CONTROL SYSTEMS
AUTONOMOUS AND ASSISTANCE SYSTEMS
ELECTRIC AND HYBRID ELECTRIC SYSTEMS

ISOBUS C3.5 Quick start instructions

EPEC

General

- These instructions concentrate on how to start ISOBUS C3.5 project and quick overview on available features and the differences between C2.3 versus C3.5.
- SDK 4.6 does not include programming manual for ISOBUS C3.5
  - Manual will be added to future SDK
  - For library interface documentation, refer to ISOBUS library in CODESYS 3.5 IDE’s Library manager
  - For “how to” examples refer to:
    - this quick start guide
    - example applications available in Epec Extranet

Products

- ISOBUS C3.5 is currently available in MultiTool for EC44-020
  - Device description 3.5.13.62 or newer is required
- ISOBUS license is required in ECU
  - Library will give error if license is not valid
  - EC44-020**E** functional version includes ISOBUS license
- EC44-020E hardware is not AEF conformance tested

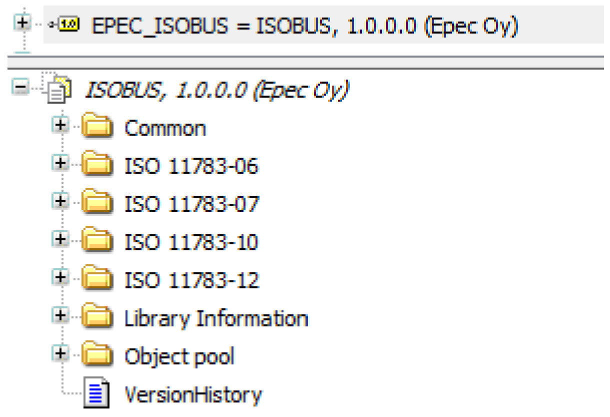
  →end-user application cannot receive AEF conformance test approval
- ISOBUS CAN sample point initialization is not implemented. This does not affect protocol functionality.

Example applications

- Following example test applications are available in Epec Extranet
  - VTComplexEC44
    - VT client examples for supported object types
  - VTAuxFunctionsEC44
    - Includes AUX Function objects for all of the AUX-N function types
    - VT client datamasks are used to visualize the AUX input values/states for testing purposes
  - EpecSprayerEC44
    - TC client application example
      - TC-BAS
      - TC-SC in 2 booms, 8 sections each
      - TC-GEO with 2 rate controls (one BIN per boom)
    - VT client datamasks are used to visualize the TC control states and values for testing purposes

Library

- ISOBUS protocols are implemented in a single C3.5 ISOBUS library which includes functionalities from following C2.3 libraries
  - ISOBUS.lib
  - ISOBUS_VT.lib
  - ISOBUS_TC.lib
  - ExtraBins.lib
- J1939 and AddressClaiming are separate libraries as in C2.3
- SDK4.6 includes ISOBUS library version 1.0.0.0

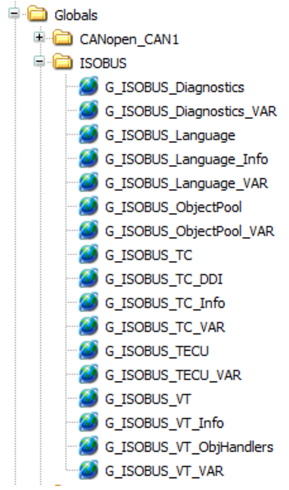


EPEC_ISOBUS = ISOBUS, 1.0.0.0 (Epec Oy)

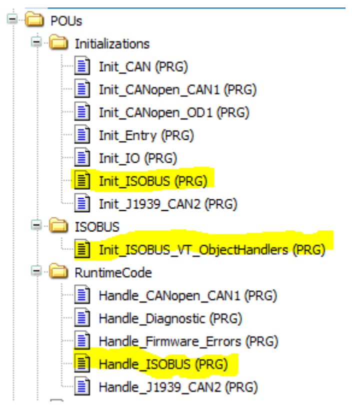
ISOBUS, 1.0.0.0 (Epec Oy)
- Common
- ISO 11783-06
- ISO 11783-07
- ISO 11783-10
- ISO 11783-12
- Library Information
- Object pool
- VersionHistory

Library & code template

- Library uses C3.5 features such as methods and interfaces → library is not interface compatible with C2.3 library even if similar functionalities are included
- Code template structure is also different from C2.3 and is now more like in CANopen implementation
- Generated code and GVLs depend on selected options in MultiTool
- Library initialization methods are automatically executed by code template



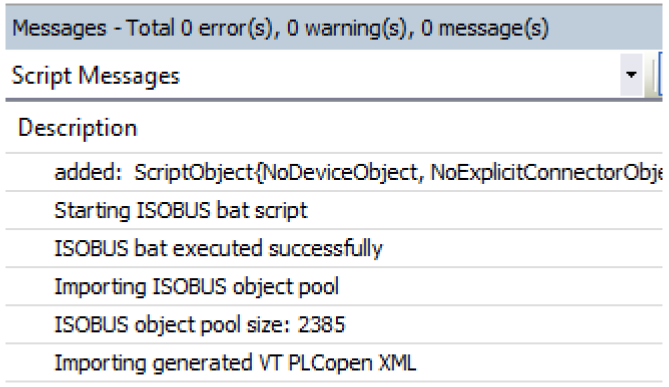
```
Globals
  CANopen_CAN1
  ISOBUS
    G_ISOBUS_Diagnostics
    G_ISOBUS_Diagnostics_VAR
    G_ISOBUS_Language
    G_ISOBUS_Language_Info
    G_ISOBUS_Language_VAR
    G_ISOBUS_ObjectPool
    G_ISOBUS_ObjectPool_VAR
    G_ISOBUS_TC
    G_ISOBUS_TC_DDI
    G_ISOBUS_TC_Info
    G_ISOBUS_TC_VAR
    G_ISOBUS_TECU
    G_ISOBUS_TECU_VAR
    G_ISOBUS_VT
    G_ISOBUS_VT_Info
    G_ISOBUS_VT_ObjHandlers
    G_ISOBUS_VT_VAR
```



```
POUs
  Initializations
    Init_CAN (PRG)
    Init_CANopen_CAN1 (PRG)
    Init_CANopen_OD1 (PRG)
    Init_Entry (PRG)
    Init_IO (PRG)
    Init_ISOBUS (PRG)
    Init_J1939_CAN2 (PRG)
  ISOBUS
    Init_ISOBUS_VT_ObjectHandlers (PRG)
  RuntimeCode
    Handle_CANopen_CAN1 (PRG)
    Handle_Diagnostic (PRG)
    Handle_Firmware_Errors (PRG)
    Handle_ISOBUS (PRG)
    Handle_J1939_CAN2 (PRG)
```

Object pool

- ISOBUS object pool is integrated to the CODESYS application so only application is loaded to the ECU (generated to GVL array but directed to flash memory in S/E series units)
- Object pool is automatically updated when CODESYS code template is generated or updated using project_update python script. So, there is no separate ”import ISOBUS” script in C3.5.



```
Messages - Total 0 error(s), 0 warning(s), 0 message(s)
Script Messages
Description
    added:  ScriptObject{NoDeviceObject, NoExplicitConnectorObje
    Starting ISOBUS bat script
    ISOBUS bat executed successfully
    Importing ISOBUS object pool
    ISOBUS object pool size: 2385
    Importing generated VT PLCopen XML
```

Migrating C2.3 to C3.5

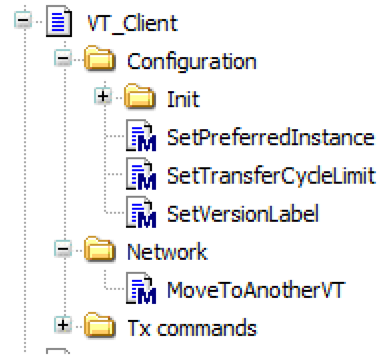
- ISOBUS C3.5 unit can be created :
  - By configuring new EC44-020 unit
  - Using MultiTool’s *Change unit type* feature to convert MultiTool configuration from e.g. 3724 to EC44
- General instructions for using change unit type is in programming manual’s Programming > How to change unit type
- Following are ISOBUS specific notes for change unit type:
  - ISOBUS is not available to CAN1 in S/E series units
  - ISOBUS resource template for C3.5 is different from C2.3 so don’t copy whole ISOBUS folder from C2.3 application.
    - When converting C2.3 unit to C3.5 the *Create CODESYS project* always needs to be used.
    - After change unit type is used in MultiTool it is recommended to either rename the old ECU’s folder or rename the EC44 unit in MultiTool. This is done to avoid unnecessary leftover files from C2.3 project in your C3.5 ISOBUS folder.
  - After C3.5 project template has been created the Jetter ISO-Designer project and TC’s object pool xml file can be copied from old project. Code template update will import the objects.
  - C3.5 ISO-Designer template project uses version 5.5.1 (C2.3 is 4.0.6). ISO-Designer can convert older project to newer version, but the old version should work also (not tested)
  - Since the library interfaces and code template differs from C2.3 it is recommended to re-implement existing program logic case-by-case basis. First generate the C3.5 code template and see what’s automatically provided since there are more automatically generated code compared to C2.3.
    - Also check the example applications for use-case examples

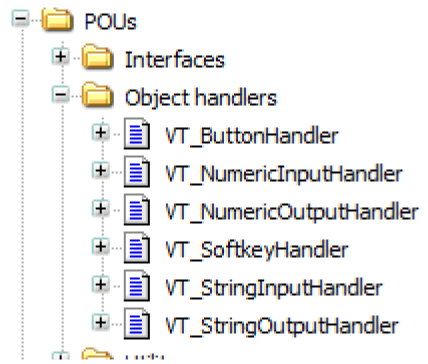
Features

- Implemented:
    - All main protocol options:
        - VT client (UT2.0)
        - AUX-N Functions
        - TC client (TC1.0)
        - TECU class 2 PGN interface
        - Diagnostics (Min CF 1.0)
    - MultiTool support
    - Code template support

Features, VT

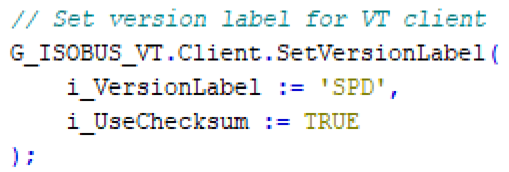
- VT client in code template is found at *G_ISOBUS_VT.Client*
- Many of the C2.3 functionalities are now methods (instead of functions or inputs)
- Mask handlers are optionally created by application so they are not enforced by the code template as in C2.3. C3.5 code template executes *VT_MainMaskHandler* which executes the initialized application mask handlers.
- Library includes several high-level object handlers which replace the PRGs from C2.3 code template. C3.5 code template executes *VT_MainObjectHandler* which is used to update the object specific handlers.
  - Code template's *G_ISOBUS_VT_ObjHandlers* GVL includes automatically generated handler instances (buttons, softkeys, input/output numbers). These are also initialized automatically in code template.
  - String handlers are implemented by library but defined and initialized by application when required
  - More handlers may be added to library in future releases
  - It is recommended to use high-level handlers for functionalities which they are available for (instead of low-level VT client interfaces)





Examples, VT version label

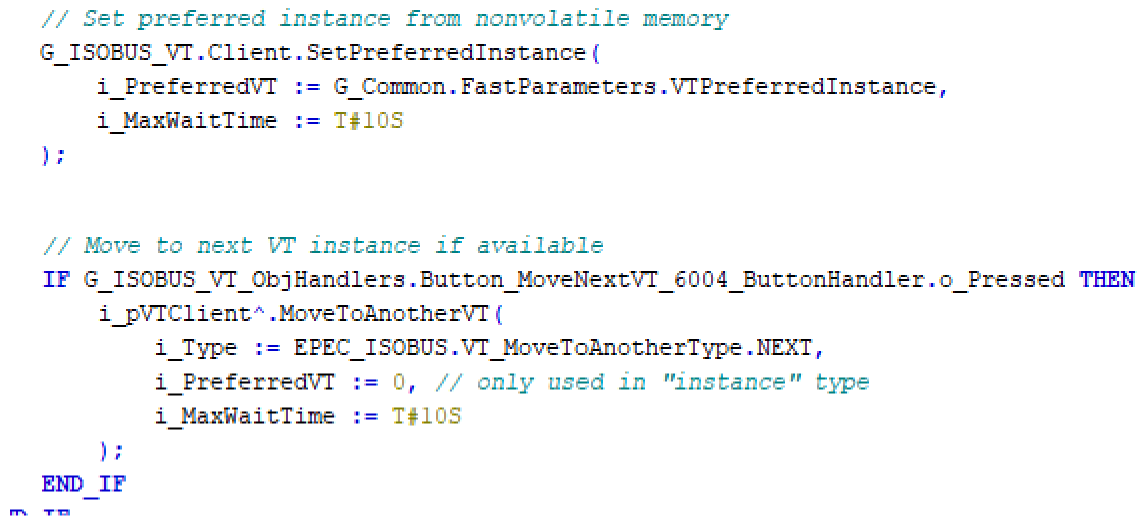
- If checksum is used, the version label behavior is as in C2.3. Otherwise, application may define the whole 7 characters label



```
// Set version label for VT client
G_ISOBUS_VT.Client.SetVersionLabel(
    i_VersionLabel := 'SPD',
    i_UseChecksum := TRUE
);
```

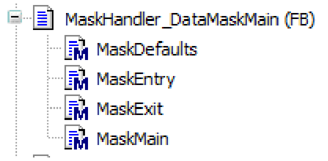
Examples, VT instance & move

- Preferred instance and move to another VT works in similar way than at C2.3 but they are used by methods instead of functions



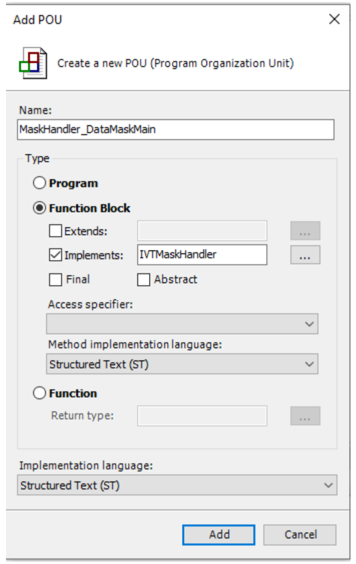
```
// Set preferred instance from nonvolatile memory
G_ISOBUS_VT.Client.SetPreferredInstance(
    i_PreferredVT := G_Common.FastParameters.VTPreferredInstance,
    i_MaxWaitTime := T#10S
);

// Move to next VT instance if available
IF G_ISOBUS_VT_ObjHandlers.Button_MoveNextVT_6004_ButtonHandler.o_Pressed THEN
    i_pVTClient^.MoveToAnotherVT(
        i_Type := EPEC_ISOBUS.VT_MoveToAnotherType.NEXT,
        i_PreferredVT := 0, // only used in "instance" type
        i_MaxWaitTime := T#10S
    );
END_IF
```

Examples, VT datamask



```
MaskHandler_DataMaskMain (FB)
    MaskDefaults
    MaskEntry
    MaskExit
    MaskMain
```

- Mask handler is created in similar way than VXD callbacks by creating FB POU and implementing *IVTMaskHandler* interface
  - The mask interface methods are automatically executed when mask is visible
  - Future note: Due to SP10 safety project restrictions these need to be implemented in application library at safety projects. This will restrict the direct access to code template variables and pointers may need to be used by application. This does not apply to non-safety units e.g. EC44.
- Application needs to initialize mask instance to main handler



Add POU

Create a new POU (Program Organization Unit)

Name: MaskHandler_DataMaskMain

Type

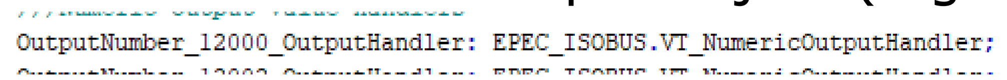
- Program
- [✓ Function Block]
  - Extends: ☐
  - Implements: ☑ IVTMaskHandler
  - Final ☐ Abstract ☐
  - Access specifier:
  - Method implementation language: Structured Text (ST)
- Function
  - Return type:

Implementation language: Structured Text (ST)

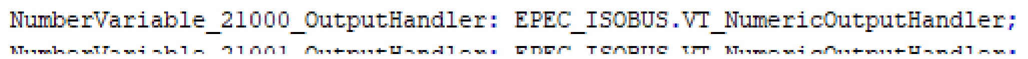
```
VAR_GLOBAL
    MH_DataMaskMain: MaskHandler_DataMaskMain;


   initErrors[1] := G_ISOBUS_VT.MainMaskHandler.AddMaskHandler(
       i_itfHandler := G_MaskHandlers.MH_DataMaskMain,
       i_MaskType := EPEC_ISOBUS.VT_MaskHandlerType.DATA_MASK,
       i_ObjectID := G_ISOBUS_VT_Info.OBJ_ID_DATAMASKMAIN
   );
```

Examples, VT numeric output

- Handler instance is automatically generated. If numeric variable is used, the handler is created for variable object. Otherwise, the instance is created for the output object (e.g. output number).

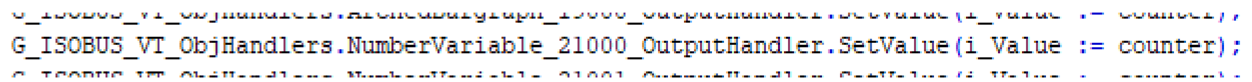


```
OutputNumber_12000_OutputHandler: EPEC_ISOBUS.VT_NumericOutputHandler;
```



```
NumberVariable_21000_OutputHandler: EPEC_ISOBUS.VT_NumericOutputHandler;
```

- Same numeric variable may be used by multiple output objects but do not use the same variable in input objects
- Updating output value:

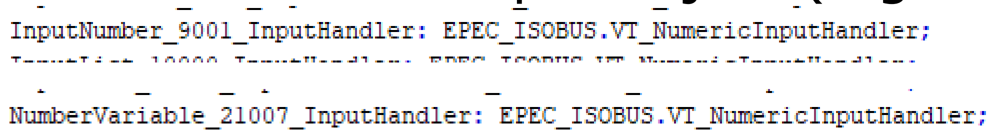


```
G_ISOBUS_VT_ObjHandlers.NumberVariable_21000_OutputHandler.SetValue(i_Value := counter);
```

- Send can be triggered manually by *SendTrigger* method or periodically by using *SetSendInterval*
- Minimum send interval can be set by *SetInhibitTime* method

Examples, VT numeric input

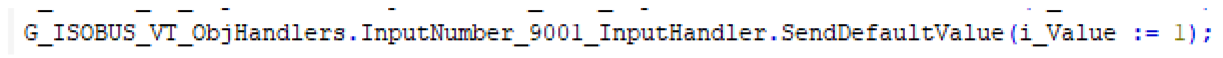
- Handler instance is automatically generated. If numeric variable is used, the handler is created for variable object. Otherwise, the instance is created for the input object (e.g. input number).



```
InputNumber_9001_InputHandler: EPEC_ISOBUS.VT_NumericInputHandler;
```

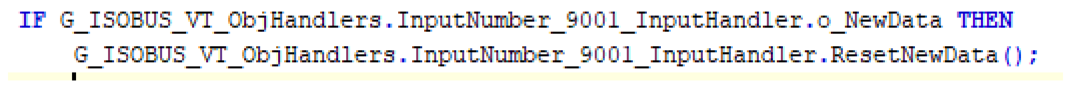
```
NumberVariable_21007_InputHandler: EPEC_ISOBUS.VT_NumericInputHandler;
```

- Optionally default value can be sent to input object:



```
G_ISOBUS_VT_ObjHandlers.InputNumber_9001_InputHandler.SendDefaultValue(i_Value := 1);
```

- NewData output is TRUE when new value is received. Flag can be reset by application:

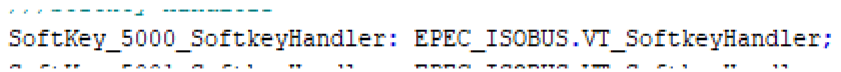


```
IF G_ISOBUS_VT_ObjHandlers.InputNumber_9001_InputHandler.o_NewData THEN
    G_ISOBUS_VT_ObjHandlers.InputNumber_9001_InputHandler.ResetNewData();
```

- Received value is in handler's *o_Value* output

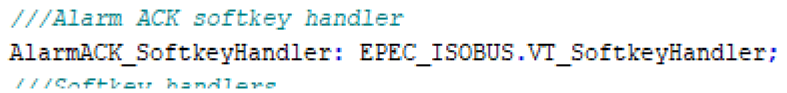
Examples, VT softkey

- Handler instance is automatically generated for Key objects



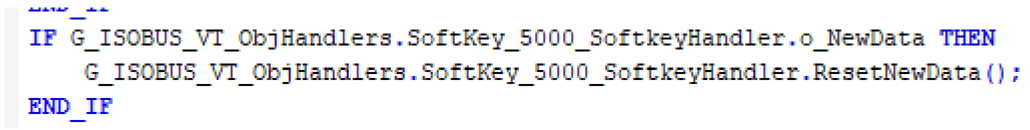
```
SoftKey_5000_SoftkeyHandler: EPEC_ISOBUS.VT_SoftkeyHandler;
```

- Special alarm ACK instance is always generated for receiving alarm ACK button status (this is not implemented in C2.3)



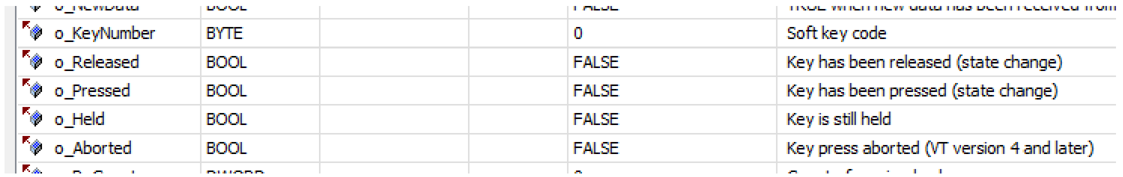
```
///Alarm ACK softkey handler
AlarmACK_SoftkeyHandler: EPEC_ISOBUS.VT_SoftkeyHandler;
```

- NewData output is TRUE when new state is received. Flag can be reset by application:



```
IF G_ISOBUS_VT_ObjHandlers.SoftKey_5000_SoftkeyHandler.o_NewData THEN
    G_ISOBUS_VT_ObjHandlers.SoftKey_5000_SoftkeyHandler.ResetNewData();
END_IF
```

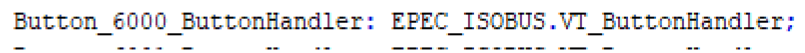
- Softkey statuses are in handler's outputs



o_KeyNumber	BYTE			0	Soft key code
o_Released	BOOL			FALSE	Key has been released (state change)
o_Pressed	BOOL			FALSE	Key has been pressed (state change)
o_Held	BOOL			FALSE	Key is still held
o_Aborted	BOOL			FALSE	Key press aborted (VT version 4 and later)

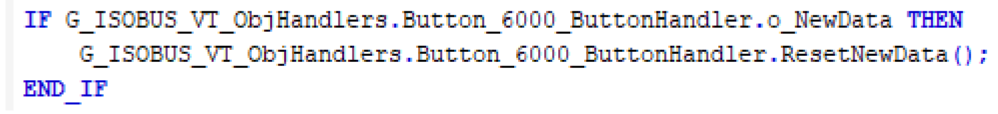
Examples, VT button

- Handler instance is automatically generated for Button objects



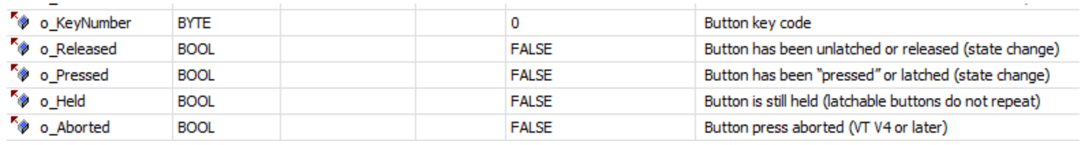
```
Button_6000_ButtonHandler: EPEC_ISOBUS.VT_ButtonHandler;
```

- NewData output is TRUE when new state is received. Flag can be reset by application:



```
IF G_ISOBUS_VT_ObjHandlers.Button_6000_ButtonHandler.o_NewData THEN
    G_ISOBUS_VT_ObjHandlers.Button_6000_ButtonHandler.ResetNewData();
END_IF
```

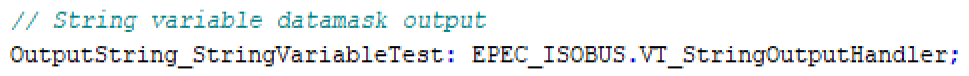
- Button statuses are in handler’s outputs



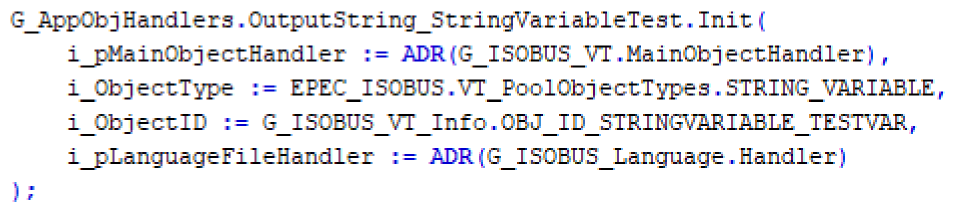
o_KeyNumber	BYTE			0	Button key code
o_Released	BOOL			FALSE	Button has been unlatched or released (state change)
o_Pressed	BOOL			FALSE	Button has been "pressed" or latched (state change)
o_Held	BOOL			FALSE	Button is still held (latchable buttons do not repeat)
o_Aborted	BOOL			FALSE	Button press aborted (VT V4 or later)

Examples, VT string output

- String output handler is implemented by library but defined and initialized by application
  - Create handler only for objects which you need to modify on-fly. Static texts do not need handler
- String output handler can:
  - Send local string variable from application code using *SetString* method
  - Read string from language file using *SetLanguageString* method
- Initialization example:



```
// String variable datamask output
OutputString_StringVariableTest: EPEC_ISOBUS.VT_StringOutputHandler;
```



```
G_AppObjHandlers.OutputString_StringVariableTest.Init(
    i_pMainObjectHandler := ADR(G_ISOBUS_VT.MainObjectHandler),
    i_ObjectType := EPEC_ISOBUS.VT_PoolObjectTypes.STRING_VARIABLE,
    i_ObjectID := G_ISOBUS_VT_Info.OBJ_ID_STRINGVARIABLE_TESTVAR,
    i_pLanguageFileHandler := ADR(G_ISOBUS_Language.Handler)
);
```

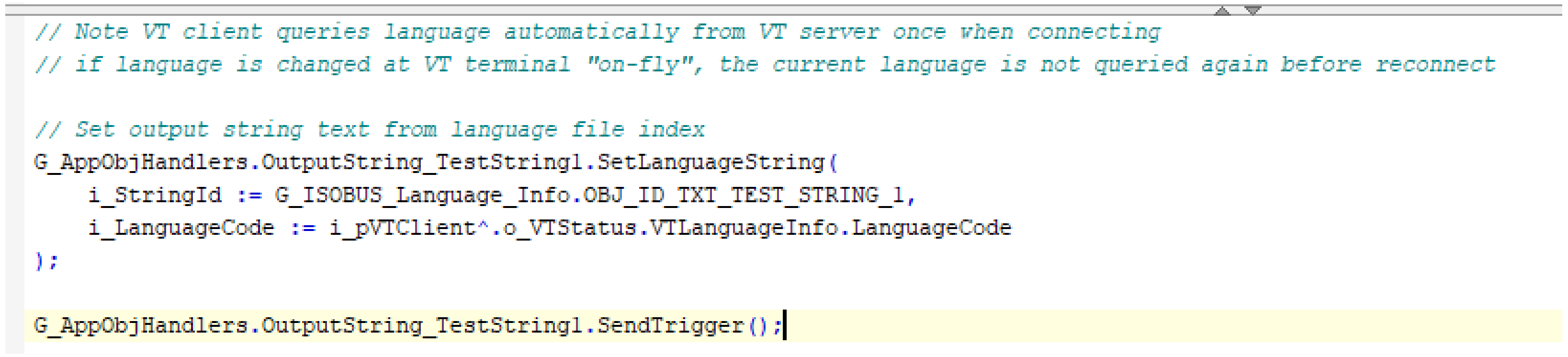
Examples, VT string output, local string

- Send local string example:

```
// set string to handler, length is modified on runtime
G_AppObjHandlers.OutputString_StringVariableTest.SetString(
    i_pString := ADR(localText),
    i_StringLength := strlen
);
G_AppObjHandlers.OutputString_StringVariableTest.SendTrigger();
```

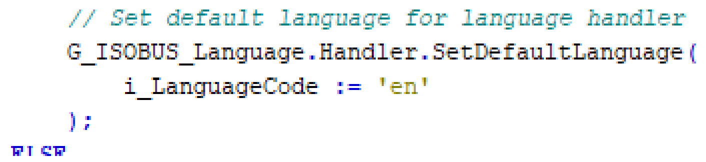
Examples, VT string output, using languages

- Language file is found in project's *ISOBUS\Python\Languages* folder
- Example how to send language string with string output handler:



```
// Note VT client queries language automatically from VT server once when connecting
// if language is changed at VT terminal "on-fly", the current language is not queried again before reconnect

// Set output string text from language file index
G_AppObjHandlers.OutputString_TestString1.SetLanguageString(
    i_StringId := G_ISOBUS_Language_Info.OBJ_ID_TXT_TEST_STRING_1,
    i_LanguageCode := i_pVTClient^.o_VTStatus.VTLanguageInfo.LanguageCode
);

G_AppObjHandlers.OutputString_TestString1.SendTrigger();
```

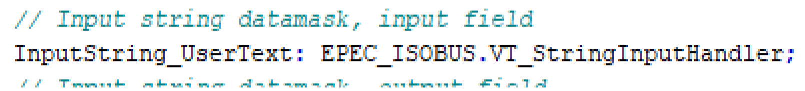
- Default language can be initialized to language handler if selected language is not found:



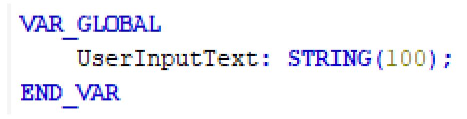
```
    // Set default language for language handler
    G_ISOBUS_Language.Handler.SetDefaultLanguage(
        i_LanguageCode := 'en'
    );
ELSE
```

Examples, VT string input

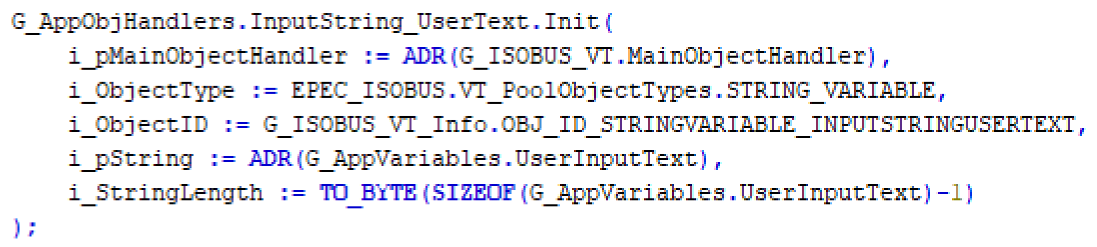
- String input handler is implemented by library but defined and initialized by application
- Application defines the string buffer where the input value is updated to
- Initialization example:



```
// Input string datamask, input field
InputString_UserText: EPEC_ISOBUS.VT_StringInputHandler;
// Input string datamask, output field
```

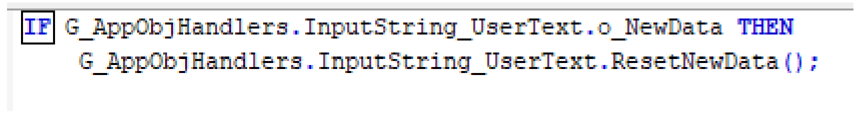


```
VAR_GLOBAL
    UserInputText: STRING(100);
END_VAR
```



```
G_AppObjHandlers.InputString_UserText.Init(
    i_pMainObjectHandler := ADR(G_ISOBUS_VT.MainObjectHandler),
    i_ObjectType := EPEC_ISOBUS.VT_PoolObjectTypes.STRING_VARIABLE,
    i_ObjectID := G_ISOBUS_VT_Info.OBJ_ID_STRINGVARIABLE_INPUTSTRINGUSERTEXT,
    i_pString := ADR(G_AppVariables.UserInputText),
    i_StringLength := TO_BYTE(SIZEOF(G_AppVariables.UserInputText)-1)
);
```

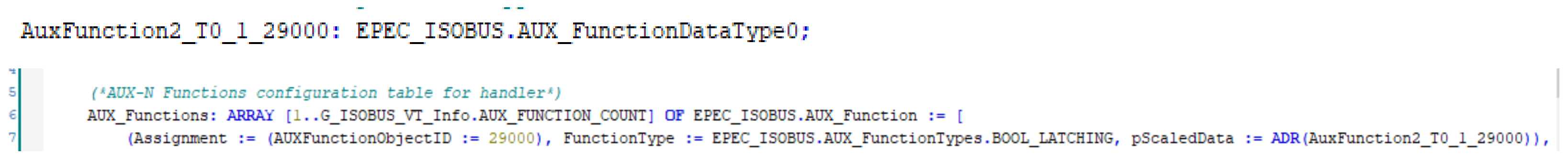
- NewData flag is set TRUE when new input text is received



```
IF G_AppObjHandlers.InputString_UserText.o_NewData THEN
    G_AppObjHandlers.InputString_UserText.ResetNewData();
```

Features, AUX-N

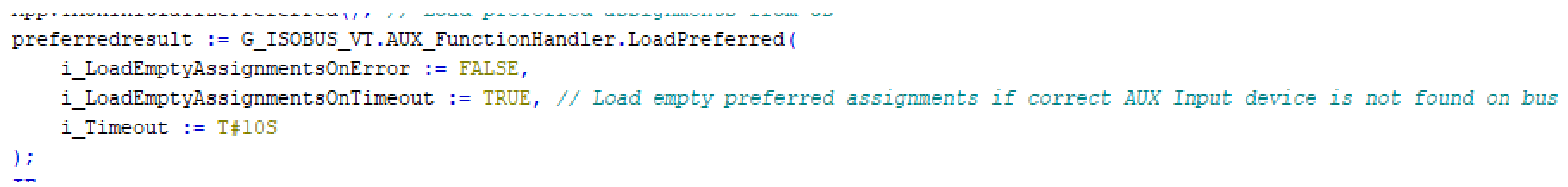
- AUX-N features are comparable to C2.3.
  - AUX function variables are found in *G_ISOBUS_VT_AUX* GVL
  - AUX Function handler instance is found in *G_ISOBUS_VT* GVL



```
AuxFunction2_T0_1_29000: EPEC_ISOBUS.AUX_FunctionDataType0;
```

```
    (*AUX-N Functions configuration table for handler*)
    AUX_Functions: ARRAY [1..G_ISOBUS_VT_Info.AUX_FUNCTION_COUNT] OF EPEC_ISOBUS.AUX_Function := [
        (Assignment := (AUXFunctionObjectID := 29000), FunctionType := EPEC_ISOBUS.AUX_FunctionTypes.BOOL_LATCHING, pScaledData := ADR(AuxFunction2_T0_1_29000)),
```

- LoadPreferred is method instead of function:

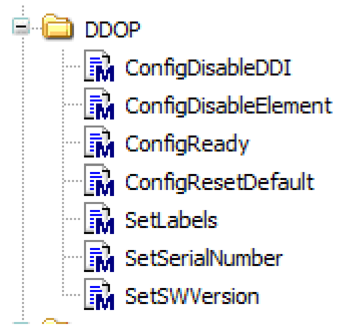


```
preferredresult := G_ISOBUS_VT.AUX_FunctionHandler.LoadPreferred(
    i_LoadEmptyAssignmentsOnError := FALSE,
    i_LoadEmptyAssignmentsOnTimeout := TRUE, // Load empty preferred assignments if correct AUX Input device is not found on bus
    i_Timeout := T#10S
);
```

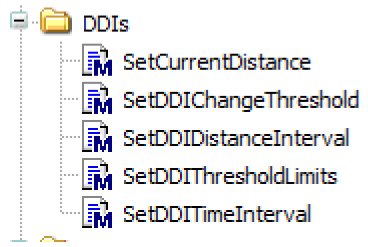
- Refer to example application for more information

Features, TC

- TC client in code template is found at *G_ISOBUS_TC.Client*
- DDI variable array is found in *G_ISOBUS_TC_DDI* GVL
- Many of the C2.3 functionalities are now methods (instead of functions or inputs)
  - DDOP configuration is done with methods
  - Default thresholds/intervals can be set by methods if default set is used and requested by TC
- Same DDI object ID can be referenced in multiple elements. In C2.3 this is not supported.
- Refer to Epec sprayer example application for full application example



- DDOP
  - ConfigDisableDDI
  - ConfigDisableElement
  - ConfigReady
  - ConfigResetDefault
  - SetLabels
  - SetSerialNumber
  - SetSWVersion



- DDIs
  - SetCurrentDistance
  - SetDDIChangeThreshold
  - SetDDIDistanceInterval
  - SetDDIThresholdLimits
  - SetDDITimeInterval

Examples, TC configuration

- TC DDOP serial number is set by code template
- TC DDOP version can be set by application:

```
// Set version to TC client DDOP, this application uses same version than diagnostic PGN
swversion := Standard.LEFT(STR:=G_ISOBUS_Diagnostics.Data.SwIdentification.Application_SW, 32);
G_ISOBUS_TC.Client.SetSWVersion(
    i_ClientSWVersion := swversion
);
```

- TC DDOP labels can be set by application:

```
TC_StructureLabel: ARRAY[1..7] OF BYTE;

TC_LocalizationLabel:EPEC_ISOBUS.PGN_Data_LanguageCmd;

// Set structure and localization label to TC client
G_ISOBUS_TC.Client.SetLabels(
    i_StructureLabel := TC_StructureLabel,
    i_LocalizationLabel := TC_LocalizationLabel,
    i_LoadDDOPFromLabel := TRUE
);
```

Examples, TC configuration

- DDI disabling is more flexible versus C2.3
  - ConfigDisableDDI disables single DDI variable

```
G_ISOBUS_TC.Client.ConfigDisableDDI(i_ObjectID:=G_ISOBUS_TC_DDI.DDIList[G_ISOBUS_TC_Info.DDI_INDEX_SECTION_CONTROL_STATE_OBJ_2105].ObjectId);
```

  - ConfigDisableElement disables element and its DDI variable references

```
G_ISOBUS_TC.Client.ConfigDisableElement(i_ElementNumber:=G_ISOBUS_TC_Info.DET_NBR_B1_S3_OBJ_10103);
```

  - If DDI exists in multiple elements, its disabled only if all element references are disabled or DDI object is disabled separately
  - C2.3 supports only disabling of whole element and its DDI references
- Config methods can be used when TC client is in RECONFIGURATION state
- ConfigReady method is mandatory even if application does not disable any DDIs. Client will stay in RECONFIGURATION state until ready is triggered.

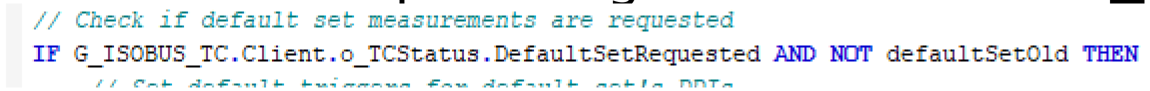
```
G_ISOBUS_TC.Client.ConfigReady();
```

- ConfigResetDefault method can be used to re-enable all objects

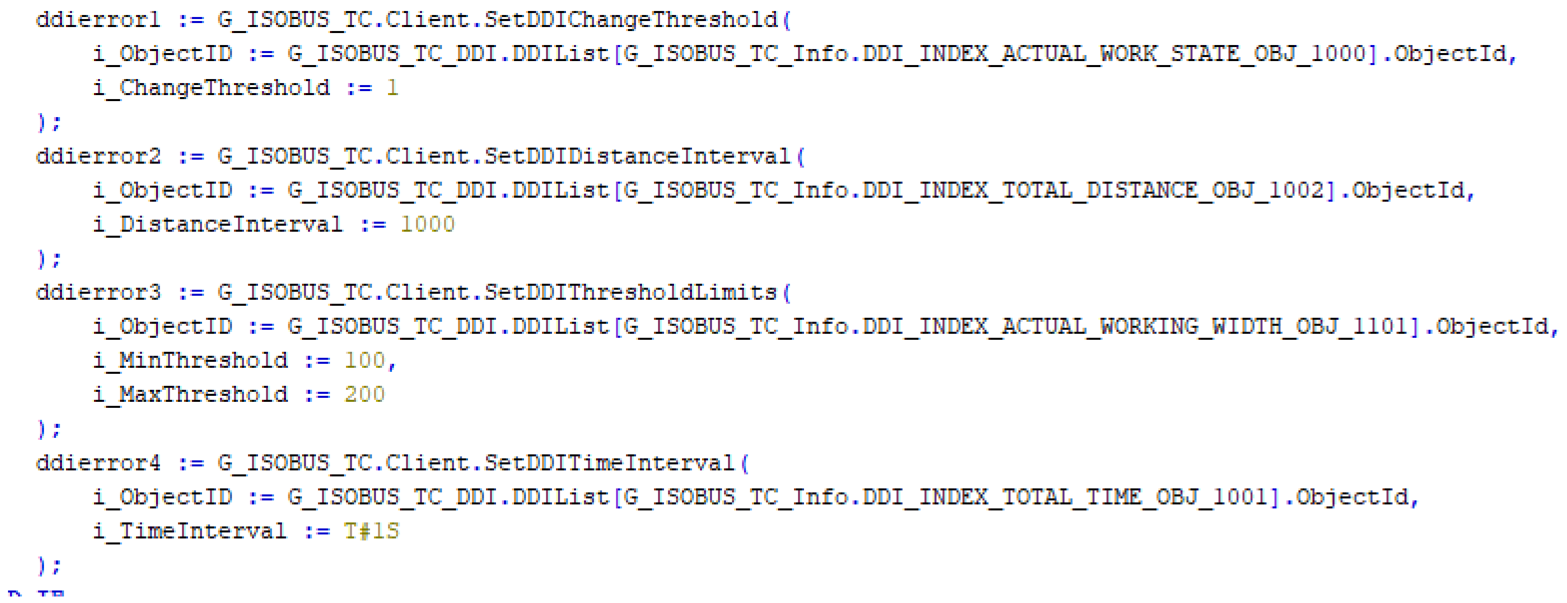
```
G_ISOBUS_TC.Client.ConfigResetDefault();
```

Examples, TC configuration

- If DDI belongs in default set and TC sends default set request, the application can use TC client methods to set default settings
- Default set request flag is in TC client's o_TCStatus output



```
// Check if default set measurements are requested
IF G_ISOBUS_TC.Client.o_TCStatus.DefaultSetRequested AND NOT defaultSetOld THEN
```



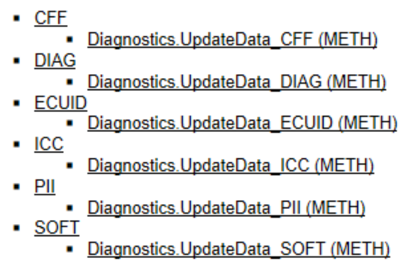
```
ddierror1 := G_ISOBUS_TC.Client.SetDDIChangeThreshold(
    i_ObjectID := G_ISOBUS_TC_DDI.DDIList[G_ISOBUS_TC_Info.DDI_INDEX_ACTUAL_WORK_STATE_OBJ_1000].ObjectId,
    i_ChangeThreshold := 1
);
ddierror2 := G_ISOBUS_TC.Client.SetDDIDistanceInterval(
    i_ObjectID := G_ISOBUS_TC_DDI.DDIList[G_ISOBUS_TC_Info.DDI_INDEX_TOTAL_DISTANCE_OBJ_1002].ObjectId,
    i_DistanceInterval := 1000
);
ddierror3 := G_ISOBUS_TC.Client.SetDDIThresholdLimits(
    i_ObjectID := G_ISOBUS_TC_DDI.DDIList[G_ISOBUS_TC_Info.DDI_INDEX_ACTUAL_WORKING_WIDTH_OBJ_1101].ObjectId,
    i_MinThreshold := 100,
    i_MaxThreshold := 200
);
ddierror4 := G_ISOBUS_TC.Client.SetDDITimeInterval(
    i_ObjectID := G_ISOBUS_TC_DDI.DDIList[G_ISOBUS_TC_Info.DDI_INDEX_TOTAL_TIME_OBJ_1001].ObjectId,
    i_TimeInterval := T#1S
);
```

Features, TECU

- Application data is located in *G_ISOBUS_TECU.Data*
- AUX valve command PGN (class 3 message) is not implemented at C3.5 library because TIM feature is not implemented. Only TECU Class 2 or lower is supported.
- In C2.3 TECU interface is able to connect only to one TECU instance defined in MultiTool
- In C3.5 MultiTool defines maximum TECU instance count and library is able to receive tractor facilities message from multiple TECU devices
- Library uses the TECU facilities messages to determine which signals are implemented by which TECU instance. Lowest TECU instance number is prioritized. Standard defines that secondary TECU shall not send signals which are available by primary instance.

Features, Diagnostics

- Configuration data is located in *G_ISOBUS_Diagnostics.Data*
- Data is automatically initialized by code template
- If data needs to be changed on-fly, the raw PGN data can be updated by PGN specific method after data has been changed in GVL



- CFF
  - Diagnostics.UpdateData_CFF (METH)
- DIAG
  - Diagnostics.UpdateData_DIAG (METH)
- ECUID
  - Diagnostics.UpdateData_ECUID (METH)
- ICC
  - Diagnostics.UpdateData_ICC (METH)
- PII
  - Diagnostics.UpdateData_PII (METH)
- SOFT
  - Diagnostics.UpdateData_SOFT (METH)

- DM1 PGN is not implemented by ISOBUS library. The DM log functionality will be added to J1939 library in future SDK and ISOBUS applications will use J1939 implementation when available.
  - Example applications temporarily include the DM PGN in application code for testing purposes

THANK YOU!

For questions,
contact techsupport@epec.fi

www.epec.fi