VIRTEC Foundation Library User Guide

Table of Contents

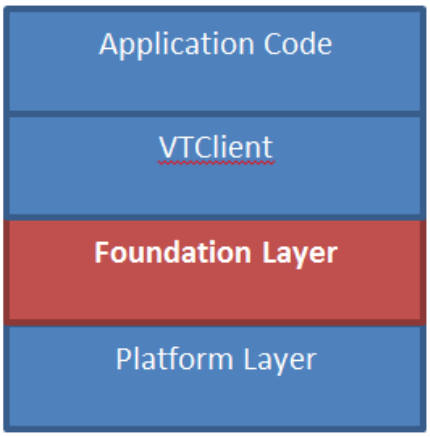
Introduction

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 Foundation Library.

Overview



VIRTEC Layers block diagram

The foundation layer provides core ISOBUS functionality, including:

- Network NAME Table Management
- Address Claiming
- TP and ETP message transport management
- Packet routing
- Request and acknowledgement handling
- ECU, Software, and Product ID management
- Diagnostic protocols

Definitions

Message
A multi-packet message. Within the VIRTEC libraries, messages are of type `ISOBUS_Message_T`. See [`Multi-Packet Messages`].

Packet
A single packet message. Within the VIRTEC libraries, packet are of type `ISOBUS_Packet_T`. See [`Single Packet Messages`].

Setting up Foundation

Foundation Data Structures and Required Declarations

In order to develop an ISOBUS application, several foundation data structures must be declared.

Pipes

Pipes allow for data to be transported between different parts of an application. Pipes are declared as type `Pipe_T` while a collection of pipes are declared as type `Pipes_T`

Pipes have a name, a priority and a size.

name is self explanatory
priority is the scheduler priority upper limit for all tasks that can access this pipe
size is the maximum number of bytes the pipe can hold

Declaring pipes and a pipe collection

The easiest method of creating a collection of pipes for use in your application is by storing pipe information in a separate header file and using the Foundation's built-in macros, along with some custom ones for reading the appropriate information from the header file.

Example Pipes.h

Example

```
#ifndef PIPE
#define PIPE(name, priority, size)
#endif //PIPE

//PIPE(name,priority,size)
PIPE(Pipe0, MY_MUTEX_PRIORITY,    8)
PIPE(Pipe1, MY_MUTEX_PRIORITY,    8)
PIPE(Pipe2, MY_MUTEX_PRIORITY,    8)
PIPE(Pipe3, MY_MUTEX_PRIORITY,    8)
PIPE(Pipe4, MY_MUTEX_PRIORITY,  256)
PIPE(Pipe5, MY_MUTEX_PRIORITY,  256)
PIPE(Pipe6, MY_MUTEX_PRIORITY,  512)
PIPE(Pipe7, MY_MUTEX_PRIORITY,  512)
PIPE(Pipe8, MY_MUTEX_PRIORITY, 1785)
PIPE(Pipe9, MY_MUTEX_PRIORITY, 1785)

#undef PIPE
```

Pipe Initialization Code

Example

```
// Create individual Pipe arrays (for pipe collection)
#define PIPE(name, priority, size)  static MAKE_PIPE_ARRAY(name,
MinAddressable_T, size);
#include "Pipes.h"

// Pipe Array
static Pipe_T MyApp_PipeArray[] =
{
```

```
  #define PIPE(name, priority, size)  MAKE_Pipe_T(name, priority),
  #include "Pipes.h"
};

// Final Pipe Collection
static Pipes_T MyApp_PipeCollection = MAKE_Pipes_T(MyApp_PipeArray,
MY_MUTEX_PRIORITY);
```

Transport Sessions

Transport sessions hold all the necessary information to keep track of TP and ETP transport sessions in progress.

Declaring Transport Sessions

Transport sessions are declared as type `ISOBUS_TransportSession_T`

You'll want to ensure that your foundation includes enough transport sessions to support the requirements of your application.

Example

```
// TP session array for Application
static ISOBUS_TransportSession_T MyApp_TP_Sessions[4];
```

Functionalities

Functionalities are used for conformance testing and designate the capabilities of your application. If your application will not be conformance tested, you do not need to include them. At a minimum, an application must include the Minimum Control Function functionality.

For VTClient applications, you will also need to include the Universal Terminal Working Set functionality.

The Foundation library and the VTClient library include MAKE_XXXX macros you can use to declare your applications functionalities. See the example code below for usage.

Declaring functionalities

Example

```
// Functionalities supported by this application
static const Functionalities_T  MyApp_Functionalities[] =
{
  // Supports Minimum Control Functionality
  MAKE_Functionalities_T__MinimumControlFunction(),
  // Supports Universal Terminal
  MAKE_Functionalities_T__UniversalTerminal_WorkingSet()
};
```

Diagnostic Trouble Codes

If you application needs to implement Diagnostics, a DTC structure will need to be declared.

Declaring the DTC Structure

Diagnostics can be declared easily using the MAKE_DTC_T macro. The first parameter is the SPN (Suspect Parameter Number) and the second is the FMI (Failure Mode Indicator).

Example

```
// DTCs used by MyApp
static const DTC_T  MyApp_DTCArray[] =
{
  MAKE_DTC_T(0, 0)
};

static DTC_Status_T
MyApp_DTCStatusArray[sizeof(MyApp_DTCArray)/sizeof(DTC_T)];
```

RX Message Queue

Received packets can be queued to lower processing priority. This is an optional feature depending on the requirements of your application.

Declaring an RX Message Queue

The foundation library comes with several macros to simplify the code necessary to declare an RX queue. Please see the below example code for usage.

Example

```
// Queue used for receive packets to lower processing priority
static MAKE_QUEUE_ARRAY(MyApp_RxQueueArray, ISOBUS_Packet_T, 50);
static Queue_T MyApp_RxQueue = MAKE_Queue_T(MyApp_RxQueueArray,
MY_MUTEX_PRIORITY);
```

Software ID

The software will require a software version to be associated with the Foundation.

Declaring a Software ID

The are several macros to simplify the code necessary to declare and register a software ID. Please see the below example code for usage.

Example

```
// Version of App Software    (Product,Major,minor,build)
#define APP_SOFTWARE_VERSION    SoftwareVersion("MyApp",0,0,1)
```

```
// Initialize Software ID structure for the App software
SoftwareId_T MyApp_SoftwareIdEntry = MAKE_SoftwareId_T(APP_SOFTWARE_VERSION);

// Register Software ID to the list
SoftwareId_Register(&Solution_SoftwareId_List, &MyApp_SoftwareIdEntry);
```

Foundation

Declaring a Foundation

Example

```
// Create Foundation Functionality structure
Foundation_T MyApp_Foundation =
  MAKE_Foundation_T(
    &Solution_SwTimerList,
    &Networks[0],
    // ************************************************************
    // sa_primary            = 128 (0x80) Primary source address
    // choose_sa_fn          = NULL (use built-in 128-247 range)
    // priority              = PL_8
    // ************************************************************
    MAKE_ISOBUS_AddressClaim_S(128, NULL, PL_8),
    // ************************************************************
    // self_configurable     = 1, this is a Self-configurable address
    // industry_group        = 2, Agricultural and forestry equipment
    // device_class_instance = 0,
    // device_class          = 2,
    // function              = 129, On-board Diagnostic Unit
    // function_instance     = 0,
    // ecu_instance          = 3,
    // manufacturer_code     = 514, DISTek Integration, Inc
    // identity_number       = 1
    // ************************************************************
    MAKE_ISOBUS_Name_T(1,2,0,2,129,0,3,514,1),
    MAKE_ISOBUS_Transport_T(MY_MUTEX_PRIORITY, 2, 16, MyApp_TP_Sessions,
MyApp_PipeCollection),
    MAKE_LanguageCallbackList_T(MY_MUTEX_PRIORITY),
    MAKE_ISOBUS_EcuId_T(MY_MUTEX_PRIORITY, Solution_EcuId_Fields),
    MAKE_ISOBUS_SoftwareId_T(PL_6, Solution_SoftwareId_List),
    MAKE_ISOBUS_ProductId_T(MY_MUTEX_PRIORITY, Solution_ProductId_Fields),
    MAKE_ISOBUS_DiagnosticProtocol_T(ECU_DIAGNOSTICS_ISO_11783_LEVEL_1,
MY_MUTEX_PRIORITY),
    MAKE_DTC_List_T(MyApp_DTCArray, MyApp_DTCStatusArray, MY_MUTEX_PRIORITY),
    MAKE_ISOBUS_Functionalities_T(MY_MUTEX_PRIORITY, MyApp_Functionalities),
    MAKE_ISOBUS_Certification_T(14, 0, 514, 7, 0,  1, 0, 0, 0, 0, 0, 1, 1, 0,
0, 0, 0, 0),
    MAKE_Memory_T(Memory_Read, Memory_Write),
    MAKE_Foundation_PacketHandler_List_S(&MyApp_RxQueue, NULL,
MY_MUTEX_PRIORITY),
```

```
    MAKE_Request_S(MY_MUTEX_PRIORITY),
    MAKE_Acknowledge_S(MY_MUTEX_PRIORITY)
  );
```

Foundation Initialization

Once the Pipes and `Foundation_T` structure have been set up, using the corresponding macros, they will need to be initialized. Within the initialization function for the application, the `Pipes_Init(&<pipe collection name>)` function and `Foundation_Init(&<Foundation_T structure name>)` function will need to be called. Neither of these functions will return any value.

See example code below for a function definition of a foundation initialization routine.

Example

```
void MyApp_ISOBUS_Init(void)
{
   Pipes_Init(&MyApp_PipeCollection);
   Foundation_Init(&MyApp_Foundation);
}
```

Scheduling a Foundation Task

There are three functions that must be referenced within the task scheduler pertaining to the Foundation.

- An initialization function
- The periodic `Foundation_Task` function
- The `Foundation_Uninit` function

The initialization function you must write yourself, an example is included above. The other two functions are part of the library and must therefore be passed a pointer to your foundation structure. See the example code below for guidance.

Example

```
//   Init function
INIT(MyApp_ISOBUS_Init)

//   Task function,                        period (ms),   priority
LIBRARY_TASK(Foundation_Task, &MyApp_Foundation,   80,   PL_10)

//   Exit function
LIBRARY_EXIT(Foundation_Uninit, &MyApp_Foundation)
```

Using the Foundation

Receiving

Available API Functions

- Transport_MessageHandler_Register()
- Transport_MessageHandler_Unregister()

Transmitting

Available API Functions

- Transport_SendMessage()

Canceling

Available API Functions

- Transport_Abort()

Filtering

Available API Functions

- Filter_DestinationSpecificToMe()
- Filter_SentToMyWorkingSet()
- Filter_GlobalOrWorkingSetOrDestinationSpecificToMe()
- Filter_SentToWorkingSetMember()
- Filter_SentFromMyWorkingSet()

Writing your own Filter

The Filter function receives a pointer to the incoming `packet`, and to the application's `struct Foundation_S` structure. This should grant the function access to all the information necessary to determine whether or not to filter a packet. This includes access to the `Network_T` structure via the `struct Foundation_S` structure.

To accept the packet, return `TRUE`. To reject the packet, return `FALSE`.

`Filter_<name>()` functions must conform to the following Function Prototype (where the actual name of the function is user-defined):

Example

```
bool_t Filter_Name(const ISOBUS_Packet_T *packet, const struct Foundation_S
*foundation);
```

Developer Notes

Filter functions limit the processing of packets to those of concern. A number of `Filter_<name>()` functions are provided by the VIRTEC Foundation library. A user may also write their own.

Note: Because the application is typically interested in packets destined globally, to the working set, or to the application's claimed address, the default filter used in most cases is Filter_GlobalOrWorkingSetOrDestinationSpecificToMe().

Address Claiming

Available API Functions

- AddressClaim_IsClaimed()

Developer Notes

Address claiming is handled automatically by the Foundation library. After initialization the Foundation will attempt to claim the address specified in your Foundation data structure. If the desired address cannot be claimed, the foundation will attempt to claim the next available address within the range 128 - 247. If you opted to provide a source address function to the `MAKE_ISOBUS_AddressClaim_S` macro, then that user-defined function will be called to determine the next source address to try.

In your application code, the only address claim function that may be of use to you is the `AddressClaim_IsClaimed()` function, which will return a boolean indicating whether or not the foundation has claimed an address. The other address claim functions are used internally in the foundation and should not be used.

Network Management

Available API Functions

- Network_SendPacket()

Developer Notes

The Network Management API Functions are used to transport data over a CAN network.

The `Network_SendPacket()` function sends a packet over the ISOBUS, while respecting the ISOBUS Polite Address Claim rules.

Transport

Available API Functions

- Transport_SendMessage()
- Transport_Abort()

Developer Notes

Transport sessions handle the transfer of multi-packet data on the bus using TP or ETP protocol. The decision of which (TP or ETP) is handled by the Foundation automatically and determined by the size of the data you want to transfer.

For more detailed information on sending multi-packet messages, please see the section of this document titled *Transmitting and Receiving Messages*.

Identification

Available API Functions

- SoftwareIdList_Init()
- SoftwareId_Register()
- SoftwareId_Unregister()

Developer Notes

These API functions should be used within your application to register your software with the linked list of Software IDs.

Utilities

Available API Functions

- Utility_MemoryCopy()
- Utility_ToLowerCase()
- Utility_ToUpperCase()
- String_Length()
- String_LimitedLength()

Developer Notes

The Utility API Functions allow you to perform a few basic operations without having to write your own function to achieve the same interaction with the Foundation.

One of the most useful of these utility API functions is the `Utility_MemoryCopy()` which allows you to input a pointer to a source, and another pointer to a destination, and the Utility API function will copy the data in the memory location of the source pointer into the memory location of the destination pointer.

The `Utility_ToLowerCase()` and `Utility_ToUpperCase()` API functions are fairly straight-forward. They take the input character(s) and convert them, as described in the function name, either from uppercase to lower case, or from lower case to upper case.

The `String_Length()` API function is also fairly strait-forward. It returns the size of the string to which the input pointer is pointing to. The `String_LimitedLength()` API function performs the same operation, but it also allows you to set a limit on the size that will be

returned. If the string in question is longer than the specified limit, the function will return the value of the specified limit.

Timing

Available API Functions

- SoftwareTimerList_Init()
- SoftwareTimer_PeriodicTask()
- SoftwareTimer_Register()
- SoftwareTimer_Unregister()
- SoftwareTimer_Get()
- SoftwareTimer_Set()

Developer Notes

Software timers allow you to track the passage of time within your application. The foundation maintains a list of software timers.

To create a timer for use in your application, you can use the Foundation's [`MAKE_SoftwareTimer_T`] macro.

Example

```
SoftwareTimer_T my_timer = MAKE_SoftwareTimer_T();
```

To create a *list* of timers for use in your application, you can use the Foundation library's [`MAKE_SoftwareTimerList_T`] macro. See example below. In this example, we are creating a list of software timers, specifying that the the periodic timers maintenance task `SoftwareTimer_PeriodicTask()` will be called by the task scheduler every 10 milliseconds, and that the ceiling priority of tasks that can access the software timer is "`PRIORITY_MAX`". Note that period specified here (10 milliseconds in this example) must match the actual task scheduling period in your task scheduler, for calls to the `SoftwareTimer_PeriodicTask()` function.

Example

```
SoftwareTimerList_T  Solution_SwTimerList =
MAKE_SoftwareTimerList_T(milliseconds(10), PRIORITY_MAX);
```

Once a timer list has been declared, the list must be initialized with the `SoftwareTimerList_Init()` function. Simply pass it a pointer to the timer list you have declared.

Once your timers list has been declared, initialized, and scheduled in your task scheduler, you can then register new timers to your timer list as well as unregister previously registered timers. To do this you can use the register and unregister functions listed above.

Once registered, a timer's value can be set using the `SoftwareTimer_Set()` function. There are a series of macros that will help you set a timer using the proper units you wish to use.

The macros will generate the proper value, of the proper data type ([`Time_T`]). These macros are:

- microseconds(x)
- milliseconds(x)
- seconds(x)

Example

```
// Set time to 100ms
SoftwareTimer_Set(&my_timer, milliseconds(100));
```

Once a timer's value is set, its value will be decremented in real time by the scheduled periodic task until it reaches 0. For code readability the value 0, of type `Time_T` can be returned by the macro `TIMER_EXPIRED`. See the example below for usage, which also demonstrates usage of the `SoftwareTimer_Get` function:

Example

```
if(SoftwareTimer_Get(&my_timer) == TIMER_EXPIRED)
{
  // Do something we were waiting to do
}
```

API Reference

Data Types

AddressClaim_PendingEchoCount_T: uint8_t
CAN_Identifier_T: uint32_t
DTC_Index_T: uint16_t
DTC_OccurrenceCount_T: uint8_t
DTC_T: uint8_t
EcuDiagnosticProtocolId_T: uint8_t
Frequency_T: uint32_t
FunctionalityGeneration_T: uint8_t
ISOBUS_DLC_T: uint32_t
ISOBUS_GroupFunction_T: uint8_t
ISOBUS_ManufacturerCode_T: uint16_t
ISOBUS_PacketData_T: unsigned char
ISOBUS_PacketPriority_T: uint8_t
ISOBUS_PGN_T: uint32_t
ISOBUS_PacketSequence_T: uint32_t
ISOBUS_TransportRetry_T: uint8_t
NameTableIndex_Bitfield_T: uint32_t
NameTableIndex_T: uint8_t

SourceAddress_T: uint8_t
Time_T: uint32_t

Enumerations

ISOBUS_Direction_T

This enum is used to identify whether an ISOBUS packet/message is being sent or received by the CAN hardware

Signature

```
typedef enum ISOBUS_Direction_E ISOBUS_Direction_T
```

Members

ISOBUS_RX
ISOBUS packet is received by the CAN hardware

ISOBUS_TX
ISOBUS packet is sent by the CAN hardware

Functions

AddressClaim_IsClaimed()

API indicating whether the application has an address. Indicates that the application has claimed an address and is permitted to actively participate (send messages) on the bus.

Signature

```
bool_t AddressClaim_IsClaimed(const Foundation_T *foundation)
```

Parameters

foundation
Pointer to App Foundation_T structure

Returns

bool_t
TRUE The application may send messages
FALSE The application may not send messages

Filter_DestinationSpecificToMe()

Filter packets sent only to my claimed source address. Accepts only messages sent destination specific to the source address claimed by the supplied foundation structure. All other packets are rejected.

Signature

```
bool_t Filter_DestinationSpecificToMe(const ISOBUS_Packet_T *packet, const
struct Foundation_S *foundation)
```

Parameters

`packet`
Incoming packet to test

`foundation`
Pointer to the application's Foundation structure

Returns

`bool_t`
`TRUE` Packet passes the filter (process)
`FALSE` Packet failed the filter (drop)

`Filter_GlobalOrWorkingSetOrDestinationSpecificToMe()`

Filter packets sent to my claimed source address, globally, or to my working set master. Accepts only messages sent destination specific to the source address claimed by the supplied `foundation` structure, to the working set master address, or sent globally. All other packets are rejected.

Signature

```
bool_t Filter_GlobalOrWorkingSetOrDestinationSpecificToMe(const
ISOBUS_Packet_T *packet, const struct Foundation_S *foundation)
```

Parameters

`packet`
Incoming packet to test

`foundation`
Pointer to the application's Foundation structure

Returns

`bool_t`
`TRUE` Packet passes the filter (process)
`FALSE` Packet failed the filter (drop)

`Filter_SentFromMyWorkingSet()`

Filter packets sent from members of my working set. Accepts only messages sent from a member of my working set. All other packets are rejected.

Note: This filter will cause all of this application's transmitted packets to also be received!

Signature

```
bool_t Filter_SentFromMyWorkingSet(const ISOBUS_Packet_T *packet, const
struct Foundation_S *foundation)
```

Parameters

`packet`
Incoming packet to test

`foundation`
Pointer to the application's Foundation structure

Returns

`bool_t`
`TRUE` Packet passes the filter (process)
`FALSE` Packet failed the filter (drop)

`Filter_SentToMyWorkingSet()`

Filter packets sent only to my working set master. Accepts only messages sent destination specific to the source address claimed by the supplied `foundation` structure. All other packets are rejected.

Signature

```
bool_t Filter_SentToMyWorkingSet(const ISOBUS_Packet_T *packet, const struct
Foundation_S *foundation)
```

Parameters

`packet`
Incoming packet to test

`foundation`
Pointer to the application's Foundation structure

Returns

`bool_t`
`TRUE` Packet passes the filter (process)
`FALSE` Packet failed the filter (drop)

`Filter_SentToWorkingSetMember()`

Filter packets to a member of my working set. Accepts only messages sent to a member of my working set. All other packets are rejected.

Signature

```
bool_t Filter_SentToWorkingSetMember(const ISOBUS_Packet_T *packet, const
struct Foundation_S *foundation)
```

Parameters

packet
Incoming packet to test

foundation
Pointer to the application's Foundation structure

Returns

bool_t
`TRUE` Packet passes the filter (process)
`FALSE` Packet failed the filter (drop)

`Foundation_Init()`

Meta-task for initializing VIRTEC Foundation structure for one app

Signature

```
void Foundation_Init(Foundation_T *foundation)
```

Parameters

foundation
Pointer to the application's Foundation structure

Returns

(void)

`Foundation_PacketHandler_Register()`

Register a PacketHandler with a application's Foundation structure

Signature

```
bool_t Foundation_PacketHandler_Register(Foundation_T *foundation, struct
Foundation_PacketHandler_Node_S *handler)
```

Parameters

foundation
Pointer to the application's Foundation structure

handler
Packet handler struct to unregister

Returns

bool_t
`TRUE` Packet handler was successfully registered
`FALSE` Packet handler was not registered (perhaps already registered?)

Foundation_PacketHandler_Unregister()

Unregister a PacketHandler with a application's Foundation structure

Signature

```
bool_t Foundation_PacketHandler_Unregister(Foundation_T *foundation, struct
Foundation_PacketHandler_Node_S *handler)
```

Parameters

foundation
Pointer to the application's Foundation structure

handler
Packet handler struct to unregister

Returns

bool_t
`TRUE` Packet handler was successfully unregistered
`FALSE` Packet handler was not unregistered (was not registered in this list)

Foundation_Task()

Meta-task for processing all VIRTEC Foundation tasks for one app

Signature

```
void Foundation_Task(Foundation_T *foundation)
```

Parameters

foundation
Pointer to the application's Foundation structure

Returns

(void)

Network_SendPacket()

Send packet on CAN interface and enforce Address Claim rules. Acts as a gateway to enforce the ISOBUS Polite Address Claim rules. Don't send any massages until the NAME Table is populated. Then send all AddressClaim messages, and application messages if the application has claimed an address, and the message is not destination spicific or the destination is global or the destination address has also been claimed.

1. Return value
    1. `TRUE` indicates that the CAN driver accepted responsibility to ensure the `packet` goes out on the bus. This may mean that the CAN packet has been placed on the hardware to send, or that it is placed in a Queue and will be sent when there is opportunity.

    2. `FALSE` indicates that the CAN driver is unable to accept responsibility to send the packet, so the calling task should try again later.
2. Callback
    1. The callback function pointer is called when the packet is actually sent on the bus. This typically corresponds to the transmit interrupt after the packet is sent. In some cases, the CAN driver may call the callback when the packet is placed on the hardware.
    2. Passing `NULL` as the `callback` parameter is valid, and indicates that no callback is provided.

Signature

```
bool_t Network_SendPacket(ISOBUS_Packet_T *iso_packet, const
ISOBUS_Callback_T *callback, const Foundation_T *foundation)
```

Parameters

`iso_packet`
ISOBUS packet to be sent on the bus

`callback`
Callback to be called once packet is successfully sent

`foundation`
Pointer to the application's Foundation structure

Returns

`bool_t`
`TRUE` Packet queued to be sent
`FALSE` For some reason, packet will not be sent (at this time)

`SoftwareId_Register()`

Register a SoftwareId_T structure

Signature

```
bool_t SoftwareId_Register(SoftwareIdList_T *list, SoftwareId_T *swid)
```

Parameters

`list`
Software ID list with which to register

`swid`
Pointer to the Software_ID structure to be registered

Returns

bool_t
`TRUE` Successfully registered
`FALSE` Registration failed

`SoftwareId_Unregister()`

Register a SoftwareId_T structure

Signature

```
bool_t SoftwareId_Unregister(SoftwareIdList_T *list, SoftwareId_T *swid)
```

Parameters

list
Software ID list with which to unregister

swid
Pointer to the Software_ID structure to be unregistered

Returns

bool_t
`TRUE` Successfully unregistered
`FALSE` Unregistered failed

`SoftwareIdList_Init()`

Initialize the Software ID List

Signature

```
void SoftwareIdList_Init(SoftwareIdList_T *list)
```

Parameters

list
Software ID List to initialize

Returns

(void)

`SoftwareTimerList_Init()`

Initialize the Software Timer List

Signature

```
void SoftwareTimerList_Init(SoftwareTimerList_T *list)
```

Parameters

`list`
Software Timer List to initialize

Returns

(void)

`SoftwareTimer_Get()`

Get the value of a timer

Signature

```
Time_T SoftwareTimer_Get(const SoftwareTimer_T *timer)
```

Parameters

`timer`
timer to read

Returns

`Time_T`
Value of the timer

`SoftwareTimer_PeriodicTask()`

Decrements each timer in the list by timer period until it reaches a value of 0

Signature

```
SoftwareTimer_PeriodicTask(SoftwareTimerList_T *list)
```

Parameters

`list`
List of Software Timers to decrement

Returns

(void)

`SoftwareTimer_Register()`

Register a Software Timer

Signature

```
bool_t SoftwareTimer_Register(SoftwareTimerList_T *list, SoftwareTimer_T
*timer)
```

Parameters

`list`
List to register with

timer
timer to register

Returns

bool_t
`TRUE` Timer was successfully registered
`FALSE` Timer registration failed

`SoftwareTimer_Set()`

Set the value of a timer

Signature
```
void SoftwareTimer_Set(SoftwareTimer_T *timer, Time_T timeout)
```

Parameters

timer
Timer to set

timeout
Time until timeout

Returns

(void)

`SoftwareTimer_Unregister()`

Unregister a Software Timer

Signature

```
bool_t SoftwareTimer_Unregister(SoftwareTimerList_T *list, SoftwareTimer_T
*timer)
```

Parameters

list
List to unregister with

timer
timer to unregister

Returns

bool_t
`TRUE` Timer was successfully registered `FALSE` Timer registration failed

`String_Length()`

Determines length of string (no limit to length)

Signature

```
Size_T String_Length(const char *string)
```

Parameters

`string`
C string to determine length of

Returns

`Size_T`
Size of string

`String_LimitedLength()`

Determines length of string (with a maximum length)

Signature

```
Size_T String_LimitedLength(const char *string, Size_T limit)
```

Parameters

`string`
C string to determine length of

`limit`
Maximum length of string

Returns

`Size_T`
Size of string

`Transport_Abort()`

Abort a transport session (if it's still open)

Signature

```
bool_t Transport_Abort(const Foundation_T *foundation, const ISOBUS_Message_T
*message)
```

Parameters

`foundation`
Foundation Functionality structure for this application

`message`
Message/session to abort

Returns

bool_t
`TRUE` Session successfully closed
`FALSE` Session not closed

`Transport_MessageHandler_Register()`

Register a Message/Event Handler

Note: If the PGN is always a single packet message, you may improve performance by using Foundation_PacketHandler_Register()_ instead._

Signature

```
bool_t Transport_MessageHandler_Register(Foundation_T *foundation, struct
Transport_MessageHandler_Node_S *message_handler_node)
```

Parameters

`foundation`
Foundation Functionality structure for this application

`message_handler_node`
Node containing DataPage/PGN and handler to register

Returns

bool_t
`TRUE` `message_handler_node` was successfully registered
`FALSE` `message_handler_node` was not successfully registered

`Transport_MessageHandler_Unregister()`

Unregister a Message/Event Handler

Signature

```
bool_t Transport_MessageHandler_Unregister(Foundation_T *foundation, struct
Transport_MessageHandler_Node_S *message_handler_node)
```

Parameters

`foundation`
Foundation Functionality structure for this application

`message_handler_node`
Node to unregister

Returns

bool_t
`TRUE` `message_handler_node` was successfully unregistered
`FALSE` `message_handler_node` was not successfully unregistered

`Transport_SendMessage()`

Initiate the transport of a message and sends packet using appropriate protocol

Note: Please do not pass a valid structure with a `NULL_` function pointer_

Signature

```
bool_t Transport_SendMessage(const Foundation_T *foundation, ISOBUS_Message_T
*message, const ISOBUS_MessageCallback_T *callback)
```

Parameters

`foundation`
Foundation Functionality structure for this application

`message`
Message to Send

`callback`
Contains callback information

Returns

`bool_t`
`TRUE` Transport session opened
`FALSE` Transport session not opened

`Utility_MemoryCopy()`

Copies from source to destination

Signature

```
void Utility_MemoryCopy(void destination, void source, Size_T size)
```

Parameters

`destination`
Destination for data to copy

`source`
Source of data to copy

`size`
size of data to copy (from sizeof())

Returns

(void)

`Utility_ToLowerCase()`

Converts character to lower case

Signature

```
char Utility_ToLowerCase(char character)
```

Parameters

`character`
Character to convert

Returns

`char`
Character converted to uppercase

`Utility_ToUpperCase()`

Converts character to lower case

Signature

```
char Utility_ToUpperCase(char character)
```

Parameters

`character`
Character to convert

Returns

`char`
Character converted to uppercase

Macros

`MAKE_Acknowledge_S()`

This macro is used to initialize a struct Acknowledge_S

Signature

```
MAKE_Acknowledge_S(priority)
```

Parameters

`priority`
Highest priority of the tasks that access this structure

`MAKE_DTC_List_T()`

This macro is used to create the DTC_List_T

Signature

```
MAKE_DTC_List_T(dtc_array, dtc_status_array, priority)
```

Parameters

`dtc_array`
Array name for SPN/FMI information

`dtc_status_array`
Array name for active/count information

`priority`
Maximum task priority accessing DTCs

`MAKE_Foundation_T()`

This macro is used to initialize the Foundation_T type

Signature

```
MAKE_Foundation_T(sw_timer_list, network, addressclaim, name, transport,
language_callbacklist, ecu_id, software_id, product_id, diagnostics,
dtc_list, functionalities, certification, memory, packet_handlers, request,
acknowledge)
```

Parameters `sw_timer_list` : Pointer to the SoftwareTimerList used by this App

`network`
Pointer to the Network used by this App

`addressclaim`
Address Claim data structure for this App

`name`
8 byte array to hold the CAN Name for this application

`transport`
Transport Protocol structure (including Extended TP)

`language_callbacklist`
Linked list of callbacks

`ecu_id`
ECU ID structure

`software_id`
SW ID structure

`product_id`
Product ID structure

`diagnostics`
Diagnostics services structure

`dtc_list`
List of DTCs

functionalities
Functionalities services structure

certification
Certification structure

memory
Memory function pointer structure

packet_handlers
Allows registration of packet handlers

request
Allows registration of Request handlers

acknowledge
Allows registration of Acknowledgement handlers

MAKE_Foundation_PacketHandler_List_S()

This macro is used to initialize a struct Foundation_PacketHandler_List_S

Signature

```
MAKE_Foundation_PacketHandler_List_S(queue_ptr, global_filter, priority)
```

Parameters

queue_ptr
Pointer to optional Queue_T structure (NULL = no queue)

global_filter
Global filter applied to all packets received by this Foundation structure

priority
Highest priority of the tasks that access this structure

MAKE_ISOBUS_AddressClaim_S()

This macro is used to initialize an ISOBUS_AddressClaim_S structure

Signature

```
MAKE_ISOBUS_AddressClaim_S(sa_primary,choose_sa_fn,priority)
```

Parameters

sa_primary
Primary application source address

choose_sa_fn
Function pointer to choose next source address (NULL to use built-in function)

priority
Priority of calling function

MAKE_ISOBUS_Certification_T()

This macro is used to initialize the ISOBUS_Certification_T type

Signature

```
MAKE_ISOBUS_Certification_T(year, rev, lab_id, lab_type, reference_number,
min_ecu, tecu_1, tecu_2, tecu_3, class3_ecu, virtual_terminal, vt_ws_master,
vt_ws_member, task_controller, tc_ws_master, tc_ws_member, file_server,
gps_receiver)
```

Parameters

year
Year of the compliance test protocol to which the certification test was performed

rev
Revision of the compliance test performed. In years where there are multiple revisions of the test protocol, an alphabetic suffix is used in addition to the certification year

lab_id
Manufacturer code of the laboratory that performed the compliance test. In the case of a self-certified ECU, this matches the manufacturer code contained in the address claim PGN. The value of this parameter is assigned by committee

lab_type
Approving body for the certification laboratory (3-bits)
000 - Non-certified laboratory/self-certification
001 - European Union certified laboratory
010 - North American certified laboratory
111 - Not available (not certified)

reference_number
Certification reference number assigned by a certification laboratory. This value can be used together with the Certification Lab ID and ECU Manufacturer ID to uniquely identify the test file of the certification laboratory

min_ecu
Indicates whether the Minimum ECU compliance test was performed

tecu_1
Indicates whether the TECU Class 1 compliance test was performed

tecu_2
Indicates whether the TECU Class 2 compliance test was performed

tecu_3
Indicates whether the TECU Class 3 compliance test was performed

class3_ecu
Indicates whether the Class 3 ECU compliance test was performed

virtual_terminal
Indicates whether the Virtual Terminal compliance test was performed

vt_ws_master
Indicates whether the VT Working Set Master compliance test was performed

vt_ws_member
Indicates whether the VT Working Set Member compliance test was performed

task_controller
Indicates whether the Task Controller compliance test was performed

tc_ws_master
Indicates whether the TC Working Set Master compliance test was performed

tc_ws_member
Indicates whether the TC Working Set Member compliance test was performed

file_server
Indicates whether the File Server compliance test was performed

gps_receiver
Indicates whether the GPS Receiver compliance test was performed

MAKE_ISOBUS_DiagnosticProtocol_T()

This macro is used to initialize an ISOBUS_DiagnosticProtocol_T

Signature

```
MAKE_ISOBUS_DiagnosticProtocol_T(protocol, priority)
```

Parameters

protocol
Selected ECU diagnostic protocol enumeration

priority
Highest priority of the tasks that access this structure

MAKE_ISOBUS_EcuId_T()

This macro is used to initialize an ISOBUS_EcuId_T structure

Signature

```
MAKE_ISOBUS_EcuId_T(priority, fields)
```

Parameters

priority
Highest priority of the tasks that access this structure

fields
Name of the ECU ID fields (EcuIdFields_T) structure

MAKE_ISOBUS_Functionalities_T()

This macro is used to initialize an ISOBUS_Functionalities_T structure

Signature

```
MAKE_ISOBUS_Functionalities_T(priority, functionalities)
```

Parameters

priority
Highest priority of the tasks that access this structure

functionalities
Array of Functionalities_T

MAKE_ISOBUS_Name_T()

This macro is used to initialize the ISOBUS_Name_T type

Signature

```
MAKE_ISOBUS_Name_T(self_configurable, industry_group,
device_class_instance,device_class, function, function_instance,
ecu_instance, manufacturer_code, identity_number)
```

Parameters

self_configurable
Indicates whether a Control Function is self-configurable (1) or not (0)

industry_group
Defined and assigned by ISO, identifies NAMEs associated with industries (e.g. agricultural equipment)

device_class_instance
Indicates occurrence of a particular device class in a connected network; definition depends on industry group field contents

device_class
Defined and assigned by ISO; provides a common NAME for a group of functions within a connected network; when combined with an industry group, can be correlated to a common NAME, e.g "planter" with "agricultural equipment"

function
Defined and assigned by ISO; when value between 0 and 127, independent of any other field for definition; when > 127 but < 254, definition depends on device class; when

combined with industry group and device class, can be correlated to a common NAME for specific CF, though not implying any specific capabilities

function_instance
Indicates specific occurrence of a function on a particular device system of a network

ecu_instance
Indicates which of a group of ECUs associated with a given function is referenced

manufacturer_code
Assigned by committee (see ISO 11783-1); indicates manufacturer of ECU for which the NAME is being referenced; independent of any other NAME field

identity_number
Assigned by the ECU manufacturer

MAKE_ISOBUS_ProductId_T()

This macro is used to initialize an ISOBUS_ProductId_T structure

Signature

```
MAKE_ISOBUS_ProductId_T(priority, fields)
```

Parameters

priority
Highest priority of the tasks that access this structure

fields
Address of the ProductID list (SoftwareIdList_T) structure

MAKE_ISOBUS_SoftwareId_T()

This macro is used to initialize an ISOBUS_SoftwareId_T structure

Signature

```
MAKE_ISOBUS_SoftwareId_T(priority, list)
```

Parameters

priority
Highest priority of the tasks that access this structure

list
Address of the SwID list (SoftwareIdList_T) structure

MAKE_ISOBUS_Transport_T()

This macro is used to create a ISOBUS_Transport_T that uses an array of transport sessions previously declared using the MAKE_TRANSPORT_SESSION macro.

Signature

```
MAKE_ISOBUS_Transport_T(priority, max_retries, max_packets_per_cts,
tp_sessions, tp_pipes)
```

Parameters

priority
Highest priority of the tasks that access this structure

max_retries
Maximum number of times to retry a single transport session (standard recommends 2)

max_packets_per_cts
Maximum number of data packets that can be sent in response to a single CTS (standard recommends 16). This is also the maximum number of packets that can be re-requested

tp_sessions
Name of the array of transport sessions

tp_pipes
Name of the pipe collection

MAKE_LanguageCallbackList_T()

This macro is used to initialize a LanguageCallbackList_T structure

Signature

```
MAKE_LanguageCallbackList_T(priority)
```

Parameters

priority
Highest priority of the tasks that access this structure

MAKE_Memory_T()

This macro is used to initialize an Memory_T structure

Signature

```
MAKE_Memory_T(read, write)
```

Parameters

read
Generic function to read arbitrary memory locations/devices

write
Generic function to write arbitrary memory locations/devices

MAKE_Request_S()

This macro is used to initialize a struct Request_S

Signature

```
MAKE_Request_S(priority)
```

Parameters

`priority`
Highest priority of the tasks that access this structure

`MAKE_SoftwareTimer_T()`

This macro used to initialize a software timer of type [`SoftwareTimer_T`].

Signature

```
MAKE_SoftwareTimer_T()
```

Parameters

(none)

`MAKE_SoftwareTimerList_T()`

This macro is used to create a list of software timers.

Signature

```
MAKE_SoftwareTimerList_T(period, priority)
```

Parameters

`period`
Time between calls to the periodic task

`priority`
Highest priority of tasks that access the Software Timers in the list

Structures

`Foundation_T`

Contains all Foundation Functionality information for an ISOBUS App

Signature

```
typedef struct ButtonActivation_S ButtonActivation_T
```

Members

`SoftwareTimerList_T *TimerList`
Pointer to the SoftwareTimerList used by this App

`Network_T *Network`
Pointer to the Network used by this App

ISOBUS_AddressClaim_T AddressClaim
Address Claim data structure

ISOBUS_Name_T Name
8 byte array to hold the CAN Name for this application

ISOBUS_Transport_T Transport
Transport Protocol structure (including Extended TP)

LanguageCallbackList_T LanguageCallbackList
Linked list of callbacks

ISOBUS_EcuId_T ECU_ID
ECU ID structure

ISOBUS_SoftwareId_T SW_ID
SW ID structure

ISOBUS_ProductId_T Product_ID
Product ID structure

ISOBUS_DiagnosticProtocol_T Diagnostics
Diagnostics services structure

DTC_List_T DTCs
DTC List structure

ISOBUS_Functionalities_T Functionalities
Functionalities services structure

ISOBUS_Certification_T Certification
ISOBUS Compliance Certification message

Memory_T Memory
Memory function pointer structure

Foundation_PacketHandler_List_T PacketHandlers
Packet Handler list

struct Request_S Request
Request packet handlers

struct Acknowledge_S Acknowledge
Acknowledgement packet handlers

ISOBUS_PacketHeader_T

Defines the ISOBUS translation of the 29-bit identifier (with a few additional pieces of information)

Signature

```
typedef struct ISOBUS_PacketHeader_S ISOBUS_PacketHeader_T
```

Members

`ISOBUS_PGN_T PGN`
Parameter Group Number

`NameTableIndex_T Destination`
Destination Address

`NameTableIndex_T Source`
Source Address

`ISOBUS_Direction_T Direction`
Transmitted or Received?

`ISOBUS_PacketPriority_T Priority`
This is used to optimize packet transfer in a system