# ZebOS-XP®
# Network Platform

## Version 1.4
## Extended Performance

## Precision Time Protocol
## Developer Guide

December 2015

IP Infusion Inc.
3965 Freedom Circle, Suite 200
Santa Clara, CA 95054
+1 408-400-1900
http://www.ipinfusion.com/

For support, questions, or comments via E-mail, contact:
support@ipinfusion.com

# Contents

# Preface

This guide describes the ZebOS-XP application programming interface (API) for Precision Time Protocol (PTP).

## Audience

This guide is intended for developers who write code to customize and extend PTP.

## Conventions

Table P-1 shows the conventions used in this guide.

**Table P-1: Conventions**

| Convention | Description |
|---|---|
| *Italics* | Emphasized terms; titles of books |
| Note: | Special instructions, suggestions, or warnings |
| `monospaced type` | Code elements such as commands, functions, parameters, files, and directories |

## Contents

This document contains this chapter:

- Chapter 1, *Precision Time Protocol API*

## Related Documents

The following guides are related to this document:

- *Precision Time Protocol Command Reference*
- *Precision Time Protocol Configuration Guide*

Note: All ZebOS-XP technical manuals are available to licensed customers at http://www.ipinfusion.com/support/document_list.

## Support

For support-related questions, contact support@ipinfusion.com.

## Comments

If you have comments, or need to report a problem with the content, contact techpubs@ipinfusion.com.

# CHAPTER 1 Precision Time Protocol API

Precision Time Protocol (PTP), as specified in the IEEE Standard 1588-2008, is a distributed protocol that specifies how real-time clocks in the system synchronize with each other. PTP operates within a logical scope called a domain. The clocks are organized in a master−slave synchronization hierarchy. The grandmaster clock is at the top of the hierarchy:

- The grandmaster clock determines the reference time for the entire system.
- Slaves use the timing information to adjust their clocks to match the time of their master in the hierarchy.
- Clocks are synchronized by exchanging messages.

In ZebOS-XP, PTP is handled by an independent daemon named `ptpd`.

## PTP Devices

PTP devices include ordinary clocks, boundary clocks and transparent clocks:

- An *ordinary clock* communicates with the network through a single physical port. An ordinary clock can be a master or slave.
- A *boundary clock* has multiple ports from which it selects the best master clock. One port is a slave to that master clock while the other ports are masters to downstream slaves. A boundary clock provides a means to synchronize time among subnetworks.
- A *transparent clock* forwards all PTP messages but measures the time that the packet takes to traverse the transparent clock and in some cases the link delay of the port where the packet entered.

### PTP Algorithm

The PTP process has two phases:

- Establishing the master-slave hierarchy
- Synchronizing the clocks.

Within a PTP domain, each port of an ordinary or boundary clock follows this process to determine its role as master or slave:

- Examines the contents of all received announce messages (issued by ports in the master state)
- Compares the clocks of the foreign master (in the announce message) to the local clock for priority, clock class, accuracy, and so on
- Based on this comparison, determines its role as either master or slave

After the master-slave hierarchy is established, the clocks are synchronized by the master sending synchronization messages to the slave. The devices in a PTP domain also exchange messages about link delays.

## Data Structures

The data structures in this section are defined in `ptpd\ptp_cli.h`.

## ptp_bridge

```
struct ptp_bridge
{
  struct ptp_master *ptp_master;

  /* Contains the data needed for PTP which is same across all ports */
  struct ptp_clock *clock;

  /* Port list */
  struct avl_tree *port_tree;

  /* Bridge Name */
  u_int8_t name[L2_BRIDGE_NAME_LEN + 1];
  u_int8_t bridge_type;
  u_int8_t  ptp_enabled;
};
```

## ptp_clock

```
struct ptp_clock
{
  u_int8_t  clock_type;
  u_int8_t  clock_communication_technology;
  bool_t is_boundary_clock;
  bool_t is_transparent_clock;
  //u_int8_t  subdomain_name[PTP_SUBDOMAIN_NAME_LENGTH];
  struct ptp_default_ds  ptp_dds;
  struct ptp_current_ds  ptp_cds;
  struct ptp_parent_ds   ptp_pds;
  struct ptp_time_properties_ds  ptp_tpds;
  struct ptp_transparent_clock_default_ds   ptp_tc_cds;
  struct ptp_syntonize ptp_synt;
  bool_t is_enable;
  u_int8_t transport_type;
  u_int8_t delay_mechanism;
  u_char send_msg[PTP_MSG_SIZE];
  bool_t ptp_syntonize;
};
```

## ptp_ifp

```
struct ptp_ifp
{
  struct interface *ifp;
  struct ptp_bridge *br;
  struct ptp_master *ptpm;
  struct ptp_port *port;
  s_int32_t ifindex;
  u_int16_t pvid;
  bool_t active;
  u_int8_t ptp_enabled; /* Flag to indicate ptp enable/disable state */
  u_int8_t port_type;
  u_int8_t dev_addr [PTP_MAC_ADDR_LEN];
  struct sockaddr skaddr;
  struct sockaddr_in6 skaddr6;
```

```
  /* Time intervals */
  u_int8_t annrec_time;
  u_int8_t qualification_time;
  // u_int8_t sync_time;
  float32_t sync_time;
  u_int8_t ann_int_time;
  u_int8_t pdelay_time;
};
```

## ptp_master

```
struct ptp_master
{
  /* Pointer to VR.  */
  struct ipi_vr *vr;

  /* Pointer to globals.  */
  struct lib_globals *zg;

  struct list *bridge_list;

    /* Tree of all interfaces in the system */
  struct list *if_list;
  struct
  {
 /* Debug flags for configuration. */
   struct debug_ptp conf;
   struct debug_ptp term;
  } debug;

  pal_sock_handle_t sockfd;

  u_int8_t ether_type;
  struct thread *ptp_rcv_thread;
};
```

## ptp_port_clock

```
struct ptp_port_clock
{
  u_int16_t  number_foreign_records;
  s_int32_t   ifindex;
  u_int8_t mac[6]; /* TO DO */
  pal_sock_handle_t sockfd;
  pal_sock_handle_t sockfd_gen;
  struct ptp_port_ds   ptp_port_ds;
  struct ptp_transparent_clock_port_default_ds   ptp_tc_cport_ds;
  struct ptp_foreign_master_ds  ptp_fmds[PTP_NUM_FOREIGN_RECS];
  struct ptp_protocol_ds ptp_proto_ds;
  struct ptp_timestamp last_sync_send_time;
  struct ptp_timestamp last_pdr_send_time;
  struct ptp_timestamp last_dresp_send_time;
  struct ptp_timestamp last_pdreq_recv_time;
  struct ptp_timestamp last_dreq_recv_time;
  struct ptp_timestamp last_delay_req_send_time;
  struct thread *ptp_rcv_thread;
```

```
    struct thread *ptp_rcv_thread_gen;
    u_int8_t transport_type;
};
```

# API Reference

| Function | Description |
|---|---|
| add_unicast_neighbor_api | Adds a unicast neighbor |
| create_clock_api | Creates a clock on the default bridge |
| create_port_api | Initializes a PTP port clock data structure and binds it to a PTP port |
| delete_clock_api | Deletes a clock and all its related ports on the specified bridge |
| delete_port_api | Deletes a PTP clock port |
| display_ptp_clock_foreign_master_dataset | Gets the attributes of foreign master clocks |
| display_ptp_port_dataset | Gets the attributes of a PTP port |
| find_clock_api | Returns a pointer to a PTP clock installed on the default bridge |
| ptp_api_disable_bridge_global | Disables PTP on a bridge |
| ptp_api_enable_bridge_global | Enables PTP on a bridge |
| ptp_bridge_add_port | Adds a clock port to a bridge |
| ptp_find_bridge | Returns a pointer to a PTP bridge |
| ptp_launch_recv | Sets up threads for receiving PTP messages |
| ptp_master_get | Returns the PTP master data structure |
| PTP_set_state | Sets the PTP state |
| ptp_sock_init | Initializes a socket |
| rem_unicast_neighbor_api | Deletes a unicast neighbor |
| send_announce | Sends a PTP announce message |
| show_current_ds | Gets the current data set of a clock |
| show_default_ds | Gets the default data set of an ordinary or boundary clock |
| show_parent_ds | Gets the data set of the parent clock and grandmaster clock |
| show_time_properties | Gets the attributes of the timescale |
| show_transparent_clock_ds | Gets the data set of a transparent clock |

## add_unicast_neighbor_api

This function adds a unicast neighbor.

### Syntax

```
#include "ptp_utils.h"
s_int32_t
add_unicast_neighbor_api(const char *addr, struct ptp_port_clock * port)
```

### Input Parameters

| | |
|---|---|
| `addr` | Neighbor address |
| `portclock` | Pointer to PTP clock port |

### Output Parameters

None

### Return Values

PTP_MAX_NEIGHBOR_CROSSED when the maximum number of unicast neighbors exists

PTP_ADDR_NOT_PROPER when the address is not valid

PTP_ADDR_NOT_PRESENTABLE when the address is already a neighbor

PTP_ERROR_AVL_INSERT when there is an internal error

RESULT_OK when the function succeeds

## create_clock_api

This function creates a clock on the default bridge and initializes it.

### Syntax

```
#include "ptpd/ptp_cli.h"
s_int32_t
create_clock_api(u_int8_t clk_type, u_int8_t domain_id, struct ptp_clock *clock)
```

### Input Parameters

`clk_type` One of these constants from `ptpd/ptp_types.h`:

- `PTP_ORDINARY_CLOCK`
  A clock that has a single PTP port and is a master or slave
- `PTP_BOUNDARY_CLOCK`
  A clock that has multiple ports from which it selects the best master clock. One port is a slave to that master clock while the other ports are masters to downstream slaves. A boundary clock provides a means to synchronize time among subnetworks.
- `PTP_TRANSPARENT_CLOCK`
  A clock that measures the time for a PTP event message to travel through the device and then provides this information to other clocks

`domain_id` The PTP domain identifier

### Output Parameters

| | |
|---|---|
| `clock` | A pointer to the clock initialized with default values |

### Return Values

Always RESULT_OK

## create_port_api

This function initializes a PTP port clock data structure and binds it to a PTP port.

### Syntax

```
#include "ptpd/ptp_cli.h"
s_int32_t
    create_port_api (struct ptp_ifp * ptp_if, struct ptp_port_clock * portclock)
```

### Input Parameters

| | |
|---|---|
| `ptp_if` | Pointer to PTP interface |
| `portclock` | Pointer to PTP clock port |

### Output Parameters

None

### Return Values

RESULT_OK when the PTP port was created and initialized

PTP_PORT_NOT_FOUND when the PTP port is NULL

PTP_CLOCK_NOT_FOUND when the PTP clock is NULL

PTP_BRIDGE_NOT_FOUND when the PTP bridge is NULL

PTP_ORDINARY_MORE_THAN_ONE_PORT when there is an attempt to create more than one port on ordinary clock

## delete_clock_api

This function deletes a clock and all its related ports on the specified bridge.

### Syntax

```
#include "ptpd/ptp_cli.h"
s_int32_t
    delete_clock_api (struct ptp_bridge * bridge)
```

### Input Parameters

| | |
|---|---|
| `bridge` | A pointer to the PTP bridge |

### Output Parameters

None

### Return Values

RESULT_OK when the clock was deleted

 

RESULT_ERROR when the bridge is NULL

## delete_port_api

This function deletes a PTP clock port.

### Syntax

```
#include "ptpd/ptp_cli.h"
s_int32_t
    delete_port_api (struct ptp_ifp *ptp_if)
```

### Input Parameters

| | |
|---|---|
| ptp_if | Pointer to PTP interface |

### Output Parameters

None

### Return Values

RESULT_OK when the clock port was deleted

RESULT_ERROR when the port or clock is NULL

## display_ptp_clock_foreign_master_dataset

This function gets the attributes of foreign master clocks which are ordinary clocks or boundary clocks that are sending announce messages to this clock but are not the current master recognized by this clock.

### Syntax

```
#include "ptpd/ptp_cli.h"
void
    display_ptp_clock_foreign_master_dataset (struct cli* cli,
    struct ptp_port_clock *clock)
```

### Input Parameters

| | |
|---|---|
| clock | A pointer to the clock port |

### Output Parameters

| | |
|---|---|
| cli | The attributes of the foreign masters |

### Return Values

None

## display_ptp_port_dataset

This function gets the attributes of a PTP port.

### Syntax

```
#include "ptpd/ptp_cli.h"
void
```

```
display_ptp_port_dataset (struct cli* cli, struct ptp_port_clock *clock)
```

### Input Parameters

| | |
|---|---|
| `clock` | A pointer to the clock port |

### Output Parameters

| | |
|---|---|
| `cli` | The attributes of the port |

### Return Values

None

## find_clock_api

This function returns a pointer to a PTP clock installed on the default bridge.

### Syntax

```
#include "ptpd/ptp_cli.h"
struct ptp_clock*
    find_clock_api()
```

### Input Parameters

None

### Output Parameters

None

### Return Values

A pointer to the clock when successful

NULL when the default bridge does not have a clock

## ptp_api_disable_bridge_global

This function disables PTP on a bridge.

### Syntax

```
#include "ptpd/ptp_utils.h"
s_int32_t
    ptp_api_disable_bridge_global (u_char *bridge_name)
```

### Input Parameters

| | |
|---|---|
| `bridge_name` | The name of the bridge |

### Output Parameters

None

### Return Values

RESULT_OK when PTP was disabled

PTP_ERR_GENERIC is the bridge name is empty

PTP_ERR_PTP_NOT_ENABLED when PTP is not enabled on the bridge

## ptp_api_enable_bridge_global

This function enables PTP on a bridge.

### Syntax

```
#include "ptpd/ptp_utils.h"
s_int32_t
    ptp_api_enable_bridge_global (u_char *br_name)
```

### Input Parameters

| | |
|---|---|
| `br_name` | The name of the bridge |

### Output Parameters

None

### Return Values

RESULT_OK when PTP was enabled

PTP_INTERFACE_NOT_FOUND when the interface is NULL

PTP_INTERFACE_ENABELED_NO_PORT_DATA when the clock port is NULL and PTP is enabled on the interface

PTP_MEMORY_ALLOC when there is a memory allocation error

PTP_PORT_NOT_FOUND when the PTP port is NULL

PTP_CLOCK_NOT_FOUND when the PTP clock is NULL

PTP_BRIDGE_NOT_FOUND when the PTP bridge is NULL

PTP_ORDINARY_MORE_THAN_ONE_PORT when trying to create more than one port for an ordinary clock

## ptp_bridge_add_port

This function adds a clock port to a bridge.

### Syntax

```
#include "ptpd/ptp_bridge.h"
s_int16_t
    ptp_bridge_add_port (char *bridge_name, struct interface *ifp)
```

### Input Parameters

| | |
|---|---|
| `bridge_name` | The name of the bridge |
| `ifp` | A pointer to an interface |

### Output Parameters

None

### Return Values

RESULT_OK when the port was added

RESULT_ERROR when the bridge or interface is NULL or when PTP is not associated with the interface

## ptp_find_bridge

This function returns a pointer to a PTP bridge.

### Syntax

```
#include "ptpd/ptp_bridge.h"
struct ptp_bridge *
    ptp_find_bridge (char *name)
```

### Input Parameters

| | |
|---|---|
| `name` | The name of the bridge |

### Output Parameters

None

### Return Values

A pointer to the bridge when successful

NULL when a bridge with the given name does not exist

## ptp_launch_recv

This function sets up threads for receiving PTP messages.

### Syntax

```
#include "ptpd/ptp_utils.h"
void
    ptp_launch_recv (struct ptp_ifp *ptp_if)
```

### Input Parameters

| | |
|---|---|
| `ptp_if` | Pointer to PTP interface |

### Output Parameters

None

### Return Values

None

## ptp_master_get

This function returns the PTP master data structure.

### Syntax

```
#include "ptpd/ptpd.h"
```

 

```
struct ptp_master *
    ptp_master_get (void)
```

### Input Parameters

None

### Output Parameters

None

### Return Values

PTP master data structure

## PTP_set_state

This function sets the PTP state.

### Syntax

```
#include "ptpd/ptp_fsm.h"
void
PTP_set_state (void *data, PTP_states_e new_state)
```

### Input Parameters

| | |
|---|---|
| `data` | Pointer to PTP interface (internally cast to `struct ptp_ifp *`) |
| `new_state` | One of these constants defined in the `PTP_states` enum in `ptpd/ptp_fsm.h`. See the IEEE 1588-2008 standard for an explanation of these states: |

`PTP_MASTER`

`PTP_UNCALIBRATED`

`PTP_SLAVE`

`PTP_LISTENING`

`PTP_PRE_MASTER`

`PTP_PASSIVE`

`PTP_INITIALIZING`

`PTP_DISABLE`

`PTP_FAULTY`

### Output Parameters

None

### Return Values

None

## ptp_sock_init

This function initializes a socket.

### Syntax

```
#include "ptpd/ptp_sock.h"
pal_sock_handle_t
    ptp_sock_init (struct lib_globals *zg, u_int8_t transport_type,
    struct ptp_port_clock *port_data, struct interface *ifp)
```

### Input Parameters

`zg` Daemon-specific library globals

`transport_type` One of these constants from `ptpd/ptp_types.h`:

`PTP_UDP_V4`
Enable PTP over UDP for IPv4

`PTP_UDP_V6`
Enable PTP over UDP for IPv6

`PTP_ETHER`
Enable PTP over Ethernet

`ifp` Pointer to interface

### Output Parameters

`port_data` Pointer to PTP clock port

### Return Values

The socket file descriptor when successful

RESULT_ERROR when the raw socket for the `PTP_ETHER` option could not be created

## rem_unicast_neighbor_api

This function deletes a unicast neighbor.

### Syntax

```
#include "ptp_utils.h"
s_int32_t
rem_unicast_neighbor_api(const char *addr, struct ptp_port_clock * port)
```

### Input Parameters

`addr` Neighbor address; pass NULL to delete all unicast neighbors

`portclock` Pointer to PTP clock port

### Output Parameters

None

### Return Values

PTP_ERR_DEL_ADDR when there was an error deleting the address or addresses

RESULT_OK when the function succeeds

## send_announce

This function sends a PTP announce message.

### Syntax

```
#include "ptpd/ptp_utils.h"
void
    send_announce (struct ptp_clock *clock, struct ptp_port_clock *port_data,
    int send)
```

### Input Parameters

| | |
|---|---|
| `clock` | A pointer to the clock |
| `port_data` | Pointer to PTP clock port |
| `send` | Whether to send the message immediately |

### Output Parameters

None

### Return Values

None

## show_current_ds

This function gets the current data set of a clock.

### Syntax

```
#include "ptpd/ptp_cli.h"
void
    show_current_ds (struct cli *cli, struct ptp_clock *clock)
```

### Input Parameters

| | |
|---|---|
| `clock` | A pointer to the clock |

### Output Parameters

| | |
|---|---|
| `cli` | The current data set of the clock |

### Return Values

None

## show_default_ds

This function gets the default data set of an ordinary or boundary clock.

### Syntax

```
#include "ptpd/ptp_cli.h"
void
    show_default_ds (struct cli* cli, struct ptp_clock *clock)
```

### Input Parameters

`clock` A pointer to an ordinary or boundary clock

### Output Parameters

`cli` The default data set of the ordinary or boundary clock

### Return Values

None

## show_parent_ds

This function gets the data set of the parent clock and grandmaster clock.

### Syntax

```
#include "ptpd/ptp_cli.h"
void
    show_parent_ds (struct cli *cli, struct ptp_clock *clock)
```

### Input Parameters

`clock` A pointer to the clock

### Output Parameters

`cli` The data set of the parent clock

### Return Values

None

## show_time_properties

This function gets the attributes of the timescale.

### Syntax

```
#include "ptpd/ptp_cli.h"
void
    show_time_properties (struct cli* cli, struct ptp_clock *clock)
```

### Input Parameters

`clock` A pointer to the clock

### Output Parameters

`cli` The attributes of the timescale

### Return Values

None

## show_transparent_clock_ds

This function gets the data set of a transparent clock.

### Syntax

```
#include "ptpd/ptp_cli.h"
void
    show_transparent_clock_ds (struct cli* cli, struct ptp_clock *clock)
```

### Input Parameters

| | |
|---|---|
| `clock` | A pointer to the clock |

### Output Parameters

| | |
|---|---|
| `cli` | The data set of the transparent clock |

### Return Values

None

# Index