VIRTEC TC Client User Guide

Version 2.0.beta6

Table of Contents

Overview

ISO 11783 (ISOBUS) consists of the following parts, under the general title *Tractors and machinery for agriculture and forestry - Serial control and communications data network*:

- *Part 1: General standard for mobile data communication*
- *Part 2: Physical layer*
- *Part 3: Data link layer*
- *Part 4: Network layer*
- *Part 5: Network management*

- *Part 6: Virtual Terminal*
- *Part 7: Implement messages application layer*
- *Part 8: Power train messages*
- *Part 9: Tractor ECU*
- **Part 10: Task controller and management information system data interchange**
- *Part 11: Mobile data element dictionary*
- *Part 12: Diagnostics services*
- *Part 13: File server*

The parts shown above in **bold** are included in the VIRTEC TC Client Library. The Task Controller Client Library allows applications to be built quickly which interface with a Task Controller server. An entire TC Client application can be written using the API functions outlined in the section `API Reference`.

Definitions

Boom
Physical arm to which sections are attached. In the TC Client Library, each channel is represented as a boom.

Channel
A control channel. In the DDOP, each channel is a separate boom.

DDI
Data Dictionary Identifier. Each DPD and DPT object in the DDOP must have a DDI. Information about DDIs can be found in the ISOBUS Data Dictionary found at http://dictionary.isobus.net/isobus/.

DDOP
Device Description Object Pool. The DDOP physically describes the implement, informs the TC server of which functionalities are supported by the implement, as well as which process data is available.

DET
Device Element Object. A type of object in the DDOP which contains information about a specific physical element of a device. Each individual element of the device will have its own DET object.

DPD
Device Process Data. A type of object in the DDOP which contains information the TC Server and TC Client need to interface with each other while a task is running.

DPT
Device Property Object. A type of object in the DDOP which contains a single device element property definition. This type of object is used to describe geometry and constant properties, such as the width of a section.

DVC
Device Object. A type of object in the DDOP which contains information about the entire device. Only one device object is allowed in a DDOP.

DVP
Device Value Presentation Object. A type of object in the DDOP which contains presentation information about how the value of a DPD or DPT is to be displayed to an operator.

Element Identifier / Element
A reference to a physical part of the implement. Examples include the device itself, a bin, a boom, a section, or a connector.

Functionalities
Task Controller Functionalities (TC-BAS, TC-SC, TC-GEO) identify the features supported. Both the TC Server and TC Client must support a feature for that feature to work fully and correctly.

Measurement Function
A Measurement Function is a developer defined function that can be assigned to a PDV using `TC_PDV_MeasurementFunction_Set()`, which will return an updated value for that PDV, each time the PDV is processed by The PDV List.

New Value Handler
A New Value Handler is a developer defined function that that can be assigned to a PDV using `TC_PDV_MeasurementFunction_Set()`. The assigned function will then be called, each time a new value for that PDV is received from the TC Server.

Object ID
Unique identifier for an object in the Device Description Object Pool (DDOP)

PDV
Process Data Variable. This variable contains a value that can be exchanged between the TC server and the TC client. Within the context of the TC Client Library, a PDV not only has a value, but also contains information about trigger methods needed to determine when a PDV Value needs to be sent to the TC Server. When the pool is generated, a PDV is created for each DPD in the DDOP and the created PDVs are added to The PDV List. The list is processed continuously, deciding when values need to be sent to the TC Server.

PDV List
The PDV List is an array of all of the active PDVs in an application. This list is generated when the DDOP is generated and is continuously processed by the library, deciding when values need to be sent to the TC Server. When Values are received from the TC Server, the list is updated.

TC-BAS (Basic)
Task Controllers with this functionality have the capability to process a task-based collection of totals. This includes keeping track of the total amount of input or output

resources, total distance, total time, and other totals. These totals are not necessarily location specific.

TC-GEO (Geo-Based)
Task Controllers with this functionality have the capability to process location specific values, based on a geographic position. This includes the ability to vary the rate of applications, and to provide as-applied process data for mapping purposes.

TC-SC (Section Control)
Task Controllers with this functionality have the capability to turn device elements ON or OFF based on geographic position. This includes minimizing application overlaps, skips, and applying outside of the application zone.

TC Client Guide

Setting up the TC Client

1. Run the *DDObjectPoolCreator.py* tool (included with TC Client) to convert the object pool .xml files into C code that the library can access.
    1. *NOTE:* ElementTree v1.2.7 is required and can be downloaded from http://effbot.org/zone/element-index.htm.
    2. When running the *DDObjectPoolCreator.py* tool include the object pool .xml file name that is desired to be converted.
2. Create the `TCClient_T` object using the `MAKE_TCClient_T()` macro.
    1. Add the name of your `Foundation_T` object as the `foundation_ptr` parameter of the `MAKE_TCClient_T()` macro.
    2. Add the name of your array of `TC_T` objects as the `tc_array` parameter of the `MAKE_TCClient_T()` macro.
    3. Add the supported options (refer to of the `MAKE_TCClient_T` definition for details) TC client into the `options` parameter of the `MAKE_TCClient_T()` macro.
    4. Add the number of supported booms, sections, and channels of the TC client into their corresponding `booms`, `sections`, and `channels` parameters of the `MAKE_TCClient_T()` macro.
    5. Add the desired TCClient priority into the `priority` parameter of the `MAKE_TCClient_T()` macro.

Example

```
TCClient_T MyApp_TCClient = MAKE_TCClient_T(&MyApp_Foundation, MyApp_TCs,
TC_BAS | TC_GEO_WITH_POSITION_BASED_CONTROL, 2, 10, 3, MY_MUTEX_PRIORITY);
```

Using the TC Client

Connecting to a TC

1. Find the next active TC by using the `TC_NextTC()` function.

2. You can then send your object pool to the TC.
    1. Send your object pool directly by using the `TC_SendObjectPool()` function.

Receiving Events

Registering a Developer Designed Measurement Callback

The `TC_PDV_MeasurementFunction_Set()` function is used to connect a measurement or new value handler function defined by the developer to a PDV. When a new value for the PDV is needed, the assigned measurement function will be called to calculate the new value.

1. Create a callback function to be associated to the callback structure.
2. Register the callback to an object.
    1. Add the name of your array of `TC_T` objects as the `tc` parameter of the `TC_PDV_MeasurementFunction_Set()` function.
    2. Place the object ID of the object into the `objectid` parameter of the `TC_PDV_MeasurementFunction_Set()` function.
    3. Place the callback function into the `func` parameter of the `TC_PDV_MeasurementFunction_Set()` function.
    4. Place any parameters required for the callback function into the `argument` parameter of the `TC_PDV_MeasurementFunction_Set()` function.

Example

```
typedef struct DeveloperDefined_S
{
   TC_PDV_Value_T start_value;
   TC_PDV_Value_T upper_limit;
   TC_PDV_Value_T rise_rate;
   Time_T  rise_interval;
} DeveloperDefined_T;

// Create callback function parameter
DeveloperDefined_T callback_parameter;

static TC_PDV_Value_T Callback_Function(TC_PDV_Value_T pdv_val, void
*void_parameter)
{
   ...
}

...

void MyApp_Init(void)
{
   // Initialize callback function parameter
   callback_parameter.start_value = (TC_PDV_Value_T)0;
```

```
    callback_parameter.rise_rate = (TC_PDV_Value_T)10;
    callback_parameter.rise_interval = milliseconds(50);
    callback_parameter.upper_limit = (TC_PDV_Value_T)6000;

    // Register callback
    (void)TC_PDV_MeasurementFunction_Set(MyApp_TC, ISO_OBJECTID_NUMBER,
Callback_Function, &callback_parameter);
}
```

Sending a Command

For object-related commands, the `PDV_Value_Set()` can be used to set values. The function can be used to set the value based solely on the object ID or the element number and DDI together.

Example

For object-related commands:

```
static uint16_t Value = 0;

...

Value++;

(void)PDV_Value_Set(MyApp_TC, ISO_OBJECTID_NUMBER, (TC_ElementNumber_T)0,
(TC_DDI_T)0, Value);

...
```

Other Things You Can Do

Disconnecting from a TC

Gracefully disconnect from the TC by using the `TC_Disconnect()` function. A graceful disconnect is the standard way of making sure the TC doesn't warn, alert, or complain to the operator that communication with your working set has been lost.

Example

```
bool_t disconnecting = TC_Disconnect(&MyApp_TCClient, MyApp_TC);
```

State Machine Example

Here is an example that incorporates a state machine to determine which of the above steps needs to be performed.

Example

```
typedef enum AppState_E
{
  WAIT_TC,
  CONNECT_TC,
  SEND_OP_ACTIVATE_TC,
  OPERATOR_INTERACTION_TC,
  DELETE_OP_TC,
  DISCONNECT_TC,
  DEMO_IDLE_TC
} AppState_T;

...

void MyApp_Init(void)
{
    MyApp_State = WAIT_TC;
}

...

void MyApp_Task(void)
{
  switch (MyApp_State)
  {
  case WAIT_TC:
    if (TC_NextTC(&MyApp_TCClient, &MyApp_TC))
    {
      MyApp_State = CONNECT_TC;
    }
    break;
  case CONNECT_TC:
    if (TC_SendObjectPool(&MyApp_TCClient, MyApp_TC, &MyApp_ObjectPool,
pdvlist, NUM_PDV))
    {
      MyApp_State = SEND_OP_ACTIVATE_TC;
    }
    break;
  case SEND_OP_ACTIVATE_TC:
    if (MyApp_TC->ObjectPool.State == TC_OP_OPERATOR_INTERACTION)
    {
      MyApp_State = OPERATOR_INTERACTION_TC;
    }
    else if (MyApp_TC->ObjectPool.State == TC_OP_IDLE)
    {
      MyApp_State = DELETE_OP_TC;
    }
    break;
  case OPERATOR_INTERACTION_TC:
    if (SoftwareTimer_Get(&DemoTC->Status.Timer) == 0)
    {
      MyApp_State = WAIT_TC;
    }
    else if (MyApp_TC->ObjectPool.State == TC_OP_IDLE)
    {
      MyApp_State = WAIT_TC;
    }
    break;
  case DELETE_OP_TC:
    if (TC_Disconnect(&MyApp_TCClient, MyApp_TC))
    {
      MyApp_State = WAIT_TC;
    }
    break;
  case DISCONNECT_TC:
    if (TC_Disconnect(&MyApp_TCClient, MyApp_TC))
    {
      MyApp_State = DEMO_IDLE_TC;
    }
    break;
  case DEMO_IDLE_TC:
  default:
    break;
  }
}
```

API Reference

TC Client API Reference

Data Types

TCClient_SupportedOptions_T: uint8_t
TCClient_CapableBooms_T: uint8_t
TCClient_CapableSections_T: uint8_t
TCClient_ControlChannels_T: uint8_t
TC_MetricsMessages_T: uint8_t
TC_Version_T: uint8_t
TC_MetricsRetries_T: uint8_t
TC_SubStatus_T: uint8_t
TC_Command_T: uint8_t
TC_Distance_T: int32_t
TC_Threshold_T: int32_t
TC_PDV_Value_T: int32_t
TC_TriggerMethod_T: uint8_t
DPD_Properties_T: uint8_t
TC_StringLength_T: uint8_t
TC_PD_ErrorCode_T: uint8_t
TC_NumberOfBytes_255Max_T: uint8_t

`TC_NumberOfDecimals_255Max_T`: `uint8_t`
`TC_StructureLabelByte_T`: `uint8_t`
`TC_LocalizationLabelByte_T`: `uint8_t`
`TC_ObjectID_T`: `uint16_t`
`TC_ElementNumber_T`: `uint16_t`
`TC_NumObjects_T`: `uint16_t`
`TC_DDI_T`: `uint16_t`
`TC_DevicePropertyValue_T`: `int32_t`
`TC_DVP_Offset_T`: `int32_t`
`TC_DPD_Properties_Bitmask_T`: `uint8_t`
`TC_DPD_TriggerMethodsBitmask_T`: `uint8_t`

Enumerations

Task_Status_T

Available Status states

Signature

```
typedef enum Task_Status_E Task_Status_T
```

Members

`TASK_RUNNING`
Task status: Running

`TASK_PAUSED`
Task status: Paused

`TASK_COMPLETED`
Task status: Complete

TC_DeviceElementType_T

Enumeration for TC device types

Signature

```
typedef enum TC_DeviceElementType_E TC_DeviceElementType_T
```

Members

`FIRST_DEVICE_TYPE`
Used only for looping over this enum type

`TC_DeviceElementType_Device`
Represents the complete device and makes it addressable for the task controller. DDOP shall have at least one

`TC_DeviceElementType_Function`
Can be used as a generic device element to define individually accessible components of a device like valves or sensors

`TC_DeviceElementType_Bin`
Example: the tank of a sprayer or the bin of a seeder

`TC_DeviceElementType_Section`
Example: section of a spray boom, seed toolbar or planter toolbar. A section may provide device geometry definitions(x, y, z) and a working width next to supported process data elements as device process variable values or device property values

`TC_DeviceElementType_Unit`
This device element type is, for example, used for spray boom nozzles, seeder openers or planter row units. It is intended as a layer below the device element type section in the hierarchical device element structure

`TC_DeviceElementType_Connector`
This device element type specifies the mounting/connection position of the device. More than one connector can be defined for one device(e.g.a tractor may provide front - end mounting and rear - end mounting connection locations).A connector element shall provide its device geometry definitions(x, y, z) relative to the device reference point as device process data values or as device property values, even when the device reference point is the same as the location of the connector(x = y = z = 0)

`TC_DeviceElementType_NavigationReference`
This device element type defines the navigation reference position for navigation devices such as GPS receivers.Such elements have to reference their position in the x - , y - , and z - direction as device process data values or device property values

`MAX_DEVICE_TYPES`
Used only for looping over this enumeration type

`TC_MetricsState_T`

Enumeration for TC Metrics state/storage types

Signature

```
typedef enum TC_MetricsState_E TC_MetricsState_T
```

Members

`TC_METRICS_IDLE`
Idle state

`TC_METRICS_SEND_REQUEST`
Request for TC Metrics

`TC_METRICS_REQUEST_SENT`
Request for TC Metrics sent

`TC_METRICS_WAIT_RESPONSE`
Waiting for TC Metrics response

`TC_ObjectPoolState_T`

Enumeration for TC Metrics state/storage types

Signature

```
typedef enum TC_ObjectPoolState_E TC_ObjectPoolState_T
```

Members

`TC_OP_IDLE`
Idle State

`TC_OP_WAIT_CONNECTED`
Wait for a connection to a TC to be established

`TC_OP_WAIT_METRICS`
Wait for all metrics to be gathered

`TC_OP_SEND_REQUEST_STRUCTURE_LABEL`
ECU sends Request Structure Label message to the TC

`TC_OP_REQUEST_STRUCTURE_LABEL_SENT`
Wait for ECU to send the Request Structure Label message

`TC_OP_WAIT_REQUEST_STRUCTURE_LABEL_RESPONSE`
Wait for Request Structure Label message from the TC

`TC_OP_SEND_OP_DELETE_WRONGVERSION`
ECU sends Object-pool Delete message because the OP structure labels do not match

`TC_OP_OP_DELETE_WRONGVERSION_SENT`
Wait for ECU to send the Object-pool Delete message to the TC

`TC_OP_WAIT_OP_DELETE_WRONGVERSION_RESPONSE`
Wait for Object-pool Delete Response message from the TC

`TC_OP_SEND_REQUEST_OP_TRANSFER`
ECU sends Request Object-pool Transfer message to TC

`TC_OP_REQUEST_OP_TRANSFER_SENT`
Wait for ECU to send Request Object-pool Transfer message

`TC_OP_WAIT_REQUEST_OP_TRANSFER_RESPONSE`
Wait for Request Object-Pool Transfer Response message from the TC

`TC_OP_SEND_OP_TRANSFER`
ECU sends Object-pool Transfer message to TC

`TC_OP_OP_TRANSFER_SENT`
Wait for Object-pool Transfer message to be sent

`TC_OP_WAIT_OP_TRANSFER_RESPONSE`
Wait for Object-pool Transfer Response message from TC

`TC_OP_SEND_OP_ACTIVATE`
Send Object-pool Activate message to TC

`TC_OP_OP_ACTIVATE_SENT`
Wait for Object-pool Activate message to be sent

`TC_OP_WAIT_OP_ACTIVATE_RESPONSE`
Wait for Object-pool Activate Response message from the TC

`TC_OP_OPERATOR_INTERACTION`
If connection is no longer wanted, go to TC_OP_SEND_DELETE_OP

`TC_OP_SEND_OP_DELETE`
Send Object-pool Delete message to TC

`TC_OP_OP_DELETE_SENT`
Wait for Object-pool Delete message to be sent

`TC_OP_WAIT_OP_DELETE_RESPONSE`
Wait for Object-pool Delete Response message from the TC

Structures

`ActiveTCList_T`

Structure containing state information for all active TCs

Signature

```
typedef struct ActiveTCList_S ActiveTCList_T
```

Members

`Mutex_T Mutex`
Mutex to copy

`const Size_T  MaxTCs`
Maximum number of active TCs

`TC_T  * const List`
Pointer to array of transport sessions

`DeviceElementObject_T`

Object definition of the XML element DeviceElement

Signature

```
typedef struct DeviceElementObject_S DeviceElementObject_T
```

Members

`char table_id[3]`
XML element namespace for device (DET for DeviceElementObject)

`TC_ObjectID_T object_id`
Unique object identifier

`TC_DeviceElementType_T device_element_type`
Device element type

`TC_NumberOfBytes_255Max_T number_of_designator_bytes`
Length of following designator UTF-8 string

`char device_element_designator[32]`
Descriptive text to identify this device element

`TC_ElementNumber_T device_element_number`
Element number for process data variable addressing

`TC_ObjectID_T parent_object_id`
Object ID of parent

`TC_NumObjects_T number_of_objects_to_follow`
Number of following object references

`TC_ObjectID_T *child_object_id`
Child object ID, List of references to DeviceProcessDataObjects or DevicePropertyObjects. Referable child objects: DeviceProcessDataObject OR DevicePropertyObject

DeviceObject_T

DeviceObject is the object definition of the XML element device. Each device may have only a single DeviceObject in its Device description object pool

Signature

```
typedef struct DeviceObject_S DeviceObject_T
```

Members

`char table_id[3]`
XML element namespace for device (DVC for DeviceObject)

`TC_ObjectID_T object_id`
Unique object identifier inside this object pool

`TC_NumberOfBytes_255Max_T number_of_designator_bytes`
Length of following designator UTF-8 string

`char device_designator[32]`
Descriptive text to identify this device

`TC_NumberOfBytes_255Max_T number_of_software_version_bytes`
Length of following software version UTF-8 string

`char device_software_version[32]`
Software version indicating text

`ISOBUS_PacketData_T working_set_master_NAME[8]`
Name of working-set master

`TC_NumberOfBytes_255Max_T number_of_device_serial_number_bytes`
Length of following DeviceSerialNumber UTF-8 string

`char device_serial_number[32]`
Device and manufacture-specific serial number

`TC_StructureLabelByte_T device_structure_label[7]`
Label given by device to identify the device description structure

`TC_LocalizationLabelByte_T device_localization_label[7]`
Label given by device to identify the device description localization

`DeviceProcessDataObject_T`

Object definition of the XML element DeviceProcessData

Signature

```
typedef struct DeviceProcessDataObject_S DeviceProcessDataObject_T
```

Members

`char table_id[3]`
XML element namespace for device (DPD for DeviceProcessDataObject)

`TC_ObjectID_T object_id`
Unique object identifier

`TC_DDI_T process_data_ddi`
Identifier of process data variable

`TC_DPD_Properties_Bitmask_T process_data_properties`
Process data properties

`TC_DPD_TriggerMethodsBitmask_T process_data_available_trigger_methods`
Process data available trigger methods

`TC_NumberOfBytes_255Max_T number_of_designator_bytes`
Length of following designator UTF-8 string

`char process_data_designator[32]`
Descriptive text for this device process data

`TC_ObjectID_T device_value_presentation_object_id`
Object identifier of DeviceValuePresentationObject

`DevicePropertyObject_T`

Object definition of the XML element DeviceProperty

Signature

```
typedef struct DevicePropertyObject_S DevicePropertyObject_T
```

Members

`char table_id[3]`
XML element namespace for device (DPT for DevicePropertyObject)

`TC_ObjectID_T object_id`
Unique object identifier

`TC_DDI_T process_data_ddi`
Identifier of process data variable

`TC_DevicePropertyValue_T  property_value`
Value of property

`TC_NumberOfBytes_255Max_T number_of_designator_bytes`
Length of following designator UTF-8 string

`char property_designator[32]`
Descriptive text for this device property

`TC_ObjectID_T device_value_presentation_object_ID]`
Object identifier of DeviceValuePresentationObject

`DeviceValuePresentationObject_T`

Object definition of the XML element DeviceValuePresentation

Signature

```
typedef struct DeviceValuePresentationObject_S
DeviceValuePresentationObject_T
```

Members

`char table_id[3]`
XML element namespace for device (DVP for DeviceValuePresentation)

`TC_ObjectID_T object_id`
Unique object identifier

`TC_DVP_Offset_T offset`
Offset to be applied to the value for presentation

`float scale`
Scale to be applied to the value for presentation

`TC_NumberOfDecimals_255Max_T number_of_decimals`
Specify number of decimals to display after the decimal point

`TC_NumberOfBytes_255Max_T number_of_designator_bytes`
Length of following unit designator UTF-8 string

`char unit_designator[32]`
Unit designator for this value presentation

`Task_T`

Holds information about the task

Signature

```
typedef struct Task_S Task_T
```

Members

`Task_Status_T Status`
Status of task

`TC_Connection_T`

Structure for Working Set Task

Signature

```
typedef struct TC_Connection_S TC_Connection_T
```

Members

`SoftwareTimer_T Timer`
Timer tracking when to send Working Set Task message

`bool_t IsConnected`
Indicates whether the connection is active

`bool_t ConnectionWanted`
Indicates whether the application wants a connection

`TC_Language_T`

Information for fields of Language Command message

Signature

```
typedef struct TC_Language_S TC_Language_T
```

Members

`char LanguageCode[2]`
Two character string country codes in accordance with ISO 639

`DecimalSymbol_T DecimalSymbol`
Character used for decimal

`TimeFormat_T TimeFormat`
Time format

`DateFormat_T DateFormat`
Date format

`UnitsOfMeasure_T DistanceUnits`
Units used for Distance

`UnitsOfMeasure_T AreaUnits`
Units used for Area

`UnitsOfMeasure_T VolumeUnits`
Units used for Volume

`UnitsOfMeasure_T MassUnits`
Units used for Mass

`UnitsOfMeasure_T TemperatureUnits`
Units used for Temperature

`UnitsOfMeasure_T PressureUnits`
Units used for Pressure

`UnitsOfMeasure_T ForceUnits`
Units used for Force

`UnitsOfMeasure_T UnitsSystem`
General units system to use

TC_Measurement_T

Structure TC Measurement

Signature

```
typedef struct TC_Measurement_S TC_Measurement_T
```

Members

`TC_Meas_Func_T Function`
Pointer to function that will be used to measure

`void *Argument`
Pointer to the argument that will be passed to the measurement function

`TC_Metrics_T`

Structure to store TC metrics

Signature

```
typedef struct TC_Metrics_S TC_Metrics_T
```

Members

`TC_Language_T Language`
Localization Information

`TC_Version_T Version`
TC Version

`TC_MetricsState_T State`
Indicates whether Metrics are needed

`TC_MetricsMessages_T ExpectedMetrics`
Bit Mask Indicating Supported VT messaging

`TC_MetricsMessages_T ReceivedMetrics`
Bit Mask Indicating Received VT messaging

`TC_MetricsMessages_T CurrentMetric`
Bit Mask Indicating current message we're waiting for

`TC_MetricsRetries_T Retries`
Tracks how many times a particular metric has been requested

`SoftwareTimer_T ResponseTimer`
Tracks Metrics Response Timeouts

`TC_ObjectPool_T`

Structure for Object Pool status

Signature

```
typedef struct TC_ObjectPool_S TC_ObjectPool_T
```

Members

`TC_ObjectPoolState_T State`
State of the object pool

`SoftwareTimer_T ResponseTimer`
Object Pool Response Timer

`TC_PDVList_T PDVList`
TC PDV List

SoftwareTimer_T PDVListTimer
TC PDV List Timer

TC_ObjectPoolParts_T *Pool
Pointer to the structure containing object pool parts

Size_T Part
Object Pool part currently being loaded

Pipe_WriteHandle_T WriteHandle
Pipe handle to write raw pool data into

TC_ObjectPoolPart_T

Stores one part of an object pool. Object Pool information

Signature

```
typedef struct TC_ObjectPoolPart_S TC_ObjectPoolPart_T
```

Members

MemoryPointer_T Data
Object Pool Data

Size_T Size
Size of Object Pool Data

TC_ObjectPoolParts_T

Combines Object Pool Parts into a complete Object Pool

Signature

```
typedef struct TC_ObjectPoolParts_S TC_ObjectPoolParts_T
```

Members

TC_ObjectPoolPart_T *Parts
Data for each part of the object pool

Size_T NumParts
Number of parts to the object pool

char Structure_Label[7]
App's object pool version string

TC_PDV_T

Information for Process Data Variable

Signature

```
typedef struct TC_PDV_S TC_PDV_T
```

Members

`TC_ObjectID_T ObjectID`
Unique object identifier

`TC_ElementNumber_T ElementNumber`
Number associated with the Element

`TC_DDI_T DDI`
Identifier for DDI

`DPD_Properties_T Properties`
Properties: Select default/settable

`TC_TriggerMethod_T Triggermethod`
Trigger method for sending

`Time_T SendInterval`
Interval for sending

`TC_Distance_T DistanceInterval`
PDV Distance Interval

`TC_Distance_T LastSentAtDistance`
Last Sent Distance

`SoftwareTimer_T SendTimer`
PDV Send Timer

`TC_Threshold_T MaxThreshold`
PDV Max Threshold

`TC_Threshold_T MinThreshold`
PDV Min Threshold

`TC_Threshold_T ChangeThreshold`
PDV Change Threshold

`bool_t ValueChangedFlag`
Flag indicating Value Changed

`TC_Measurement_T Measurement`
Measurement structure

`TC_PDV_Value_T PDV_Value`
Value of PDV

`struct TC_PDV_S *NextInList`
Pointer to next PDV in the list

`TC_PDVList_T`

Structure Containing list of PDVs

Signature

```
typedef struct TC_PDVList_S TC_PDVList_T
```

Members

`Mutex_T Mutex`
Mutex for TC_PDV

`Time_T Period`
Period for TC_PDV

`bool_t SendDefaults`
Select to send defaults

`struct TC_PDV_S *StartOfList`
Structure holding information about current Process Data

`TC_Status_T`

Structure for TC Status message

Signature

```
typedef struct TC_Status_S TC_Status_T
```

Members

`TC_ElementNumber_T ElementNumber`
Element number, set to not available

`TC_DDI_T DDI`
DDI, set to not available

`TC_SubStatus_T ActualStatus`
Actual task controller status

`TC_SubStatus_T PreviousStatus`
Previous status byte. (For status transition detection)

`bool_t Busy`
Flag to indicate if the TC is busy

`bool_t OutOfMemory`
Flag to indicate if the TC is out of memory

`bool_t TotalsActive`
Flag to indicate if totals are active

`SourceAddress_T WorkingSetMasterSA`
SA of working-set master, if TC is executing a B.3 command. Else, 0

`TC_Command_T CommandCurrentlyRunning`
Command being executed, if TC is executing a B.3 command. Else, 0

`SoftwareTimer_T Timer`
Timer tracking when to consider TC connection has dropped

`NameTableIndex_T Source`
Source address of the TC

`TC_T`

Stores known data for the TC

Signature

```
typedef struct TC_S TC_T
```

Members

`Mutex_T Mutex`
Mutex for TC data

`TC_Status_T Status`
Data for the TC Status messages (ISO 11783-10 Annex B)

`TC_Connection_T Connection`
Data for TC connection management

`TC_Metrics_T Metrics`
Data for the various Technical Data messages (ISO 11783-6 Annex D)

`TC_ObjectPool_T ObjectPool`
Loaded Object Pool

`TC_Distance_T Distance`
Distance (in Millimeters)

`TCClient_Capabilities_T`

Structure to hold TCClient's capabilities

Signature

```
typedef struct TCClient_Capabilities_S TCClient_Capabilities_T
```

Members

`TCClient_SupportedOptions_T SupportedOptions`
ISOBUS 11783-10 Supported Options

`TCClient_CapableBooms_T NumberOfCapableSectionControl_Booms`
Number of Capable Section Control - Booms

`TCClient_CapableSections_T NumberOfCapableSectionControl_Sections`
Number of Capable Section Control - Sections

`TCClient_ControlChannels_T`
`NumberOfCapablePostionBasedControl_ControlChannels`
Number of Capable Position Based Control - Control Channels

`TCClient_T`

Contains all Foundation Functionality information for an ISOBUS App

Signature

```
typedef struct TCClient_S TCClient_T
```

Members

`Foundation_T *FoundationPtr`
Pointer to the foundation layer

`LanguageCallback_T LanguageCallback`
Language Command Handler

`ActiveTCList_T ActiveTCs`
Active TC List (There should only be one active TC on the network according to current ISOBUS 11783)

`struct Transport_MessageHandler_Node_S TCClient_MessageHandler_Node`
Structure for registering a message handler for the TC to ECU message

`struct Request_Node_S Request_ProcessData_Node`
Structure for registering a request handler for the Process Data message

`TCClient_Capabilities_T TCClient_Capabilities`
Structure holding capabilities of the TCClient

Macros

`MAKE_ActiveTCList_T()`

Macro used to initialize an `ActiveTCList_T`

Signature

```
MAKE_ActiveTCList_T(tc_array, priority)
```

Parameters

`tc_array`
Name of the array of Active TCs

`priority`
Maximum task priority accessing this list

`MAKE_Functionalities_T__TaskController_Basic_WorkingSet()`

Initializes the `Functionalities_T` structure for a Task Controller Basic Working Set

Signature

```
MAKE_Functionalities_T__TaskController_Basic_WorkingSet()
```

Parameters

None

`MAKE_Functionalities_T__TaskController_Geo_WorkingSet()`

Initializes the `Functionalities_T` structure for a Task Controller Geo Working Set

Signature

```
MAKE_Functionalities_T__TaskController_Geo_WorkingSet(number_of_channels,
polygon_prescription_maps)
```

Parameters

`number_of_channels`
Number of supported channels

`polygon_prescription_maps`
Supports polygon prescription maps (0=no, 1=yes)

`MAKE_Functionalities_T__TaskController_SectionControl_WorkingSet()`

Initializes the `Functionalities_T` structure for a Task Controller Section Control Working Set

Signature

```
MAKE_Functionalities_T__TaskController_SectionControl_WorkingSet(number_of_bo
oms, number_of_sections)
```

Parameters

`number_of_booms`
Number of supported booms

`number_of_sections`
Number of supported sections

`MAKE_TC_ObjectPoolPart_T()`

Macro used to make `ObjectPoolPart_T`

Signature

```
MAKE_TC_ObjectPoolPart_T(object_pool)
```

Parameters

`object_pool`
Applicable object pool

`MAKE_TC_ObjectPoolParts_T()`

Macro used to make `ObjectPoolParts_T`

Signature

```
MAKE_TC_ObjectPoolParts_T(object_pool_parts, version)
```

Parameters

`object_pool_parts`
Applicable `ObjectPoolParts_T`

`version`
Application version

`MAKE_TCCapabilities_T()`

Macro for initializing `TC_Capabilities_T` structure

Signature

```
MAKE_TCCapabilities_T(options, booms, sections, channels)
```

Parameters

`options`
ISOBUS 11783-10 Supported Options

`booms`
Number of Capable Section Control - Booms

`sections`
Number of Capable Section Control - Sections

`channels`
Number of Capable Position Based Control - Control Channels

`MAKE_TCClient_T()`

This macro initializes the `TCClient_T` structure and allows the user to input information regarding the capabilities of their Task Controller. The `options` parameter is available for the user to inform the client what options are supported by using the following bitwise #defines: * TC_BAS * Supported Option: TC-BAS functionality, documentation, and task totals * TC_GEO_WITHOUT_POSITION_BASED_CONTROL * Supported Option: TC-GEO functionality without position-based control * TC_GEO_WITH_POSITION_BASED_CONTROL

* Supported Option: TC-GEO functionality with position-based control * TC_SC * Supported Option: TC-SC functionality

Additionally the `booms`, `sections`, and `channels` parameters allow the user to input the number of capable units available for each.

Signature

```
MAKE_TCClient_T(foundation_ptr, tc_array, options, booms, sections, channels,
priority)
```

Parameters

`foundation_ptr`
Pointer to the corresponding `Foundation_T` structure

`tc_array`
Name of TC_T array

`options`
ISOBUS 11783-10 Supported Options

`booms`
Number of Capable Section Control - Booms

`sections`
Number of Capable Section Control - Sections

`channels`
Number of Capable Position Based Control - Control Channels

`priority`
Maximum task priority that accesses this structure

Functions

`PDV_Value_Get()`

Public wrapper around PDV get function

Signature

```
bool_t PDV_Value_Get(TC_T *tc, TC_ObjectID_T objectid, TC_PDV_Value_T
*pdv_value)
```

Parameters

`TC_T *tc`
Pointer to the active TC data structure

`TC_ObjectID_T objectid`
Object ID of PDV. If unknown, use ISO_OBJECTID_NULL

`TC_PDV_Value_T *pdv_value`
Pointer to the PDV value to update

Returns

`bool_t`
`TRUE` pdv_value is updated
`FALSE` `object_id` not in the list (pdv_value unchanged)

`PDV_Value_Set()`

Public wrapper around PDV Search functions

Signature

```
bool_t PDV_Value_Set(TC_T *tc, TC_ObjectID_T objectid, TC_ElementNumber_T
element, TC_DDI_T ddi, TC_PDV_Value_T newvalue)
```

Parameters

`TC_T *tc`
Pointer to the active TC data structure

`TC_ObjectID_T objectid`
Object ID of PDV. If unknown, use ISO_OBJECTID_NULL

`TC_ElementNumber_T element`
Element# of PDV. NOTE: Not used if objectid is specified.

`TC_DDI_T ddi`
DDI of PDV. NOTE: Not used if objectid is specified.

`TC_PDV_Value_T newvalue`
New PDV Value

Returns

`bool_t`
`TRUE` if the PDV was found and the value was updated
`FALSE` if the PDV was not found in the PDV list

`TC_ChangeDesignatorResponse_MsgHandler()`

TC response to Change Designator message

Signature

```
void TC_ChangeDesignatorResponse_MsgHandler(ISOBUS_Message_T *message, TC_T
*tc)
```

Parameters

`ISOBUS_Message_T *message`
Pointer to the received TC Process Data Message

`TC_T *tc`
Pointer to the application's active TC data structure

Returns

`void`

`TC_Disconnect()`

Disconnects from a TC

Signature

```
bool_t TC_Disconnect(TCClient_T *tcclient, TC_T *tc)
```

Parameters

`const TCClient_T *tcclient`
TC Client data structure containing all active TCs

`TC_T *tc`
TC to connect to

Returns

`bool_t`
`TRUE` Disconnection started
`FALSE` Disconnection not started

`TC_LocalizationLabel_MsgHandler()`

Processes a Localization Label message from the TC

Signature

```
void TC_LocalizationLabel_MsgHandler(ISOBUS_Message_T *message, TC_T *tc)
```

Parameters

`ISOBUS_Message_T *message`
Pointer to the received TC Process Data Message

`TC_T *tc`
Pointer to Task Controller data structure

Returns

`void`

`TC_MeasurementCommand_ChangeThreshold_MsgHandler()`

TC message specifies the change threshold value for a DDI. ECU should send the value if it changes by the amount specified by the TC

Signature

```
void TC_MeasurementCommand_ChangeThreshold_MsgHandler(ISOBUS_Message_T
*message, const TCClient_T *tcclient, TC_T *tc)
```

Parameters

`ISOBUS_Message_T *message`
Pointer to the received TC Process Data Message

`const TCClient_T *tcclient`
TC Client structure containing all active TC's

`TC_T *tc`
Pointer to the application's active TC data structure

Returns

```
void
```

`TC_MeasurementCommand_DistanceBased_MsgHandler()`

TC message specifies the distance interval at which ECU is to send values

Signature

```
void TC_MeasurementCommand_DistanceBased_MsgHandler(ISOBUS_Message_T
*message, const TCClient_T *tcclient, TC_T *tc)
```

Parameters

`ISOBUS_Message_T *message`
Pointer to the received TC Process Data Message

`const TCClient_T *tcclient`
TC Client structure containing all active TC's

`TC_T *tc`
Pointer to the application's active TC data structure

Returns

```
void
```

`TC_MeasurementCommand_MaxThreshold_MsgHandler()`

TC message specifies the change threshold value for a DDI. ECU should send the value if it changes by the amount specified by the TC

Signature

```
void TC_MeasurementCommand_MaxThreshold_MsgHandler(ISOBUS_Message_T *message,
const TCClient_T *tcclient, TC_T *tc)
```

Parameters

`ISOBUS_Message_T *message`
Pointer to the received TC Process Data Message

`const TCClient_T *tcclient`
TC Client structure containing all active TC's

`TC_T *tc`
Pointer to the application's active TC data structure

Returns

`void`

`TC_MeasurementCommand_MinThreshold_MsgHandler()`

TC message specifies the min threshold value for a DDI. ECU should send the value if it drops below this threshold

Signature

```
void TC_MeasurementCommand_MinThreshold_MsgHandler(ISOBUS_Message_T *message,
const TCClient_T *tcclient, TC_T *tc)
```

Parameters

`ISOBUS_Message_T *message`
Pointer to the received TC Process Data Message

`const TCClient_T *tcclient`
TC Client structure containing all active TC's

`TC_T *tc`
Pointer to the application's active TC data structure

Returns

`void`

`TC_MeasurementCommand_TimeBased_MsgHandler()`

TC message specifies the time interval at which ECU is to send values

Signature

```
void TC_MeasurementCommand_TimeBased_MsgHandler(ISOBUS_Message_T *message,
const TCClient_T *tcclient, TC_T *tc)
```

Parameters

`ISOBUS_Message_T *message`
Pointer to the received TC Process Data Message

`const TCClient_T *tcclient`
TC Client structure containing all active TC's

`TC_T *tc`
Pointer to the application's active TC data structure

Returns

`void`

`TC_NextTC()`

Finds the next active TC in the list of discovered TCs on the bus.

Signature

```
bool_t TC_NextTC(const TCClient_T *tcclient, TC_T **tc)
```

Parameters

`const TCClient_T *tcclient`
TC Client data structure containing all active TCs

`TC_T **tc`
TC to connect to

Returns

`bool_t`
`TRUE` TC found (and **tc populated)
`FALSE` TC not found

`TC_ObjectPoolTimer_Init()`

Initialization function for the PDV software timers

Signature

```
void TC_ObjectPoolTimer_Init(TC_PDV_T *pdv_list, Size_T number_of_pdvs)
```

Parameters

`TC_PDV_T *pdv_list`
List of PDV's to initialize timers for

`Size_T number_of_pdvs`
Number of PDV's

Returns

void

`TC_PDNACK_MsgHandler()`

Sends PDNACK to TC if data request is invalid

Signature

```
void TC_PDNACK_MsgHandler(ISOBUS_Message_T *message, const TCClient_T
*tcclient, const TC_T *tc)
```

Parameters

`ISOBUS_Message_T *message`
Message to handle

`const TCClient_T *tcclient`
TC Client structure containing all active TCs

`const TC_T *tc`
Pointer to the app's TC_T data structure

Returns

void

`TC_PDV_List_Search()`

Searches PDV list

Signature

```
bool_t TC_PDV_List_Search(TC_T *tc, TC_ObjectID_T objectid,
TC_ElementNumber_T elementnumber, TC_DDI_T ddi, TC_PDV_T **result)
```

Parameters

`TC_T *tc`
TC to connect to

`TC_ObjectID_T objectid`
Object ID of PDV to assign a measurement to

`TC_ElementNumber_T elementnumber`
Element# of the PDV

`TC_DDI_T ddi`
DDI of the PDV

`TC_PDV_T **result`
Pointer to the found PDV

Returns

`bool_t`
`TRUE` if the PDV was found
`FALSE` if the PDV was not found

`TC_PDV_MeasurementFunction_Clear()`

Clears any assigned measurement function from a PDV

Signature

```
bool_t TC_PDV_MeasurementFunction_Clear(TC_T *tc, TC_ObjectID_T objectid)
```

Parameters

`TC_T *tc`
TC to connect to

`TC_ObjectID_T objectid`
Object ID of PDV to assign a measurement to

Returns

`bool_t`
`TRUE` if the function was cleared from the PDV
`FALSE` if the function was not cleared

`TC_PDV_MeasurementFunction_Set()`

Assigns a user-defined measurement function to a PDV

Signature

```
bool_t TC_PDV_MeasurementFunction_Set(TC_T *tc, TC_ObjectID_T objectid,
TC_Meas_Func_T func, void *argument)
```

Parameters

`TC_T *tc`
TC to connect to

`TC_ObjectID_T objectid`
Object ID of PDV to assign a measurement to

`TC_Meas_Func_T func`
Name of the function to assign

`void *argument`
Pointer to the data to be used as an argument everytime the PDV list processor calls the measurement function

Returns

`bool_t`
`TRUE` if the function was assigned to the PDV
`FALSE` if the function was not assigned because the PDV was not found in the PDV list

`TC_Preset_Packet()`

Function to setup the Header for an ECU to TC packet

Signature

```
void TC_Preset_Packet(ISOBUS_Packet_T *packet, const TCClient_T *tcclient,
const TC_T *tc)
```

Parameters

`ISOBUS_Packet_T *packet`
Pointer to a TxPacket data structure

`const TCClient_T *tcclient`
TC Client structure containing all active TCs

`const TC_T *tc`
Pointer to the app's TC_T data structure

Returns

`void`

`TC_Send_ChangeDesignator_Msg()`

Function sends Change Designator message to Task Controller

Signature

```
bool_t TC_Send_ChangeDesignator_Msg(const TCClient_T *tcclient, const TC_T
*tc, TC_ObjectID_T object_id, Size_T string_length, const char *designator)
```

Parameters

`const TCClient_T *tcclient`
TC Client structure containing all active TCs

`const TC_T *tc`
Pointer to the application's active TC data structure

`TC_ObjectID_T object_id`
Object ID of the object to update

`Size_T string_length`
Total number of bytes in the string to transfer (32 bytes max)

`const char *designator`
Pointer to a string

Returns

`bool_t`
`TRUE` if the message was queued to be sent
`FALSE` if the message was not queued

`TC_Send_PDNACK_Msg()`

Function sends Process Data Negative Acknowledge (PDNACK) message to task controller

Signature

```
bool_t TC_Send_PDNACK_Msg(const TCClient_T *tcclient, const TC_T *tc,
TC_ElementNumber_T element_number, TC_DDI_T ddi, TC_PD_ErrorCode_T
pd_error_codes)
```

Parameters

`const TCClient_T *tcclient`
TC Client structure containing all active TCs

`const TC_T *tc`
Pointer to the application's active TC data structure

`TC_ElementNumber_T element_number`
Element Number. Pass ELEMENT_NUMBER_NA if Element Number does not pertain to the error code

`TC_DDI_T ddi`
Data Dictionary Identifier. Pass DDI_NA if DDI does not pertain to the error code

`TC_PD_ErrorCode_T pd_error_codes`
Process Data Error Codes (0 = no errors)
Bit 0 = 1 = Process Data Command not supported
Bit 1 = 1 = Invalid element number
Bit 2 = 1 = DDI not supported by element
Bit 3 = 1 = Trigger method not supported
Bit 4 = 1 = Process data not setable
Bit 5 = 1 = Invalid or unsupported interval or threshold
Bit 6 = 0 = Reserved (set to zero)
Bit 7 = 0 = Reserved (set to zero)

Returns

`bool_t`
`TRUE` if the message was queued to be sent
`FALSE` if the message was not queued

`TC_Send_RequestLocalization_Msg()`

Function sends Request Localization Label message to Task Controller

Signature

`bool_t TC_Send_RequestLocalization_Msg(const TCClient_T *tcclient, const TC_T *tc, const ISOBUS_Callback_T *callback)`

Parameters

`const TCClient_T *tcclient`
TC Client structure containing all active TCs

`const TC_T *tc`
Pointer to the application's active TC data structure

`const ISOBUS_Callback_T *callback`
Message callback

Returns

`bool_t`
`TRUE` Message was successfully sent
`FALSE` Message send failed (try later)

`TC_Send_RequestVersion_Msg()`

Sends Request Version message to the TC

Signature

`bool_t TC_Send_RequestVersion_Msg(const TCClient_T *tcclient, const TC_T *tc, const ISOBUS_Callback_T *callback)`

Parameters

`const TCClient_T *tcclient`
TC Client structure containing all active TCs

`const TC_T *tc`
Pointer to the application's active TC data structure

`const ISOBUS_Callback_T *callback`
Message callback

Returns

`bool_t`
`TRUE` if the message was queued to be sent
`FALSE` if the message was not queued

`TC_Send_ValueCommand_Msg()`

TCClient sends requested value to the TC

Signature

```
bool_t TC_Send_ValueCommand_Msg(const TCClient_T *tcclient, const TC_T *tc,
const TC_PDV_T *PDV)
```

Parameters

`const TCClient_T *tcclient`
TC Client structure containing all active TCs

`const TC_T *tc`
Pointer to the application's active TC data structure

`const TC_PDV_T *PDV`
Value to set to

Returns

`bool_t`
`TRUE` if the message was queued to be sent
`FALSE` if the message was not queued

`TC_Send_Version_Msg()`

Function sends out Version message

Signature

```
bool_t TC_Send_Version_Msg(const TCClient_T *tcclient, const TC_T *tc)
```

Parameters

`const TCClient_T *tcclient`
TC Client structure containing all active TCs

`const TC_T *tc`
Pointer to the application's active TC data structure

Returns

`bool_t`
`TRUE` if the message was queued to be sent
`FALSE` if the message was not queued

`TC_Send_WSTask_Msg()`

Sends Working-Set Task message

Signature

```
bool_t TC_Send_WSTask_Msg(const TCClient_T *tcclient, const TC_T *tc)
```

Parameters

`const TCClient_T *tcclient`
TC Client structure containing all active TCs

`const TC_T *tc`
Pointer to the application's active TC data structure

Returns

`bool_t`
`TRUE` if the message was sent
`FALSE` if the message was not sent

`TC_SendObjectPool()`

Sends an object pool to the TC

Signature

```
bool_t TC_SendObjectPool(const TCClient_T *tcclient, TC_T *tc,
TC_ObjectPoolParts_T *object_pool_parts, TC_PDV_T *firstpdv, Size_T numpdvs)
```

Parameters

`const TCClient_T *tcclient`
TC Client data structure containing all active TCs

`TC_T *tc`
TC to connect to

`TC_ObjectPoolParts_T *object_pool_parts`
Object Pool to send

`TC_PDV_T *firstpdv`
Pointer to the first PDV

`Size_T numpdvs`
Number of PDV's

Returns

`bool_t`
`TRUE` Object pool was sent
`FALSE` Object pool failed to send

`TC_Set_Distance()`

Sets distance value

Signature

```
void TC_Set_Distance(TC_T *tc, TC_Distance_T distance)
```

Parameters

`TC_T *tc`
Pointer to TC data structure

`TC_Distance_T distance`
Distance value to set to

Returns

`void`

`TC_Start_Connection()`

Starts a connection to a TC

Signature

```
bool_t TC_Start_Connection(const TCClient_T *tcclient, TC_T *tc)
```

Parameters

`const TCClient_T *tcclient`
TC Client data structure containing all active TCs

`TC_T *tc`
TC to connect to

Returns

`bool_t`
`TRUE` Connection started
`FALSE` Connection not started

`TC_ValueCommand_MsgHandler()`

Value command: the process data value is the value of the data entity specified by the data dictionary identifier. This command is used both to answer a request value command and to set the value of a process data entity. The layout of this message is defined in B.3

Signature

```
void TC_ValueCommand_MsgHandler(ISOBUS_Message_T *message, const TCClient_T
*tcclient, TC_T *tc)
```

Parameters

`ISOBUS_Message_T *message`
Pointer to the received TC Process Data Message

`const TCClient_T *tcclient`
TC Client structure containing all active TC's

`TC_T *tc`
Pointer to the application's active TC data structure

Returns

`void`

`TC_Version_MsgHandler()`

Function receives Version message from the TC

Signature

```
void TC_Version_MsgHandler(ISOBUS_Message_T *message, TC_T *tc)
```

Parameters

`ISOBUS_Message_T *message`
Pointer to the received TC message

`const TC_T *tc`
Pointer to the app's TC_T data structure

Returns

`void`

`TCClient_Init()`

Initializes TCClient_T structure

Signature

```
void TCClient_Init(TCClient_T *tcclient)
```

Parameters

`TCClient_T *tcclient`
TC Client structure containing all active TCs

Returns

`void`

`TCClient_Task()`

Runs all the TC Client tasks

Signature

```
void TCClient_Task(const TCClient_T *tcclient)
```

Parameters

`TCClient_T *tcclient`
TC Client structure containing all active TCs

Returns

`void`

`TCClient_Uninit()`

Uninitializes TCClient_T structure

Signature

```
void TCClient_Uninit(TCClient_T *tcclient)
```

Parameters

`TCClient_T *tcclient`
TC Client structure containing all active TCs

Returns

```
void
```

`TCtoECU_MsgHandler()`

Message/Event handler for TC to ECU messages

Signature

```
void TCtoECU_MsgHandler(ISOBUS_Message_T *message, ISOBUS_MessageEvent_T
event, const struct Transport_MessageHandler_Node_S *handler_node)
```

Parameters

`ISOBUS_Message_T *message`
Message to handle

`ISOBUS_MessageEvent_T event`
Multi-packet event (reason for the call)

`const struct Transport_MessageHandler_Node_S *handler_node`
Pointer that includes task controller client data structure

Returns

```
void
```