# Time Of Tone Master
*Component Design Document*

## 1    Description

This is the Time of Tone Master component. It provides an alternate implementation to the Time at Tone Master component also provided within Adamant. This implementation could potentially provide more accurate time syncing if your system has the ability to accurately determine the time at which the tone is sent outside of this component (ie. the time when the tone leaves the serial port). TaT is a protocol used to sync a slave clock to a master clock. Two messages are sent from the master to the slave component. First a 'tone' message is sent which signals to the slave clock to save its current time. Next, a 'time' message is sent which provides the master time at which the 'tone' was sent. This time combined with the time the slave recorded when the 'tone' was received can be used to calculate a time delta of the slave clock with respect to the master. This component implements the master side of the protocol. This component outputs the time message and the tone as Tick.T send connectors. This design is intended to be generic enough to implement time at tone in many different manners on the other end of these connectors. For instance, you could convert the time message Tick.T to a CCSDS packet and the tone Tick.T to a GPIO pulse.

## 2    Requirements

The requirements for the Time of Tone Master are specified below.

1. The component shall send the time at tone tone message at a compile-time configurable rate.
2. The component shall send the time at tone time message after sending each time at tone tone message.
3. The component shall send a time at tone transaction one time on command.
4. The component shall produce a data product relating the number time at tone transactions sent since startup.

## 3    Design

### 3.1    At a Glance

Below is a list of useful parameters and statistics that give a quick look into the makeup of the component.

- **Execution** - *passive*
- **Number of Connectors** - 9
- **Number of Invokee Connectors** - 3
- **Number of Invoker Connectors** - 6
- **Number of Generic Connectors** - *None*
- **Number of Generic Types** - *None*

- **Number of Unconstrained Arrayed Connectors** - *None*

- **Number of Commands** - 3

- **Number of Parameters** - *None*

- **Number of Events** - 4

- **Number of Faults** - *None*

- **Number of Data Products** - 3

- **Number of Data Dependencies** - *None*
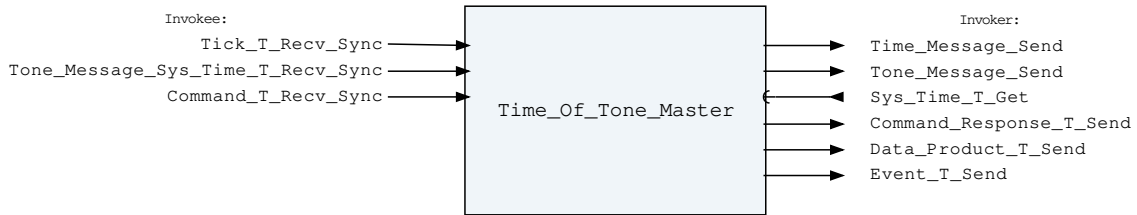
- **Number of Packets** - *None*

## 3.2 Diagram



Figure 1: Time Of Tone Master component diagram.

## 3.3 Connectors

Below are tables listing the component's connectors.

### 3.3.1 Invokee Connectors

The following is a list of the component's *invokee* connectors:

Table 1: Time Of Tone Master Invokee Connectors

| Name | Kind | Type | Return_Type | Count |
|------|------|------|-------------|-------|
| Tick_T_Recv_Sync | recv_sync | Tick.T | - | 1 |
| Tone_Message_ Sys_Time_T_Recv_ Sync | recv_sync | Sys_Time.T | - | 1 |
| Command_T_Recv_ Sync | recv_sync | Command.T | - | 1 |

Connector Descriptions:
- **Tick_T_Recv_Sync** - Tick used to trigger the sending of tone messages.

- **Tone_Message_Sys_Time_T_Recv_Sync** - This connector triggers the sending of the time message. The time received here is assumed to be an accurate time stamp of when the tone message was sent. This time can be provided to this component by a lower level component which actually records the time the tone message leaves the system. If left unconnected, no time messages will be produced by this component. In that case, the slave must use the time passed in the tone message directly, which is a less accurate measure of the master's system time.

- **Command_T_Recv_Sync** - The command receive connector.

### 3.3.2 Invoker Connectors

The following is a list of the component's *invoker* connectors:

Table 2: Time Of Tone Master Invoker Connectors

| Name | Kind | Type | Return_Type | Count |
|---|---|---|---|---|
| `Time_Message_Send` | `send` | `Tick.T` | - | 1 |
| `Tone_Message_Send` | `send` | `Tick.T` | - | 1 |
| `Sys_Time_T_Get` | `get` | - | `Sys_Time.T` | 1 |
| `Command_Response_T_Send` | `send` | `Command_Response.T` | - | 1 |
| `Data_Product_T_Send` | `send` | `Data_Product.T` | - | 1 |
| `Event_T_Send` | `send` | `Event.T` | - | 1 |

Connector Descriptions:

- **Time_Message_Send** - Time message send connector, sends a message with the time the tone message will be sent.

- **Tone_Message_Send** - Tone message send connector.

- **Sys_Time_T_Get** - Used to get system time, used by the master version of the component to get the current time.

- **Command_Response_T_Send** - This connector is used to register the components commands with the command router component.

- **Data_Product_T_Send** - The data product invoker connector

- **Event_T_Send** - The event send connector

## 3.4 Interrupts

This component contains no interrupts.

## 3.5 Initialization

Below are details on how the component should be initialized in an assembly.

### 3.5.1 Component Instantiation

This component contains no instantiation parameters in its discriminant.

### 3.5.2 Component Base Initialization

This component contains no base class initialization, meaning there is no `init_Base` subprogram for this component.

### 3.5.3 Component Set ID Bases

This component contains commands, events, packets, faults, or data products that require a base identifier to be set at initialization. The `set_Id_Bases` procedure must be called with the following parameters:

Table 3: Time Of Tone Master Set Id Bases Parameters

| Name | Type |
|---|---|
| `Event_Id_Base` | `Event_Types.Event_Id_Base` |

| | |
|---|---|
| Data_Product_Id_Base | Data_Product_Types.Data_Product_Id_Base |
| Command_Id_Base | Command_Types.Command_Id_Base |

Parameter Descriptions:
- **Event_Id_Base** - The value at which the component's event identifiers begin.
- **Data_Product_Id_Base** - The value at which the component's data product identifiers begin.
- **Command_Id_Base** - The value at which the component's command identifiers begin.

### 3.5.4 Component Map Data Dependencies

This component contains no data dependencies.

### 3.5.5 Component Implementation Initialization

The calling of this implementation class initialization procedure is mandatory. The component achieves implementation class initialization using the `init` subprogram. The `init` subprogram requires the following parameters:

Table 4: Time Of Tone Master Implementation Initialization Parameters

| Name | Type | Default Value |
|---|---|---|
| Sync_Period | Positive | 1 |
| Enabled_State | Tat_State.Tat_State_Type | Tat_State.Enabled |

Parameter Descriptions:
- **Sync_Period** - The number of ticks between sending clock sync messages.
- **Enabled_State** - Is time at tone enabled or disabled by default at startup.

## 3.6 Commands

Commands for the Time of Tone Master component.

Table 5: Time Of Tone Master Commands

| Local ID | Command Name | Argument Type |
|---|---|---|
| 0 | Enable_Time_At_Tone | – |
| 1 | Disable_Time_At_Tone | – |
| 2 | Sync_Once | – |

Command Descriptions:
- **Enable_Time_At_Tone** - This enables the sending of time at tone messages.
- **Disable_Time_At_Tone** - This enables the sending of time at tone messages.
- **Sync_Once** - This sends a time at tone message followed by a tone message at the next tick, regardless of the current sync period. This is useful during testing to send a sync one time.

## 3.7 Parameters

The Time Of Tone Master component has no parameters.

## 3.8 Events

Events for the Time of Tone Master component.

Table 6: Time Of Tone Master Events

| Local ID | Event Name | Parameter Type |
|---|---|---|
| 0 | Time_At_Tone_Enabled | – |
| 1 | Time_At_Tone_Disabled | – |
| 2 | Sending_Sync_Once | – |
| 3 | Invalid_Command_Received | Invalid_Command_Info.T |

Event Descriptions:

- **Time_At_Tone_Enabled** - The time at tone has been enabled by command.

- **Time_At_Tone_Disabled** - The time at tone has been disabled by command.

- **Sending_Sync_Once** - The component will sent the time at tone message and tone message at the next received tick.

- **Invalid_Command_Received** - A command was received with invalid parameters.

## 3.9 Data Products

Data products for the Time of Tone Master component.

Table 7: Time Of Tone Master Data Products

| Local ID | Data Product Name | Type |
|---|---|---|
| 0x0000 (0) | Tone_Messages_Sent | Packed_U32.T |
| 0x0001 (1) | Time_Messages_Sent | Packed_U32.T |
| 0x0002 (2) | Time_At_Tone_State | Tat_State.T |

Data Product Descriptions:

- **Tone_Messages_Sent** - The number of tone messages sent.

- **Time_Messages_Sent** - The number of time messages sent.

- **Time_At_Tone_State** - The disable/enable state of the time at tone component.

## 3.10 Packets

The Time Of Tone Master component has no packets.

# 4 Unit Tests

The following section describes the unit test suites written to test the component.

## 4.1 *Time_Of_Tone_Master_Tests* Test Suite

This is a unit test suite for the Time of Tone Master component.

Test Descriptions:

- **Test_Tone_Message** - This test ensures that tone messages are sent out appropriately when Ticks are sent to the component.

- **Test_Time_Message** - This test ensures that time messages are sent out appropriately when tone timestamps are sent to the component.

- **Test_Enable_Disabled** - This test ensures that enable and disable commands work.

- **Test_Sync_Once** - This test ensures that the Sync_Once command works.

- **Test_Invalid_Command** - This test ensures that an invalid command is rejected and reported.

# 5 Appendix

## 5.1 Preamble

This component contains no preamble code.

## 5.2 Packed Types

The following section outlines any complex data types used in the component in alphabetical order. This includes packed records and packed arrays that might be used as connector types, command arguments, event parameters, etc..

### Command.T:

Generic command packet for holding arbitrary commands

Table 8: Command Packed Record : 2080 bits *(maximum)*

| Name | Type | Range | Size (Bits) | Start Bit | End Bit | Variable Length |
|------|------|-------|-------------|-----------|---------|-----------------|
| Header | Command_ Header.T | - | 40 | 0 | 39 | – |
| Arg_Buffer | Command_ Types. Command_Arg_ Buffer_Type | - | 2040 | 40 | 2079 | Header.Arg_ Buffer_Length |

Field Descriptions:
- **Header** - The command header

- **Arg_Buffer** - A buffer to that contains the command arguments

### Command_Header.T:

Generic command header for holding arbitrary commands

Table 9: Command_Header Packed Record : 40 bits

| Name | Type | Range | Size (Bits) | Start Bit | End Bit |
|------|------|-------|-------------|-----------|---------|
| Source_Id | Command_Types. Command_Source_Id | 0 to 65535 | 16 | 0 | 15 |
| Id | Command_Types. Command_Id | 0 to 65535 | 16 | 16 | 31 |

| | | | | | |
|---|---|---|---|---|---|
| Arg_Buffer_Length | Command_Types. Command_Arg_Buffer_ Length_Type | 0 to 255 | 8 | 32 | 39 |

Field Descriptions:
- **Source_Id** - The source ID. An ID assigned to a command sending component.
- **Id** - The command identifier
- **Arg_Buffer_Length** - The number of bytes used in the command argument buffer

## Command_Response.T:

Record for holding command response data.

Table 10: Command_Response Packed Record : 56 bits

| Name | Type | Range | Size (Bits) | Start Bit | End Bit |
|---|---|---|---|---|---|
| Source_Id | Command_ Types.Command_ Source_Id | 0 to 65535 | 16 | 0 | 15 |
| Registration_ Id | Command_ Types.Command_ Registration_ Id | 0 to 65535 | 16 | 16 | 31 |
| Command_Id | Command_Types. Command_Id | 0 to 65535 | 16 | 32 | 47 |
| Status | Command_Enums. Command_ Response_ Status.E | 0 => Success<br>1 => Failure<br>2 => Id_Error<br>3 => Validation_Error<br>4 => Length_Error<br>5 => Dropped<br>6 => Register<br>7 => Register_Source | 8 | 48 | 55 |

Field Descriptions:
- **Source_Id** - The source ID. An ID assigned to a command sending component.
- **Registration_Id** - The registration ID. An ID assigned to each registered component at initialization.
- **Command_Id** - The command ID for the command response.
- **Status** - The command execution status.

## Data_Product.T:

Generic data product packet for holding arbitrary data types

Table 11: Data_Product Packed Record : 344 bits *(maximum)*

| Name | Type | Range | Size (Bits) | Start Bit | End Bit | Variable Length |
|---|---|---|---|---|---|---|
| Header | Data_Product_ Header.T | - | 88 | 0 | 87 | – |

| Buffer | Data_Product_<br>Types.Data_<br>Product_<br>Buffer_Type | - | 256 | 88 | 343 | Header.Buffer_<br>Length |
|---|---|---|---|---|---|---|

Field Descriptions:
- **Header** - The data product header
- **Buffer** - A buffer that contains the data product type

## Data_Product_Header.T:

Generic data_product packet for holding arbitrary data_product types

Table 12: Data_Product_Header Packed Record : 88 bits

| Name | Type | Range | Size (Bits) | Start Bit | End Bit |
|---|---|---|---|---|---|
| Time | Sys_Time.T | - | 64 | 0 | 63 |
| Id | Data_Product_Types.<br>Data_Product_Id | 0 to 65535 | 16 | 64 | 79 |
| Buffer_Length | Data_Product_<br>Types.Data_Product_<br>Buffer_Length_Type | 0 to 32 | 8 | 80 | 87 |

Field Descriptions:
- **Time** - The timestamp for the data product item.
- **Id** - The data product identifier
- **Buffer_Length** - The number of bytes used in the data product buffer

## Event.T:

Generic event packet for holding arbitrary events

Table 13: Event Packed Record : 344 bits *(maximum)*

| Name | Type | Range | Size (Bits) | Start Bit | End Bit | Variable Length |
|---|---|---|---|---|---|---|
| Header | Event_Header.T | - | 88 | 0 | 87 | – |
| Param_Buffer | Event_Types.<br>Parameter_<br>Buffer_Type | - | 256 | 88 | 343 | Header.Param_<br>Buffer_Length |

Field Descriptions:
- **Header** - The event header
- **Param_Buffer** - A buffer that contains the event parameters

## Event_Header.T:

Generic event packet for holding arbitrary events

Table 14: Event_Header Packed Record : 88 bits

| Name | Type | Range | Size (Bits) | Start Bit | End Bit |
|---|---|---|---|---|---|
| Time | Sys_Time.T | - | 64 | 0 | 63 |
| Id | Event_Types.Event_ Id | 0 to 65535 | 16 | 64 | 79 |
| Param_Buffer_Length | Event_Types. Parameter_Buffer_ Length_Type | 0 to 32 | 8 | 80 | 87 |

Field Descriptions:
- **Time** - The timestamp for the event.

- **Id** - The event identifier

- **Param_Buffer_Length** - The number of bytes used in the param buffer

## Invalid_Command_Info.T:

Record for holding information about an invalid command

Table 15: Invalid_Command_Info Packed Record : 112 bits

| Name | Type | Range | Size (Bits) | Start Bit | End Bit |
|---|---|---|---|---|---|
| Id | Command_Types. Command_Id | 0 to 65535 | 16 | 0 | 15 |
| Errant_Field_ Number | Interfaces. Unsigned_32 | 0 to 4294967295 | 32 | 16 | 47 |
| Errant_Field | Basic_Types.Poly_ Type | - | 64 | 48 | 111 |

Field Descriptions:
- **Id** - The command Id received.

- **Errant_Field_Number** - The field that was invalid. 1 is the first field, 0 means unkown field, 2**32 means that the length field of the command was invalid.

- **Errant_Field** - A polymorphic type containing the bad field data, or length when Errant_Field_Number is 2**32.

## Packed_U32.T:

Single component record for holding packed unsigned 32-bit value.

Table 16: Packed_U32 Packed Record : 32 bits

| Name | Type | Range | Size (Bits) | Start Bit | End Bit |
|---|---|---|---|---|---|
| Value | Interfaces. Unsigned_32 | 0 to 4294967295 | 32 | 0 | 31 |

Field Descriptions:
- **Value** - The 32-bit unsigned integer.

## Sys_Time.T:

A record which holds a time stamp using GPS format including seconds and subseconds since epoch (1-5-1980 to 1-6-1980 midnight).

Table 17: Sys_Time Packed Record : 64 bits

| Name | Type | Range | Size (Bits) | Start Bit | End Bit |
|------|------|-------|-------------|-----------|---------|
| Seconds | Interfaces. Unsigned_32 | 0 to 4294967295 | 32 | 0 | 31 |
| Subseconds | Interfaces. Unsigned_32 | 0 to 4294967295 | 32 | 32 | 63 |

Field Descriptions:
- **Seconds** - The number of seconds elapsed since epoch.
- **Subseconds** - The number of $1/(2^{32})$ sub-seconds.

## Tat_State.T:

The time at tone disable/enable state.

*Preamble (inline Ada definitions):*

```
1   type Tat_State_Type is (Disabled, Enabled);
2   for Tat_State_Type use (Disabled => 0, Enabled => 1);
```

Table 18: Tat_State Packed Record : 8 bits

| Name | Type | Range | Size (Bits) | Start Bit | End Bit |
|------|------|-------|-------------|-----------|---------|
| State | Tat_State_Type | 0 => Disabled<br>1 => Enabled | 8 | 0 | 7 |

Field Descriptions:
- **State** - Is time at tone enabled or disabled.

## Tick.T:

The tick datatype used for periodic scheduling. Included in this type is the Time associated with a tick and a count.

Table 19: Tick Packed Record : 96 bits

| Name | Type | Range | Size (Bits) | Start Bit | End Bit |
|------|------|-------|-------------|-----------|---------|
| Time | Sys_Time.T | - | 64 | 0 | 63 |
| Count | Interfaces. Unsigned_32 | 0 to 4294967295 | 32 | 64 | 95 |

Field Descriptions:
- **Time** - The timestamp associated with the tick.
- **Count** - The cycle number of the tick.

## 5.3 Enumerations

OfThe following section outlines any enumerations used in the component.

### Command_Enums.Command_Response_Status.E:

This status enumerations provides information on the success/failure of a command through the command response connector.

Table 20: Command_Response_Status Literals:

| Name | Value | Description |
| --- | --- | --- |
| Success | 0 | Command was passed to the handler and successfully executed. |
| Failure | 1 | Command was passed to the handler not successfully executed. |
| Id_Error | 2 | Command id was not valid. |
| Validation_Error | 3 | Command parameters were not successfully validated. |
| Length_Error | 4 | Command length was not correct. |
| Dropped | 5 | Command overflowed a component queue and was dropped. |
| Register | 6 | This status is used to register a command with the command routing system. |
| Register_Source | 7 | This status is used to register command sender's source id with the command router for command response forwarding. |