoing packet and records this information (local timestamp t3).
- When the master clock receives the **Delay_Req** message, it uses its own local clock to generate timestamp (t4), which corresponds to the exact time it received the **Delay_Req** from the slave.
- Using a **Delay_Resp** message, the master clock then transmits timestamp t4 back to the slave clock.
- Finally, the slave then uses all four times, i.e. t1, t2, t3 and t4, to compute the offset between itself and the master clock. It uses differences, offsets and delays for calculations to synchronize itself to the master clock.

3.1.4 Boundary clocks

Repeaters, routers, hubs and switches are all network devices that forward messages to other network participants. They traditionally amplify, sort, redirect and sometimes buffer network packets.

A boundary clock is a network device that implements the 1588 functionality. In short: if subnets running the PTP protocol are connected to one another by a 1588-enabled device, that device is known as a boundary clock. Since boundary clocks span subnets, they have more than one network port.

A boundary clock dynamically determines the master-slave hierarchy in a subnet and, according to the Best Master Clock algorithm (BMC), determines which clock will be at the root of the PTP clock hierarchy. The clock at the root is called the grandmaster clock.

Caution: Although non-1588 network devices don't participate in the protocol, they may introduce delays in the propagation of data packets and can therefore impact the accuracy of PTP clock synchronization. It would be well, therefore, to consider this when planning the deployment of the PTP protocol.

While boundary clocks do not propagate **Sync**, **Follow_Up**, **Delay_Req**, or **Delay_Resp** messages from one subnet to another, they do, however, forward **Management** messages into other subnets.

Often, a boundary clock will be nominated as a master clock, but if there is a better clock in the system, a GPS clock, for example, the protocol will appoint it as a grandmaster clock.

Although each port of a boundary clock appears to be an ordinary node to a subnet, there is, in fact, only one clock implemented in the boundary device.

3.1.5 Literature

- John C. Eidson, Measurement, Control and Communication Using IEEE 1588, Springer-Verlag London, ISBN-10: 1846282500
- IEEE Standard 1588-2002, IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. (2002), The Institute of Electrical and Electronics Engineers, Inc., New York
- IEEE Standard 1588-2008, IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems
- [Web: National Institute of Standards and Technology, IEEE 1588 Website: National Institute of Standards and Technology - 1588](#)

4 Overview of the RTS IEEE 1588 Windows Version

The RTS IEEE1588-PTP product enables the usage of a Microsoft Windows PC to act as a PTP master or slave.

The product consists of the following parts:

- The Intel E1000 IEEE1588 Enabled driver for Intel CT Network Cards (igbDrv.sys).
- The IEEE1588 service which implements the Precision Time Protocol (ieee1588Service.exe).
- The IEEE1588 control service which provides GUI access to the PTP service (ieee1588ControlService.exe).
- The IEEE1588 GUI (running in the system tray) to configure all settings.
- A dynamic link library to provide access to the IEEE1588 service API (ieee1588API.dll).

5 Installation of the RTS IEEE 1588 Windows Version

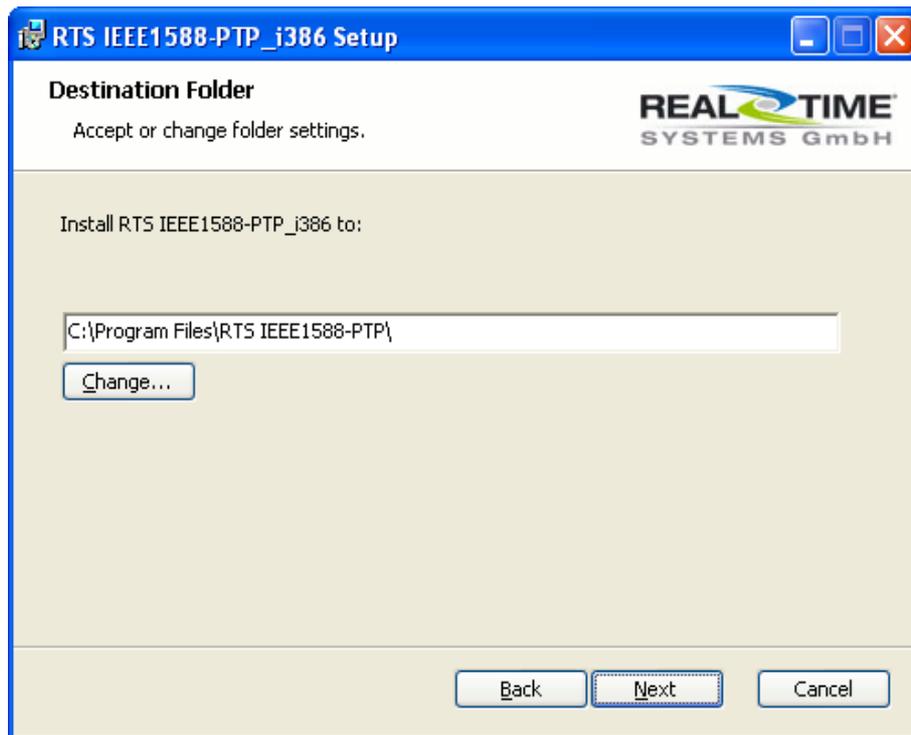
5.1 32-Bit Windows

1. Start the installation by double-clicking the ieee1588_32_x.x.x.msi file (x.x.x is the version number).

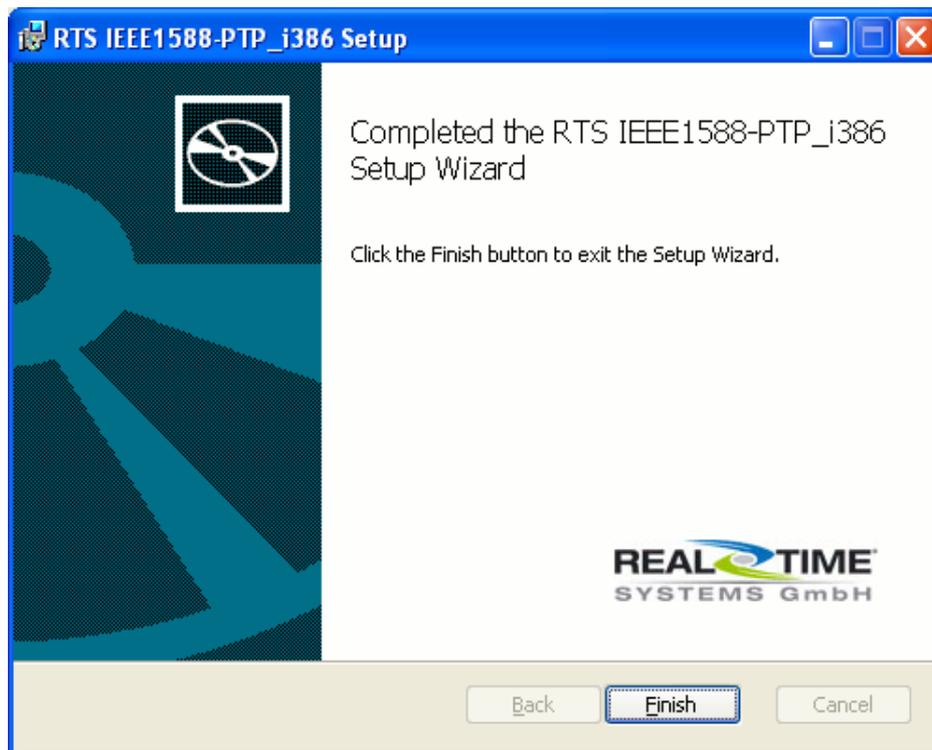
Note: Please do not install this package over the network as the network driver will be replaced / installed during the installation process.



2. Click „Next“. Read and accept the license of the RTS IEEE1588-PTP product and click „Next“.



3. Now you can change the destination folder. Click „Next“.
4. Click „Install“ to start the installation.



5. Finish the installation.

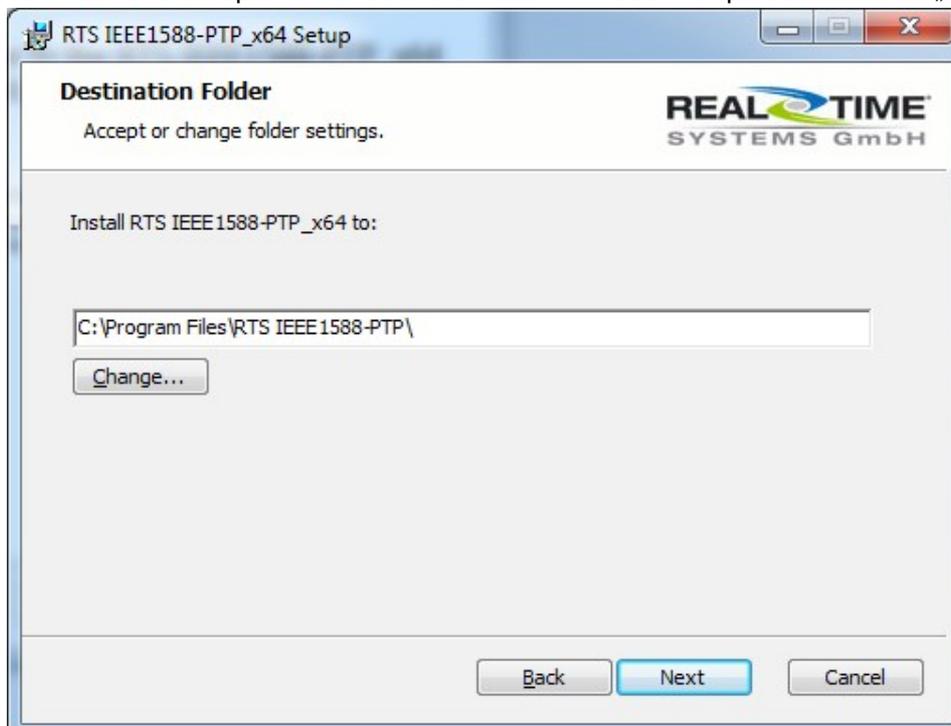
5.2 64-Bit Windows

1. Start the installation by double-clicking the `ieee1588_64_x.x.x.msi` file (x.x.x is the version number).

Note: Please do not install this package over the network as the network driver will be replaced / installed during the installation process.



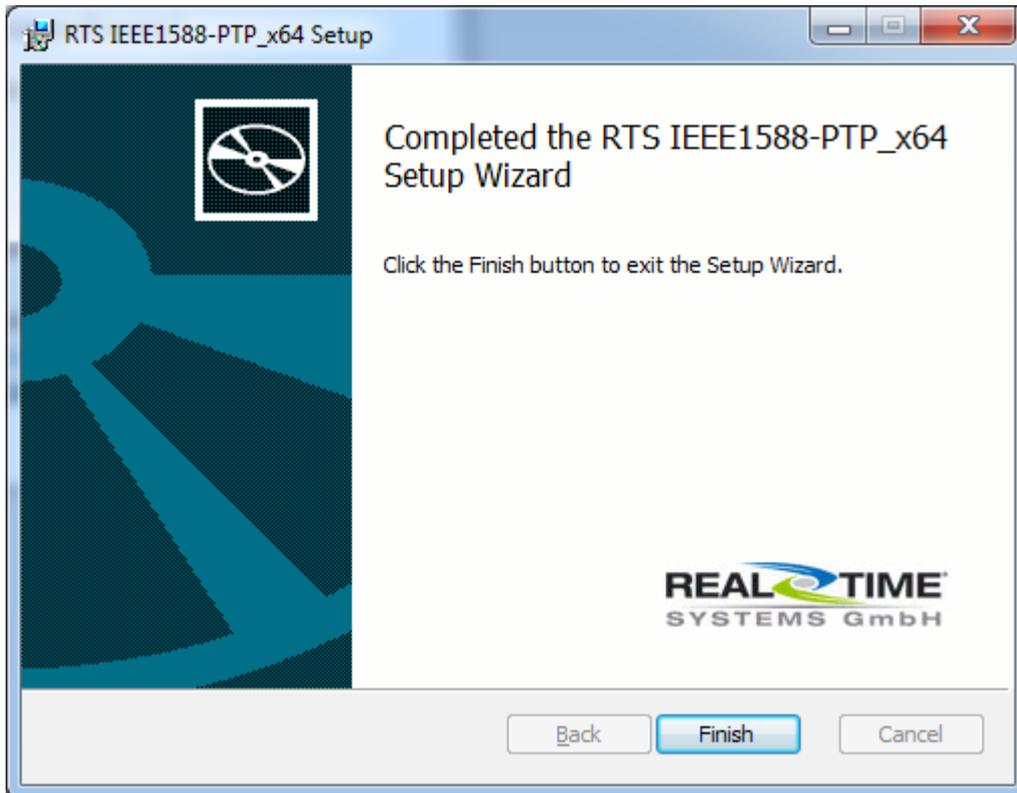
2. Click „Next“. Read and accept the license of the RTS IEEE1588-PTP product and click „Next“.



3. Now you can change the destination folder. Click „Next“.

4. Click „Install“ to start the installation.

5. During the installation you will be asked twice for confirmation (To grant administrative rights to the installer and to accept the driver installation).



6. Finish the installation.

6 Uninstall of the RTS IEEE 1588 Windows Version

Either go to Windows Control Panel and remove the product via the „Add or Remove Programs“ control panel applet or right-click the installation file (.msi file) and choose „Uninstall“.

NOTE: On 64-Bit systems a message box will ask you if you allow the uninstall of the driver.

7 Installer Requirements

1. It is allowed to have Windows Firewall to be configured like you need it. Anyway Windows Firewall Service must be started (Start->Control Panel->Administrative Tools->Services->Windows Firewall/Internet Connection Sharing(ICS)). If the Windows Firewall Service is not running during installation the Installer will throw two error messages by ignoring both of them the software will still install successful.
2. It is required you have a Intel Network card of following types plugged in.

VendorID	DeviceID
0x8086	0x10D3
0x8086	0x1521

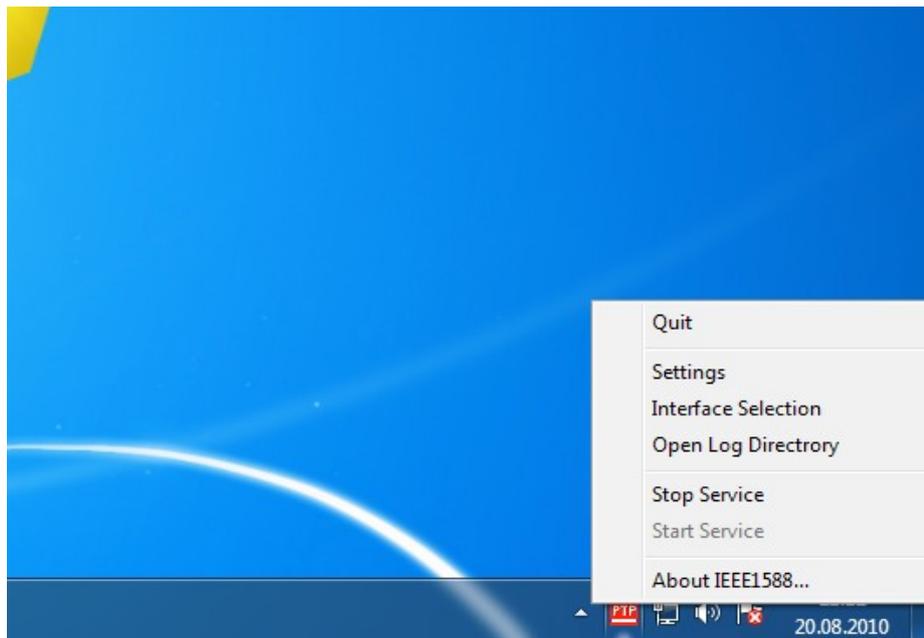
8 Use of the RTS IEEE 1588 Windows Version

The IEEE1588 service starts automatically at boot time and will start PTP tasks according to the settings. When a user logs on the GUI application will start and show an icon in the system tray. The icon is shown in different colors representing the current state of the PTP stack:

GRAY	Service uninitialized/not started
RED	Service is unsynchronized PTP slave
GREEN	Service is synchronized PTP slave
BLUE	Service is PTP master

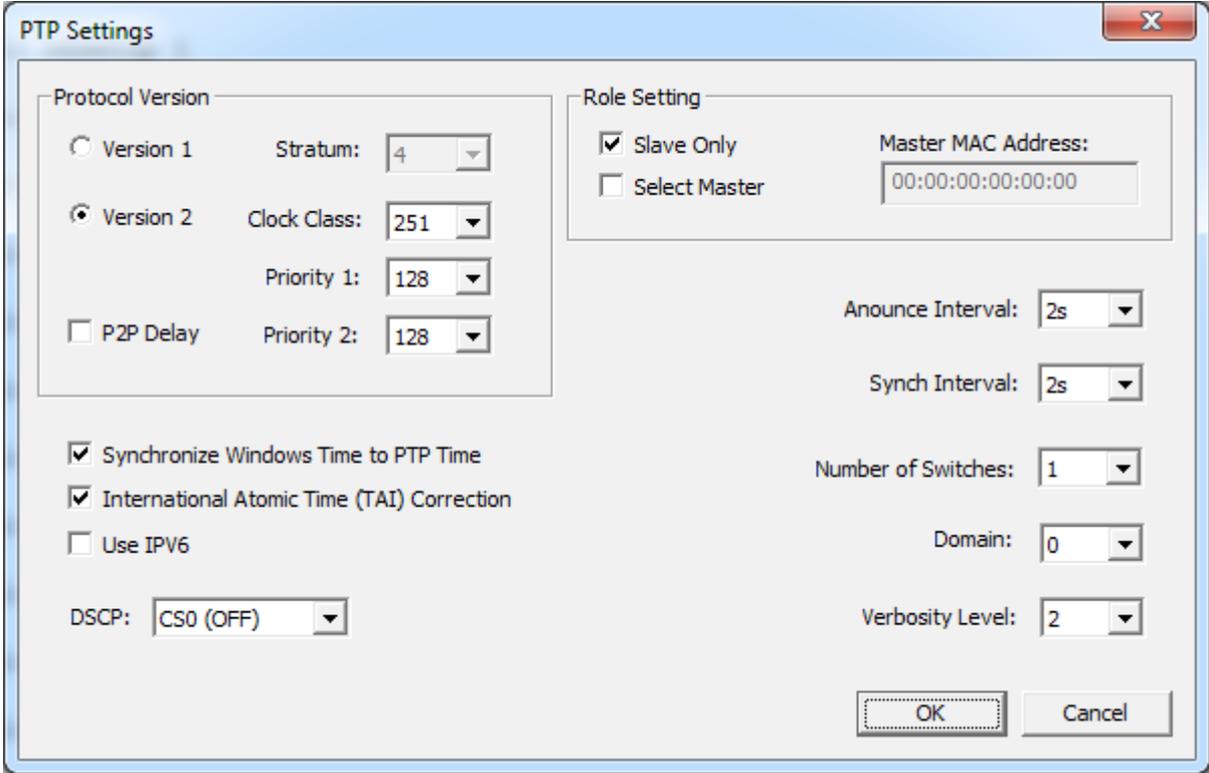
A right mouse-click opens a menu with the following options:

Quit	terminates the GUI application (PTP keeps running)
Settings	opens the settings dialog
Interface Selection	opens the interface selection dialog
Open Log Directory	opens the log file directory in Windows Explorer
Stop Service	stops the PTP service (grayed out if service is stopped)
Start Service	start the PTP service (grayed out if service is running)
About IEEE1588...	shows version information of the product



8.1 Settings

All settings are stored in the Windows Registry under HKEY_LOCAL_MACHINE\Software\Real-Time-Systems\ieee1588. Registry keys should not be changed manually. For configuration and Setup use the dialog of the GUI.



8.1.1 Protocol Version

This is an implementation specific parameter for the RTS IEEE1588 Stack. It allows to choose different protocol standards.

Value	Description
1	Stack sends and receives packets according to the IEEE Standard 1588-2002
2	Stack sends and receives packets according to the IEEE Standard 1588-2008

8.1.2 Slave Only

This is an implementation specific flag for the RTS IEEE1588 Stack. A node which sets that flag will never become master. To achieve this it automatically configures the protocol settings stratum for IEEE Standard 1588-2002 or clockClass priority1 and priority2 according to the IEEE Standard 1588-2008 specification. If this configuration field is checked, stratum is ignored.

8.1.3 Master Selection

To enable the master selection you need to specify the master's MAC Address. In a node where a specific master selection is active other masters won't be accepted as synchronization source. This might avoid problems on a topology change which might be caused by a master failure.

8.1.4 Stratum

The Stratum parameter describes the clock quality according to the specification table for protocol version 1. Default shall be 4 depending on the clock it might increase to 3. A smaller value will result in a node with a higher priority when calculating the Best Master Clock Algorithm.

8.1.5 Clock Class, Priority 1, Priority 2

8.1.6 Announce Interval

Announce Interval specifies the interval of announce messages according to IEEE Standard 1588-2008. It is not used for protocol version 1 (IEEE Standard 1588-2002).

8.1.7 Synch Interval

The Synch Interval specifies the interval synch messages are exchanged according to IEEE Standard 1588-2008 or IEEE Standard 1588-2002 (used for both protocol versions).

8.1.8 Number of Switches

This parameter is a RTS-IEEE1588 Stack specific parameter. Number of Network components (switches) between master and slave which don't support IEEE1588 Time Stamping within their implementation. This parameter is used to tune the filter algorithms to a specific network topology. Increasing this value will also result in a loss of synchronization precision.

For maximum precision it is recommended to use network components which support Time Stamping. In this case Number of Switches should be set to zero.

8.1.9 Domain

Domain parameter specifies the synchronisation domain in which your node is running. With this parameter you can have several different groups of synchronized nodes on the network. This parameter is specified as values from 0 to 3 the IEEE Standard 1588-2008. In IEEE Standard 1588-2002 this value is specified as strings _DFLT, _ALT0, _ALT1, _ALT2. The values from GUI are automatically translated into the selected protocol version.

8.1.10 Synchronize Windows Time to PTP Time

This parameter is an RTS-IEEE1588 Stack specific flag. It enables a synchronization loop between the IEEE1588 time domain and the local Windows time. If a node becomes master the IEEE1588 Time is initialized with its current Windows System Time.

8.1.11 International Atomic Time Correction

Some Masters may provide the correction value for International Atomic Time. This is distributed within IEEE Standard 1588-2008 Announce Messages or in Sync Messages for IEEE Standard 1588-2002. The message field is described as current UTC offset.

If the check box for International Atomic Time Correction option is set the current UTC offset is subtracted from current IEEE1588 Time. If the check box is selected all API calls provided will report the corrected time.

8.1.12 IPV6

If this flag is set IPV6 is used instead of IPV4. IPV6 protocol has to be installed on the system. IPV6 and IPV4 may not be used at the same time.

8.1.13 DSCP

Sets DSCP tag on outgoing frames.

8.1.14 Verbosity Level

This parameter lets you set the amount of details logged. Keep the verbosity level at 0 or 1 to only record the most important system messages. For trouble shooting and detailed system analysis, this value may be increased. Please note that with high verbosity levels, log files can grow quickly, consuming a lot of disk space and resulting in delays when opening or viewing log files.

8.1.14.1 Verbose Output

Find below a sample of the verbose output of a stack running in slave mode after Stack initialization.

```

servoLib: servoReInit
servoLib: updateOffset
servoLib: send time      7035s   988343024ns
servoLib:   rcv time      7035s   988343994ns
servoLib: ##master-to-slave delay:      0s      970ns
servoLib: ##last difference:            0s      -2685ns
servoLib: servoState:    3
servoLib: master-to-slave delay:        0s      970ns
servoLib: slave-to-master delay:        0s      0ns
servoLib: one-way delay:                 0s     1000ns
servoLib: offset from master:            0s     -268ns
servoLib: min offset from master:       99999999s  0ns
servoLib: max offset from master:       99999999s -1ns
servoLib: constDriftValue:              0s      0ns
servoLib: updateOffset
servoLib: send time      7038s   5241346ns
servoLib:   rcv time      7038s   5242517ns
servoLib: ##master-to-slave delay:      0s     1171ns
servoLib: ##last difference:            0s     970ns
servoLib: ##master-to-slave delay:      0s     1171ns
servoLib: ##last difference:            0s     970ns
servoLib: result:          0s     201ns
servoLib: servoState:    3
servoLib: master-to-slave delay:        0s     1171ns
servoLib: slave-to-master delay:        0s      0ns
servoLib: one-way delay:                 0s     1000ns
servoLib: offset from master:            0s     -268ns
servoLib: min offset from master:       99999999s  0ns
servoLib: max offset from master:       99999999s -1ns
servoLib: constDriftValue:              0s      0ns
servoLib: updateOffset
servoLib: send time      7040s   22159627ns
servoLib:   rcv time      7040s   22161001ns
servoLib: ##master-to-slave delay:      0s     1374ns
servoLib: ##last difference:            0s     1171ns
servoLib: ##master-to-slave delay:      0s     1374ns
servoLib: ##last difference:            0s     1171ns
servoLib: result:          0s     203ns
servoLib: servoState:    3
servoLib: master-to-slave delay:        0s     1374ns
servoLib: slave-to-master delay:        0s      0ns
servoLib: one-way delay:                 0s     1000ns
servoLib: offset from master:            0s     -268ns
servoLib: min offset from master:       99999999s  0ns
servoLib: max offset from master:       99999999s -1ns
servoLib: constDriftValue:              0s      0ns

```

```

servoLib: updateDelay
servoLib: updateOffset
servoLib: send time      7042s    38698069ns
servoLib:   recv time    7042s    38699644ns
servoLib: ##master-to-slave delay:      0s      1575ns
servoLib: ##last difference:            0s      1374ns
servoLib: ##master-to-slave delay:      0s      1575ns
servoLib: ##last difference:            0s      1374ns
servoLib: result:          0s      201ns
servoLib: servoState:     3
servoLib: master-to-slave delay:        0s      1575ns
servoLib: slave-to-master delay:        0s        0ns
servoLib: one-way delay:                0s     1000ns
servoLib: offset from master:           0s     -268ns
servoLib: min offset from master:       99999999s    0ns
servoLib: max offset from master:       99999999s   -1ns
servoLib: constDriftValue:              0s        0ns
servoLib: updateOffset
servoLib: send time      7044s    54190070ns
servoLib:   recv time    7044s    54191847ns
servoLib: ##master-to-slave delay:      0s      1777ns
servoLib: ##last difference:            0s      1575ns
servoLib: ##master-to-slave delay:      0s      1777ns
servoLib: ##last difference:            0s      1575ns
servoLib: result:          0s      202ns
array[0]:          0s      201ns
array[1]:          0s      203ns
array[2]:          0s      201ns
array[3]:          0s      202ns
constant:          0s      201ns
servoLib: updateClock
servoLib: corr:          0s      978ns
servoLib: const:        0s      201ns

```

It starts with a servoReInit that means all previously calculated values are reinitialized. It starts from scratch after servoReInit. The start servoStart 3 is the scratch state for each node. In this state the node collects data from the sync/follow_up message pairs provided by the master. With this information it calculates the drift between the clocks. The values of array[0] to array [3] show the 4 samples taken. With the const value is the average of those samples.

```

servoLib: servoState:     1
servoLib: master-to-slave delay:        0s      1777ns
servoLib: slave-to-master delay:        0s        0ns
servoLib: one-way delay:                0s     1000ns
servoLib: offset from master:           0s     777ns
servoLib: min offset from master:       99999999s    0ns
servoLib: max offset from master:       99999999s   -1ns
servoLib: constDriftValue:              0s      201ns
servoLib: updateOffset
servoLib: send time      7046s    71170992ns
servoLib:   recv time    7046s    71171774ns
servoLib: ##master-to-slave delay:      0s      782ns
servoLib: ##last difference:            0s     1777ns
servoLib: updateClock
servoLib: corr:          0s     -17ns
servoLib: const:        0s      201ns
servoLib: servoState:     1
servoLib: master-to-slave delay:        0s      782ns

```

```

servoLib: slave-to-master delay:      0s      0ns
servoLib: one-way delay:              0s      1000ns
servoLib: offset from master:         0s      -218ns
servoLib: min offset from master:     99999999s  0ns
servoLib: max offset from master:     99999999s  -1ns
servoLib: constDriftValue:            0s      201ns
servoLib: updateDelay
delay[0]:          0s      1000ns
delay[1]:          0s      1000ns
delay[2]:          0s      241ns
servoLib: updateOffset

```

After reaching servoState 1 the clock frequency difference is known and now the stack starts to reduce the offset from master and recalculate the packet delays on the network. To leave the servoState 1 to servoState 2 (which means fully synchronized) it must match following criteria:

1. multiple measured delay values must match the filter rules for path delay measurement
2. multiple measured values of offset from master must match the filter rules for sync detection

Please note that for both conditions the stack parameter layer2hops is used to modify the filter rules. Increasing layer2hops loosens the strength of the filter algorithm.

```

servoLib: send time      8312s  845515193ns
servoLib:   rcv time    8312s  845515576ns
servoLib: ##master-to-slave delay:      0s      383ns
servoLib: ##last difference:            0s      383ns
servoLib: ##master-to-slave delay:      0s      383ns
servoLib: ##last difference:            0s      383ns
servoLib: result:          0s      0ns
array[0]:          0s      0ns
array[1]:          0s      0ns
array[2]:          0s      0ns
array[3]:          0s      0ns
constant:          0s      0ns
servoLib: updateClock
servoLib: corr:          0s      -617ns
servoLib: const:         0s      0ns
servoLib: servoState:   1
servoLib: master-to-slave delay:        0s      383ns
servoLib: slave-to-master delay:        0s      0ns
servoLib: one-way delay:                0s      1000ns
servoLib: offset from master:           0s      -617ns
servoLib: min offset from master:       99999999s  0ns
servoLib: max offset from master:       99999999s  -1ns
servoLib: constDriftValue:              0s      0ns
servoLib: updateOffset
servoLib: send time      8314s  860127995ns
servoLib:   rcv time    8314s  860128776ns
servoLib: ##master-to-slave delay:      0s      781ns
servoLib: ##last difference:            0s      383ns
servoLib: updateClock
servoLib: corr:          0s      -219ns
servoLib: const:         0s      0ns
servoLib: servoState:   1
servoLib: master-to-slave delay:        0s      781ns
servoLib: slave-to-master delay:        0s      0ns
servoLib: one-way delay:                0s      1000ns
servoLib: offset from master:           0s      -219ns
servoLib: min offset from master:       99999999s  0ns

```

```

servoLib: max offset from master:    99999999s    -1ns
servoLib: constDriftValue:          0s           0ns
servoLib: updateDelay
delay[0]:          0s           1000ns
delay[1]:          0s           1000ns
delay[2]:          0s           232ns
servoLib: updateOffset
servoLib: send time      8316s    865310515ns
           rcv time      8316s    865311300ns
servoLib: ##master-to-slave delay:    0s           785ns
servoLib: ##last difference:          0s           383ns
servoLib: updateClock
servoLib: corr:          0s           41ns
servoLib: const:        0s           0ns
servoLib: servoState:   1
servoLib: master-to-slave delay:      0s           785ns
servoLib: slave-to-master delay:      0s           0ns
servoLib: one-way delay:              0s           744ns
servoLib: offset from master:         0s           41ns
servoLib: min offset from master:     99999999s    0ns
servoLib: max offset from master:     99999999s    -1ns
servoLib: constDriftValue:          0s           0ns
servoLib: updateOffset
servoLib: send time      8318s    881033117ns
           rcv time      8318s    881033901ns
servoLib: ##master-to-slave delay:    0s           784ns
servoLib: ##last difference:          0s           383ns
servoLib: updateClock
servoLib: corr:          0s           40ns
servoLib: const:        0s           0ns
servoLib: servoState:   1
servoLib: master-to-slave delay:      0s           784ns
servoLib: slave-to-master delay:      0s           0ns
servoLib: one-way delay:              0s           744ns
servoLib: offset from master:         0s           40ns
servoLib: min offset from master:     99999999s    0ns
servoLib: max offset from master:     99999999s    -1ns
servoLib: constDriftValue:          0s           0ns
servoLib: updateOffset
servoLib: send time      8320s    897304238ns
           rcv time      8320s    897305023ns
servoLib: ##master-to-slave delay:    0s           785ns
servoLib: ##last difference:          0s           383ns
servoLib: updateClock
servoLib: corr:          0s           41ns
servoLib: const:        0s           0ns
servoLib: servoState:   2
servoLib: master-to-slave delay:      0s           785ns
servoLib: slave-to-master delay:      0s           0ns
servoLib: one-way delay:              0s           744ns
servoLib: offset from master:         0s           41ns
servoLib: min offset from master:     99999999s    0ns
servoLib: max offset from master:     99999999s    -1ns
servoLib: constDriftValue:          0s           0ns
servoLib: updateDelay
delay[0]:          0s           1000ns
delay[1]:          0s           232ns
delay[2]:          0s           234ns

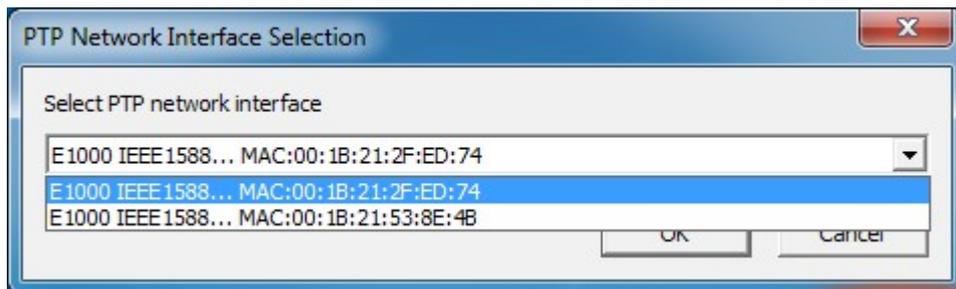
```

Important Note:

If the node starts up as master of the network it will just print servoRelnit. If the node will change into an error state (e.g. wrong sync interval) it will permanently repeat the servoRelnit.

8.2 Interface Selection

The interface selection dialog shows a list of available IEEE1588 enabled devices to choose from. If a new device is chosen the OK button will open a message box and ask you to restart the service. If you click OK the service is restarted and the new interface will be used for PTP messages.



8.3 Open Log Directory

A Windows Explorer window is opened showing all available log files.

9 API Description of the RTS IEEE 1588 Windows Version

The API is defined in the IEEE1588API.h header file. Sample sources and header files are available upon request. Please contact ieee1588@real-time-systems.com for more information.

9.1 Time-Representation Data Structure

```
typedef struct {
    unsigned long seconds;
    signed long nanoseconds;
} TimeRepresentation;
```

Two data fields to describe time are defined:

```
unsigned long TimeRepresentation::seconds
```

TimeRepresentation::seconds holds local clock time expressed in seconds. After the system starts up, this expresses the number of elapsed seconds since 1970.

```
signed long TimeRepresentation::nanoseconds
```

TimeRepresentation::nanoseconds holds local clock time in nanoseconds. After the system has started, this expresses the number of elapsed nanoseconds since 1970.

NOTE: if *nanoseconds* is less than zero, the entire time structure is understood to be negative.

9.2 Enumerations

The possible return values for the `ieee1588GetState()` function:

```
enum eIEEE1588State
{
    mSlaveStateNotSynced,
    mSlaveStateSynced,
    mMasterState
};
```

The possible return values for the `ieee1588ServiceGetState()` function:

```
enum eIEEE1588ServiceState
{
    mServiceStopped,
    mServiceRunning,
    mServiceOther,
    mServiceError
};
```

9.3 Service Control Functions

9.3.1 `ieee1588ServiceStop`

`signed long ieee1588ServiceStop(void)`

This function stops the IEEE1588Service.

Returns:

PTPD_OK - If service was stopped
PTPD_ERROR - If service could not be stopped

NOTE: Depending on the service state function execution can last up to 20 seconds!

9.3.2 `ieee1588ServiceStart`

`signed long ieee1588ServiceStart(void)`

This function starts the IEEE1588Service.

Returns:

PTPD_OK - If service was started
PTPD_ERROR - If service could not be started

9.3.3 ieee1588ServiceGetState

```
enum eIEEE1588ServiceState ieee1588ServiceGetState(void)
```

This function retrieves the status of the IEEE1588 service.

Returns:

- mServiceStopped*** - If service is stopped.
- mServiceRunning*** - If service is started.
- mServiceOther*** - If service is in transition state (stop or start is pending).
- mServiceError*** - If service state cannot be retrieved (e.g. not installed).

9.4 PTP Specific Functions

9.4.1 ieee1588GetAdapters

```
signed long ieee1588GetAdapters(IP_ADAPTER_INFO* AdapterInfo,  
                                signed long MaxAdapters)
```

This function retrieves information about the IEEE1588 enabled adapters in the system.

Parameters:

- AdapterInfo*** [in/out] – Array of IP_ADAPTER_INFO structs.
- MaxAdapters*** [in] – Number of IP_ADAPTER_INFO structs in *AdapterInfo*.

Returns:

Number of valid adapters in *AdapterInfo* array.

9.4.2 ieee1588GetTime

```
signed long ieee1588GetTime(TimeRepresentation *time)
```

This function retrieves the current IEEE1588 time.

Parameters:

- time*** [out] - Current IEEE1588 time

Returns:

- PTPD_OK*** - If okay
- PTPD_ERROR*** - If not initialized, not synch)

9.4.3 ieee1588GetTimeEx

```
signed long ieee1588GetTimeEx(TimeRepresentation *time, __int16 *utcOffset)
```

This function retrieves the current IEEE1588 time. It does not care about the TAI flag of the configuration. The time value will always be TAI time. The caller may calculate the UTC time by subtracting time->seconds - *utcOffset.

Parameters:

- time*** [out] - Current IEEE1588 time
- utcOffset*** [out] – Current offset to UTC in seconds

Returns:

PTPD_OK - If okay
PTPD_ERROR - If not initialized, not synch)

9.4.4 ieee1588GetGmMac

signed long ieee1588GetGmMac(char *masterId)

This function returns the mac of the current grand master selected by bmc.

Parameters:

masterId [out] area to store mac id of grand master field size must 6 bytes

Returns:

PTPD_OK - If okay
PTPD_ERROR - If not initialized

9.4.5 ieee1588GetState

enum eIEEE1588State ieee1588GetState(void)

This function returns the current IEEE1588 state.

enum returns:

mSlaveStateNotSynced - Slave and not synchronized to master.
MslaveStateSynced - Slave and synchronized to master.
MmasterState - Master.

9.4.6 ieee1588SetInboundLatency

signed long ieee1588SetInboundLatency(signed long nanoseconds)

This function sets the inbound latency.

Parameters:

nanoseconds [in] measured inbound latency in nanoseconds [should always be >= 0]

Returns:

PTPD_OK - If inbound latency was set
PTPD_ERROR - If service cannot be accessed

9.4.7 ieee1588SetOutboundLatency

signed long ieee1588SetOutboundLatency(signed long nanoseconds)

This function sets the outbound latency.

Parameters:

nanoseconds [in] measured outbound latency in nanoseconds [should always be =<0]

Returns:

PTPD_OK - If inbound latency was set
PTPD_ERROR - If service cannot be accessed

9.4.8 ieee1588SetInitialTime

signed long ieee1588SetInitialTime(__int64 time_sec)

This function sets the initial time for the IEEE1588 stack.

Parameters:

time_sec [in] time in seconds since 1970

Returns:

PTPD_OK - If okay
PTPD_ERROR - If failed (e.g. not master)

9.5 Callback Functions

9.5.1 ieee1588SetTimerCallback

NOTE: As Microsoft Windows is not a deterministic Operating System, it is not guaranteed that the intervals are always constant, especially if an interval smaller than e.g. 10ms has been selected. Still, it is ensured that no callbacks are lost.

HANDLE ieee1588SetTimerCallback(IEEE1588_TIME_CALLBACK CallBackFunc,
 unsigned long interval)

This function allows to set a callback function to get cyclic IEEE1588 time.

Parameters:

CallBackFunc [in] Functionpointer to a callback function (see below)
interval [in] Callback interval in milliseconds (minimum is 1 millisecond)

Returns:

HANDLE - If okay
INVALID_HANDLE_VALUE - If failed

Callback Function:

The callback function has to meet the following definition.

```
typedef void (CALLBACK *IEEE1588_TIME_CALLBACK)(TimeRepresentation *, BOOL);
```

e.g.

```
void CALLBACK TimeCallback(TimeRepresentation *time, BOOL bValid);
```

9.5.2 ieee1588FreeTimerCallback

signed long ieee1588FreeTimerCallback(HANDLE hHandle)

This function frees the callback installed via ieee1588SetTimerCallback().

Parameters:

hHandle [in] The handle returned by ieee1588SetTimerCallback().

Returns:

PTPD_OK - If okay
PTPD_ERROR - If failed

9.6 GPIO Functions

9.6.1 ieee1588ConfigureGpioPeriodicPulse

signed long ieee1588ConfigureGpioPeriodicPulse(unsigned long pin,
 TimeRepresentation *period,
 unsigned dutyCycle,
 unsigned polarity)

For Intel I350 cards a GPIO pin may be configured to generate a periodic pulse synchronized to the IEEE1588 clock.

NOTE: Before setting a new pulse period you have to turn off pulse generation by setting period to 0.

Parameters:

pin [in] – desired output pin
period [in] – pulse period (0 turns off pulse generation)
dutyCycle [in] – pulse duty cycle in percent (1-100)
polarity [in] – desired polarity (0 is positive, 1 is negative)

Returns:

PTPD_OK - If okay
PTPD_ERROR - If failed (e.g. not synchronized)

9.7 Configuration Functions

A set of functions to read and write the settings is provided.

9.7.1 ieee1588ConfigReadParameter

signed long ieee1588ConfigReadParameter(enum eIeee1588Parameter parameter,
 unsigned long *value)

This function reads a IEEE1588 configuration parameter.

Parameters:

parameter [in] desired parameter (see enum eIeee1588Parameter).
value [out] – value of parameter.

Returns:

PTPD_OK - If okay
PTPD_ERROR - If failed

9.7.2 ieee1588ConfigWriteParameter

signed long ieee1588ConfigWriteParameter(enum eIeee1588Parameter parameter,
 unsigned long value,
 BOOL bRestartService)

This function writes a IEEE1588 configuration parameter.

Parameters:

parameter [in] desired parameter (see enum `eIeee1588Parameter`).
value [in] – value of parameter.
bRestartService [in] – Restart service after parameter change.

Returns:

PTPD_OK - If okay
PTPD_ERROR - If failed

9.7.3 `ieee1588ConfigGetInterface`

signed long `ieee1588ConfigGetInterface(char *interfaceName,
unsigned len)`

This function reads the currently selected interface.

Parameters:

interfaceName [out] zero terminated string containing the interface name.
len [in] – max. length of `interfaceName` string in bytes.

Returns:

PTPD_OK - If okay
PTPD_ERROR - If failed

9.7.4 `ieee1588ConfigSetInterface`

signed long `ieee1588ConfigSetInterface(char *interfaceName,
unsigned len,
BOOL bRestartService)`

This function selects a interface to use for the service.

Parameters:

interfaceName [in] zero terminated string containing the interface name.
len [in] – length of `interfaceName` string in bytes.
bRestartService [in] – Restart service after parameter change.

Returns:

PTPD_OK - If okay
PTPD_ERROR - If failed

9.7.5 `ieee1588ConfigGetMasterMacAddress`

signed long `ieee1588ConfigGetMasterMacAddress(char *masterMac,
unsigned len)`

This function reads the current MAC address of the master.

Parameters:

masterMac [out] zero terminated string containing the master MAC
(Format "00:11:22:33:44:55").
len [in] – length of masterMac string in bytes.

Returns:

PTPD_OK - If okay
PTPD_ERROR - If failed

9.7.6 ieee1588ConfigSetMasterMacAddress

```
signed long ieee1588ConfigSetMasterMacAddress(char *masterMac,
                                              unsigned len,
                                              BOOL bRestartService)
```

This function sets a master MAC address.

Parameters:

masterMac [in] zero terminated string containing the master MAC
(Format "00:11:22:33:44:55").
len [in] – length of interfaceName string in bytes.
bRestartService [in] – Restart service after parameter change.

Returns:

PTPD_OK - If okay
PTPD_ERROR - If failed

10 Utilities

Two software tools are available, free-of-charge, for debugging purposes.

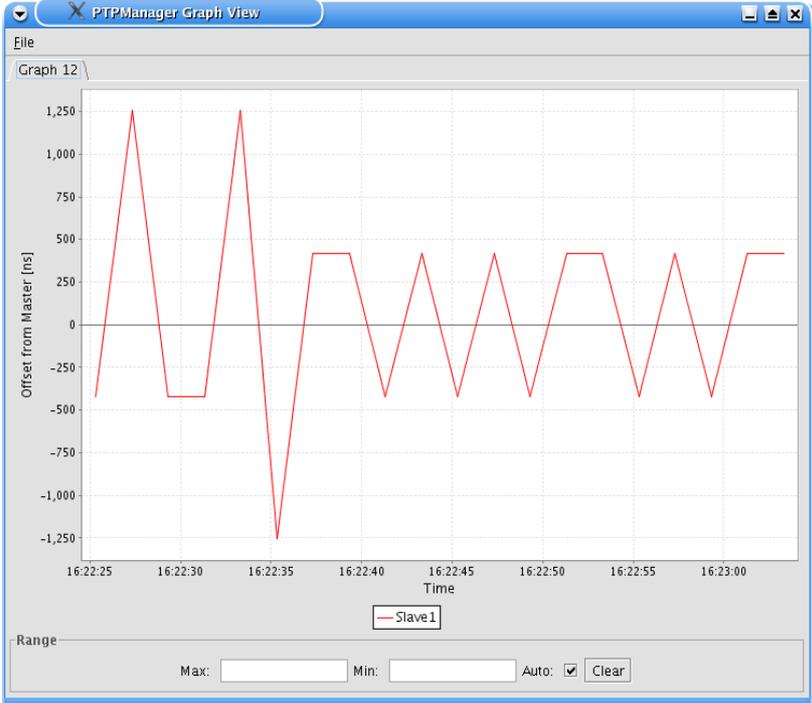
10.1 PTPManager

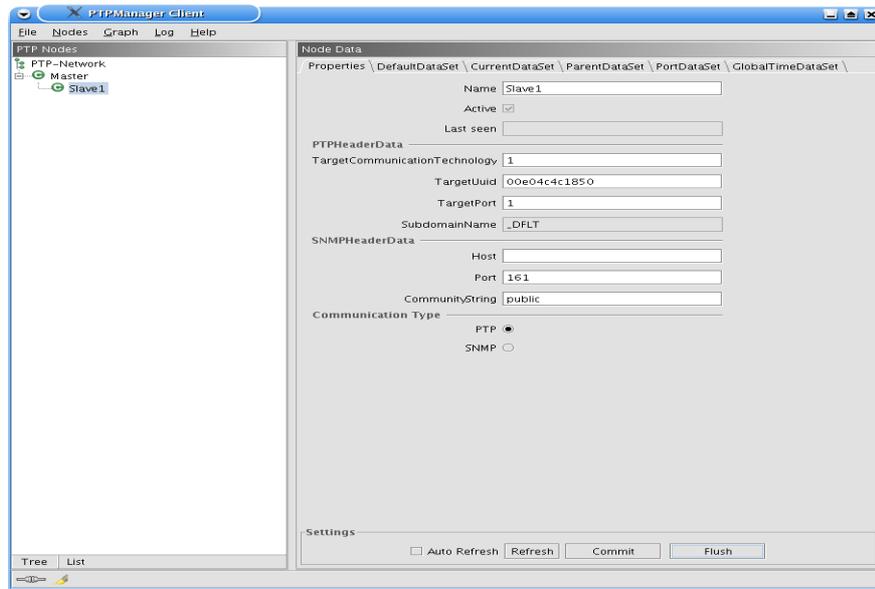
This convenient utility program from Zurich University of Applied Sciences (Switzerland) permits one to graphically view and edit all datasets defined by the IEEE 1588 PTP standard. In particular, it performs the following services:

- It does only support to talk to IEEE1588-2002 nodes
- Displays graphically the PTP network's synchronization hierarchy
- Displays graphically various views of PTP clock data such as the current offset of a local node's protocol clock from the master clock vs. time
- Collects and logs PTP clock data in .XML files, which may be easily viewed.
- Written in JAVA, the code is platform independent.

The **PTPManager** can be downloaded, for example, from a **Real-time Systems'** Web page. Refer to: [IEEE1588-2002 PTPManager](#).

Using the PTPManager [from Zurich University of Applied Sciences (Switzerland)], not only can the PTP software clock's initializing data be read, but the activities of the slave-nodes in the system being examined can also be recorded and visualized. Using the menu item **Graph**, graphical displays of up to 100 values captured from the **oneWayDelay** and **offsetFromMaster** variables can be generated. Using the PTPManager's **Log** feature, values from the IEEE 1588 protocol stack can be extracted, written to .XML files and displayed in an easy-to-read format.





10.2 PTPv2Browser

This tiny Windows tool provides management functionality for IEEE1588-2008 nodes. In particular it provides following features:

- Set/get of parameters in the data sets of each PTP device
- Group set function
- Slave offset monitoring with real time graphical view
- Configuration parameters and log results to/from files locally stored

Refer to: [IEEE1588-2008 PTPv2Browser](#)

10.3 Wireshark

This network protocol analyzer, implemented entirely in software, works very much like a hardware network analyzer. That is, it monitors, captures, and displays packets being transferred in a network cable. Because it is sensitive to 1588 PTP messages, it can display them in meaningful formats.

The tool is distributed under a GNU General Public License (GPL), and versions are available for a variety of host platforms, including Windows, OS X, and Linux.

The current version of the Wireshark program may be downloaded from the following website:

www.wireshark.org.

Communication between a master and a slave and communication and between the PTPManager and the slave may be seen in the following screenshot.

10.3.1 Protocol related Wireshark Identifiers

Following table describes a set of parameters configurable on stack startup. If you have nodes where you don't know the current configuration of these parameters you might use WireShark to determine how to configure the stack to operate with those devices.

Protocol Version	Wireshark Identifier	Description
V1	SyncInterval (part of sync message)	Represents the time between Sync Messages sent by the Master. $2^{\text{syncInterval}} = \text{sync interval in seconds}$ Default value is 1. Configuration must be equal on all nodes in the same domain otherwise nodes may change into a faulty state. This interval is configurable with the stack parameter: syncInterval
V1	SubDomain (part of sync message)	Domain identifier (String) which groups nodes that belong to the same synchronization area. For example nodes may exist that are in different functional groups eg. Group 1: control units and Group 2: monitoring units which share the same physical network but do not use the same base time. Group 2 may have a real world time provided by a master with GPS. Group 1 may have a time based on a central control unit where the time starts at 0 when you turn on the central control unit. Default is „_DFLT“ already specified domains are „_ALT1“, „_ALT2“, „_ALT_3“. This is configured with the parameter „domain“ on stack startup.
V2	LogMessagePeriod (part of sync message)	Represents the time between Sync Messages sent by the Master. $2^{\text{LogMessagePeriod}} = \text{sync interval in seconds}$ Default value is 0. Configuration must be equal on all nodes in the same domain otherwise nodes may change into a faulty state. This interval is configurable with the stack parameter: syncInterval
V2	LogMessagePeriod (part of announce message)	Represents the time between Announce Messages sent by the Master. $2^{\text{LogMessagePeriod}} = \text{announce interval in seconds}$ Default value is 1. Configuration must be equal on all nodes in the same domain otherwise nodes may change into a faulty state. This parameter is not configurable and represents DEFAULT_ANNOUNCE_INTERVAL which is 1.
V2	subdomainNumber	Domain identifier. It is mapped accordingly V1 -> V2 mapping. „_DFLT“ : 0 „_ALT1“ : 1 „_ALT1“ : 2 „_ALT1“ : 3 Default value is 0. It is configured with the domain parameter on stack startup.

11 END OF DOCUMENT