

# Ccsds Serial Interface

## *Component Design Document*

## 1 Description

This component is meant to be a backdoor serial component which uses Ada.Text\_IO to send and receive data over a serial port. On Linux, this will send/recv data to/from the terminal, but Ada.Text\_IO is attached to a diagnostic uart on most embedded systems. This means that this component can be used as a quick and dirty serial interface without implementing hardware specific uart drivers.

## 2 Requirements

No requirements have been specified for this component.

## 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** - *active*
- **Number of Connectors** - 4
- **Number of Invokee Connectors** - 1
- **Number of Invoker Connectors** - 3
- **Number of Generic Connectors** - *None*
- **Number of Generic Types** - *None*
- **Number of Unconstrained Arrayed Connectors** - *None*
- **Number of Commands** - *None*
- **Number of Parameters** - *None*
- **Number of Events** - 3
- **Number of Faults** - *None*
- **Number of Data Products** - *None*
- **Number of Data Dependencies** - *None*
- **Number of Packets** - *None*

## 3.2 Diagram

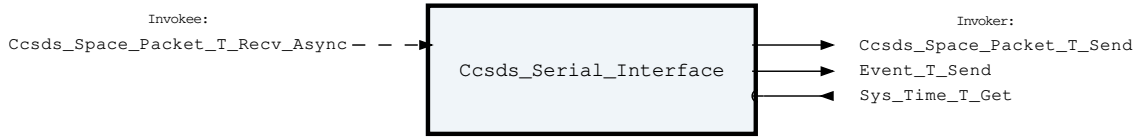


Figure 1: Ccsds Serial Interface component diagram.

## 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: Ccsds Serial Interface Invokee Connectors

Name	Kind	Type	Return_Type	Count
Ccsds_Space_Packet_T_Recv_Async	recv_async	Ccsds_Space_Packet.T	-	1

Connector Descriptions:

- **Ccsds\_Space\_Packet\_T\_Recv\_Async** - On this connector the Serial Interface Component receives data and sends it out of the serial port.

### 3.3.2 Internal Queue

This component contains an internal first-in-first-out (FIFO) queue to handle asynchronous messages. This queue is sized at initialization as a configurable number of bytes. Determining the size of the component queue can be difficult. The following table lists the connectors that will put asynchronous messages onto the queue, and the maximum sizes of each of those messages on the queue. Note that each message put onto the queue also incurs an overhead on the queue of 5 additional bytes, which is included in the max message size below:

Table 2: Ccsds Serial Interface Asynchronous Connectors

Name	Type	Max Size (bytes)
Ccsds_Space_Packet_T_Recv_Async	Ccsds_Space_Packet.T	1285

If you are unsure how to size the queue of this component, it is recommended that you make the queue size a multiple of the largest size found above.

### 3.3.3 Invoker Connectors

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

Table 3: Ccsds Serial Interface Invoker Connectors

Name	Kind	Type	Return_Type	Count
------	------	------	-------------	-------

Ccsds_Space_Packet_T_Send	send	Ccsds_Space_Packet.T	-	1
Event_T_Send	send	Event.T	-	1
Sys_Time_T_Get	get	-	Sys_Time.T	1

Connector Descriptions:

- **Ccsds\_Space\_Packet\_T\_Send** - On this connector the Serial Interface Component sends any data it received from the serial port.
- **Event\_T\_Send** - Events are sent out of this connector.
- **Sys\_Time\_T\_Get** - The system time is retrieved via this connector.

### 3.4 Initialization

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

#### 3.4.1 Component Subtask Instantiation

This component contains subtasks. Subtasks are distinct from the component's standard active or passive configuration. Subtasks must be initialized with their own stack, secondary stack, and execution priority during initialization. This component contains the following subtasks.

Component Subtasks:

- **Listener** - This internal task is used to listen on the serial port for incoming packets.

#### 3.4.2 Component Instantiation

This component contains no instantiation parameters in its discriminant.

#### 3.4.3 Component Base Initialization

This component achieves base class initialization using the `init_Base` subprogram. This subprogram requires the following parameters:

Table 4: Ccsds Serial Interface Base Initialization Parameters

Name	Type
Queue_Size	Natural

Parameter Descriptions:

- **Queue\_Size** - The number of bytes that can be stored in the component's internal queue.

#### 3.4.4 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 5: Ccsds Serial Interface Set Id Bases Parameters

Name	Type
Event_Id_Base	Event_Types.Event_Id_Base

---

Parameter Descriptions:

- **Event\_Id\_Base** - The value at which the component's event identifiers begin.

### 3.4.5 Component Map Data Dependencies

This component contains no data dependencies.

### 3.4.6 Component Implementation Initialization

The calling of this implementation class initialization procedure is mandatory. Init to provide gap between packets if necessary The `init` subprogram requires the following parameters:

Table 6: Ccsds Serial Interface Implementation Initialization Parameters

Name	Type	Default Value
Interpacket_Gap_Ms	Natural	0

Parameter Descriptions:

- **Interpacket\_Gap\_Ms** - Amount of time in milliseconds to wait in between transmission of each CCSDS packet. Some UART protocols rely on a gap to differentiate between packets, and this can be used to enforce that.

## 3.5 Events

Below is a list of the events for the Ccsds Serial Interface component.

Table 7: Ccsds Serial Interface Events

Local ID	Event Name	Parameter Type
0	Packet_Send_Failed	Ccsds_Primary_Header.T
1	Packet_Recv_Failed	Ccsds_Primary_Header.T
2	Have_Not_Seen_Sync_Pattern	Packed_U32.T

Event Descriptions:

- **Packet\_Send\_Failed** - Failed to send a packet over the serial port because it has an invalid CCSDS header.
- **Packet\_Recv\_Failed** - Failed to receive a packet over the serial port because it has an invalid CCSDS header.
- **Have\_Not\_Seen\_Sync\_Pattern** - The component has received N number of bytes without seeing a sync pattern yet.

## 4 Unit Tests

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

## 4.1 Ccsds\_Serial\_Interface\_Tests Test Suite

This is the packet send unit test suite for the Serial Interface Component

Test Descriptions:

- **Test\_Packet\_Receive** - This unit test makes sure that packets received through the serial port are forwarded through the send connector. This test exercises the additional internal task of the Serial Interface Component.

## 4.2 Ccsds\_Serial\_Interface\_Tests Test Suite

This is the packet send unit test suite for the Serial Interface Component

Test Descriptions:

- **Test\_Packet\_Send** - This unit test makes sure that packets sent through the component's queue are forwarded through the serial port.

# 5 Appendix

## 5.1 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..

### Ccsds\_Primary\_Header.T:

Record for the CCSDS Packet Primary Header *Preamble (inline Ada definitions)*:

```
1 subtype Three_Bit_Version_Type is Interfaces.Unsigned_8 range 0 .. 7;  
2 type Ccsds_Apid_Type is mod 2**11;  
3 type Ccsds_Sequence_Count_Type is mod 2**14;
```

Table 8: Ccsds\_Primary\_Header Packed Record : 48 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Version	Three_Bit_Version_Type	0 to 7	3	0	2
Packet_Type	Ccsds_Enums.Ccsds_Packet_Type.E	0 => Telemetry 1 => Telecommand	1	3	3

Secondary_Header	Ccsds_Enums.Ccsds_Secondary_Header_Indicator.E	0 => Secondary_Header_Not_Present 1 => Secondary_Header_Present	1	4	4
Apid	Ccsds_Apid_Type	0 to 2047	11	5	15
Sequence_Flag	Ccsds_Enums.Ccsds_Sequence_Flag.E	0 => Continuationsegment 1 => Firstsegment 2 => Lastsegment 3 => Unsegmented	2	16	17
Sequence_Count	Ccsds_Sequence_Count_Type	0 to 16383	14	18	31
Packet_Length	Interfaces.Unsigned_16	0 to 65535	16	32	47

Field Descriptions:

- **Version** - Packet Version Number
- **Packet\_Type** - Packet Type
- **Secondary\_Header** - Does packet have CCSDS secondary header
- **Apid** - Application process identifier
- **Sequence\_Flag** - Sequence Flag
- **Sequence\_Count** - Packet Sequence Count
- **Packet\_Length** - This is the packet data length. One added to this number corresponds to the number of bytes included in the data section of the CCSDS Space Packet.

### Ccsds\_Space\_Packet.T:

Record for the CCSDS Space Packet *Preamble (inline Ada definitions)*:

```

1 use Basic_Types;
2 subtype Ccsds_Data_Type is Byte_Array (0 ..
  ↳ Configuration.Ccsds_Packet_Buffer_Size - 1);

```

Table 9: Ccsds\_Space\_Packet Packed Record : 10240 bits (*maximum*)

Name	Type	Range	Size (Bits)	Start Bit	End Bit	Variable Length
------	------	-------	-------------	-----------	---------	-----------------

Header	Ccsds_ Primary_ Header.T	-	48	0	47	-
Data	Ccsds_Data_ Type	-	10192	48	10239	Header. Packet_Length

Field Descriptions:

- **Header** - The CCSDS Primary Header
- **Data** - User Data Field

### Event.T:

Generic event packet for holding arbitrary events

Table 10: 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. Parameter_ Buffer_Type	-	256	88	343	Header.Param_ 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 11: 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

### Packed\_U32.T:

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

Table 12: 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 13: 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.

## 5.2 Enumerations

The following section outlines any enumerations used in the component.

### Ccsds\_Enums.Ccsds\_Packet\_Type.E:

This single bit is used to identify that this is a Telecommand Packet or a Telemetry Packet. A Telemetry Packet has this bit set to value 0; therefore, for all Telecommand Packets Bit 3 shall be set to value 1.

Table 14: Ccsds\_Packet\_Type Literals:

Name	Value	Description
Telemetry	0	Indicates a telemetry packet
Telecommand	1	Indicates a telecommand packet

### Ccsds\_Enums.Ccsds\_Secondary\_Header\_Indicator.E:

This one bit flag signals the presence (Bit 4 = 1) or absence (Bit 4 = 0) of a Secondary Header data structure within the packet.

Table 15: Ccsds\_Secondary\_Header\_Indicator Literals:

Name	Value	Description
------	-------	-------------

Secondary_Header_Not_Present	0	Indicates that the secondary header is not present within the packet
Secondary_Header_Present	1	Indicates that the secondary header is present within the packet

### **Ccsds\_Enums.Ccsds\_Sequence\_Flag.E:**

This flag provides a method for defining whether this packet is a first, last, or intermediate component of a higher layer data structure.

Table 16: Ccsds\_Sequence\_Flag Literals:

<b>Name</b>	<b>Value</b>	<b>Description</b>
Continuationsegment	0	Continuation component of higher data structure
Firstsegment	1	First component of higher data structure
Lastsegment	2	Last component of higher data structure
Unsegmented	3	Standalone packet