KNX Cookbook

KNX development on basis of existing system components

Components by Opternus

2

5

1

Summary

This document is a development help for KNX newcomers.

This document describes the development of a KNX device based on existing KNX components provided by the company Opternus.

This document is part of the KNX Specifications v2.1.

Document updates

Version	Date	Modifications
1.0.00	2011.03.28	Preparation of the final version.
1.01.01	2011.08.11	Update according final Manufacturer Tool release.
01.01.02	2013.10.14	Editorial updates for the publication of KNX Specifications 2.1.

References

[01] Chapter 3/1/2 “Glossary”
[02] Volume 6 “Profiles”

Filename:	02_05_01 Components by Opternus v01.01.02.docx
Version:	01.01.02
Status:	Final version
Savedate:	2013.10.14
Number of pages:	34

Contents

1 Developing applications on the basis of Opternus Components BIM M130, BIM M131, BIM M132 and BIM M135

1.1 Needed software

- Installed MT4 software and valid license.
- Installed ETS4 software and valid license.
- An Opternus Development Tool package.

1.2 About the Opternus Development Tool package

- This product is based on the BIM M132 functionality:
    - KNX Profile Class = System 2, see [02].
- The Opternus Development Tool package containing:
    - 1 development board (PCB)
    - NEC Minicube development platform for flash MCU
    - IAR Embedded Workbench (limited edition)
    - BIM Tools
- This solution requires development from scratch, as BIM Modules do not contain predefined Group Objects; i.e. the AP is developed in the C programming language.
- This solution comes with an own specific (not standardized) API (see document BIM_M_13x_API_Reference_v1_2_01.pdf).

2 Concept of the Opternus BIM M13x Solution

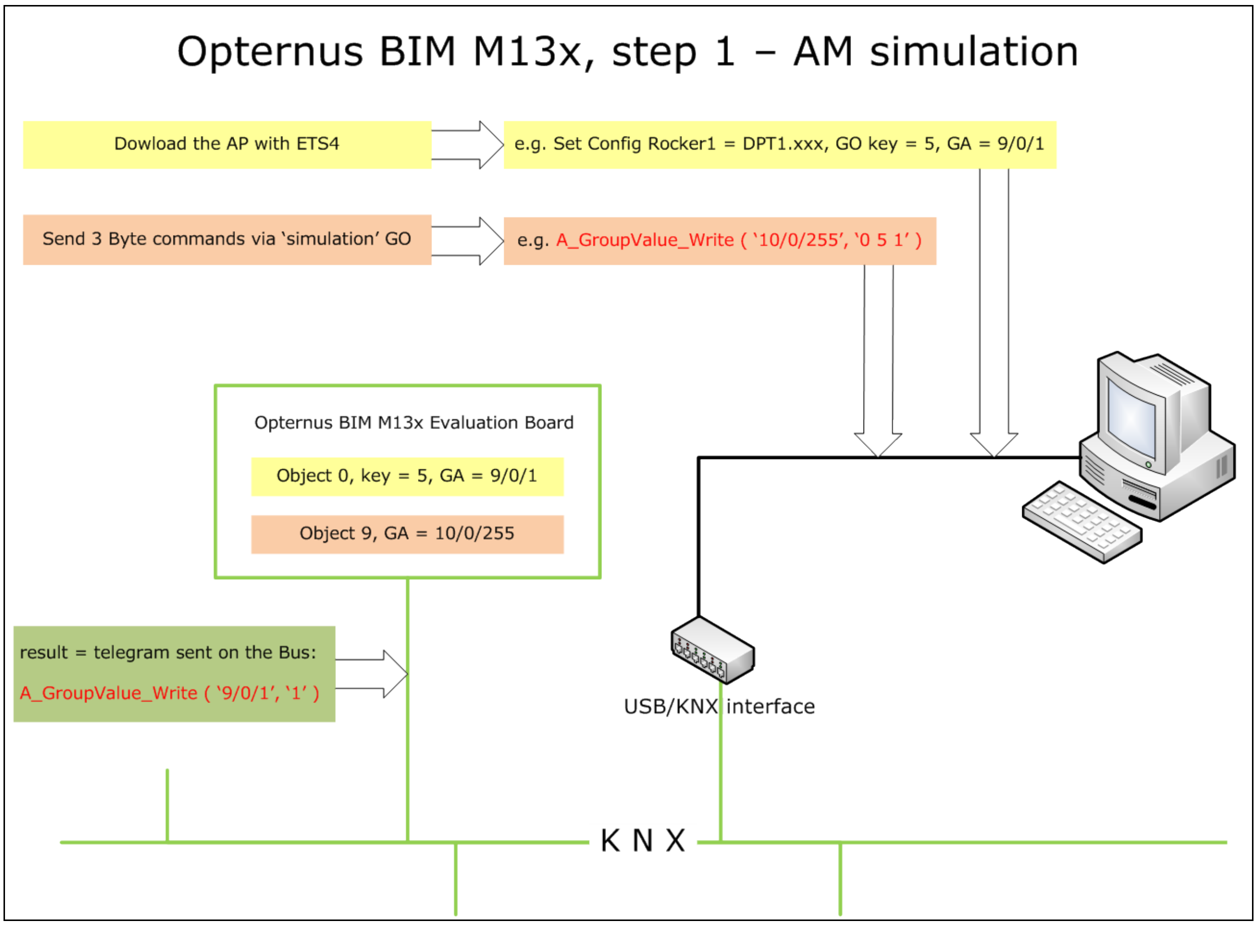
2.1 In general

- Develop a 'Debug' version of the AP with IAR and test.
- Then create a 'Release' version of the AP with IAR; this results in an s19 file.
- Create an MT4 project based on this s19 file.

2.2 Steps

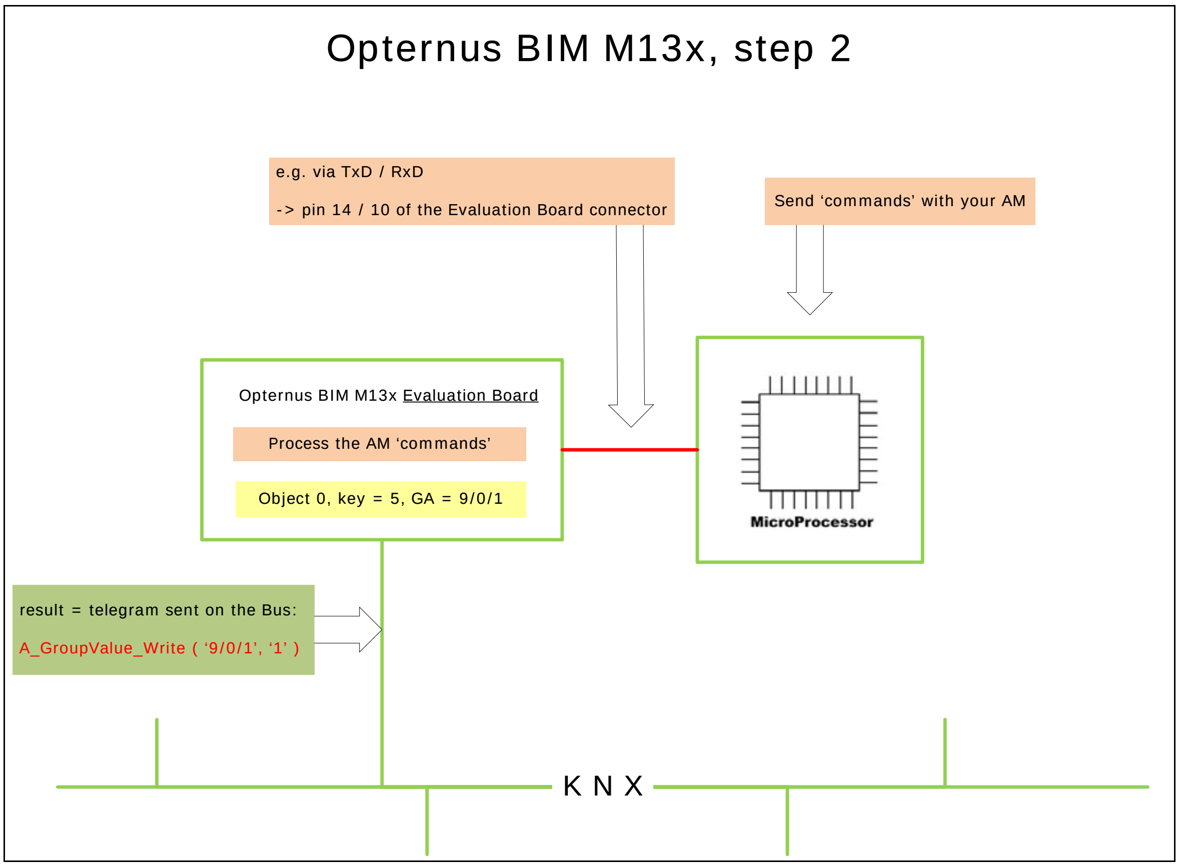
2.2.1 Step 1

- Use the BIM M13x Evaluation Board.
- Configure the GOs with ETS.
- Send Group Telegrams with ETS in order to simulate the AM.
- Check for all GOs.


Opternus BIM M13x, step 1 – AM simulation
Dowload the AP with ETS4
e.g. Set Config Rocker1 = DPT1.xxx, GO key = 5, GA = 9/0/1
Send 3 Byte commands via 'simulation' GO
e.g. A_GroupValue_Write ('10/0/255', '0 5 1')
Opternus BIM M13x Evaluation Board
Object 0, key = 5, GA = 9/0/1
Object 9, GA = 10/0/255
result = telegram sent on the Bus:
A_GroupValue_Write ('9/0/1', '1')
USB/KNX interface
K N X

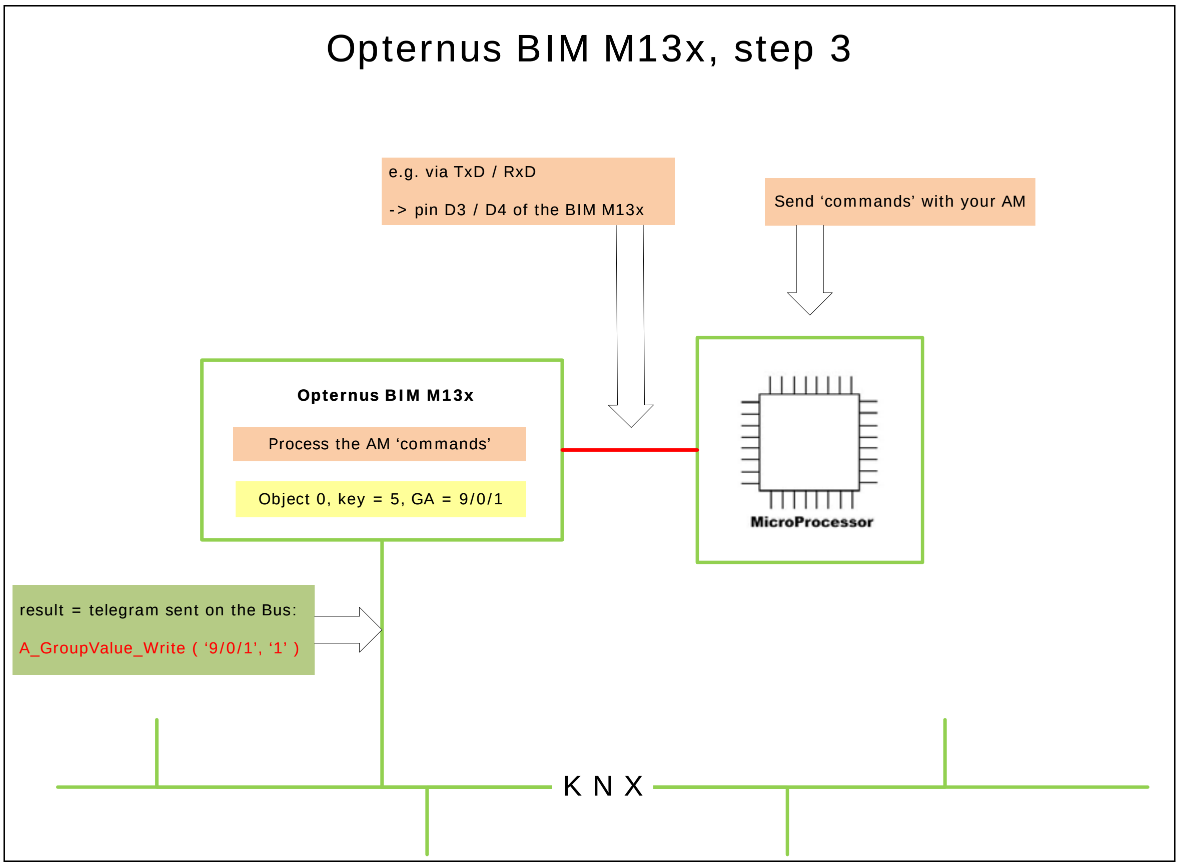
2.2.2 Step 2

- Use the BIM M13x Evaluation Board.
- Connect your AM e.g. via TxD / RxD; i.e. via pin 14 and pin 10 of the connector.
- Modify the AP of step 1; make sure to process the AM commands accordingly.
- Configure the GOs with ETS.
- Send commands with your AM.
- Check for all GOs.


Opternus BIM M13x, step 2
e.g. via TxD / RxD
-> pin 14 / 10 of the Evaluation Board connector
Send 'commands' with your AM
Opternus BIM M13x Evaluation Board
Process the AM 'commands'
Object 0, key = 5, GA = 9/0/1
MicroProcessor
result = telegram sent on the Bus:
A_GroupValue_Write ('9/0/1', '1')
K N X

2.2.3 Step 3

- Replace the evaluation device by a regular BIM M13x device.
- Connect your AM e.g. via TxD / RxD; i.e. via pin D3 and pin D4.
- Configure the GOs with ETS.
- Send commands with your AM.
- Check for all GOs.


Opternus BIM M13x, step 3
e.g. via TxD / RxD
-> pin D3 / D4 of the BIM M13x
Send 'commands' with your AM
Opternus BIM M13x
Process the AM 'commands'
Object 0, key = 5, GA = 9/0/1
MicroProcessor
result = telegram sent on the Bus:
A_GroupValue_Write ('9/0/1', '1')
K N X

2.3 About the AP (in step 1)

- As HW development is beyond the scope of this Cookbook, the Cookbook AP example shown here will simulate its own AM, i.e. by sending via the bus appropriate values to some Group Objects; other available Group Objects will generate Telegrams on the bus.
- The AP has 8 GOs: 4 x switching objects (DPT 1.xxx) and 4 x dimming relative objects (DPT 3.007).
- For each GO only one parameter is defined:
  - stored in EEPROM
  - length = 8 bits
  - serving as a kind of (GO) identifier, called 'GO key'

- The simulation of the AM is realized via an extra GO (i.e. 8 + 1): this extra GO will only be used for receiving data. It will have a data width of 3 bytes.
  - first byte = sending GO
  - second byte = GO key
  - third byte = value to be sent
  - GO key: if this (received) value corresponds with the parameter as stored in EEPROM for the GO indicated by the first received byte, a Telegram with value corresponding to the third received byte for this GO will be sent.

    EXAMPLE 1 Data received via the ‘simulation’ GO = 03h 0Ah 01h: if the stored GO key parameter for the third (03h) GO = 10 (0Ah) then a Telegram with value = 1 (01h) will be sent for this GO.

2.4 Phase 1: Evaluation

- See also 2.2.1 “Step 1” and 2.2.2 “Step 2”.
- Use the Opternus BIM M13x Evaluation Board.
- Develop the AP with IAR.
- Create an ETS4 product with MT4 in order to pre-configure the required GOs for the AP.
- Define the individual types.
- Set up the required parameters.
- Add the product to an ETS4 database/project
- Link its GOs with GAs.
- Set its parameters.
- Set its GOs flags.
- Program the device (with ETS via KNX).
- Use ETS4 Device Info to check whether the device was configured correctly.
- Use ETS4 Group Monitor to check the proper working of the AP.

2.5 Phase 2: AM integration

- See also 2.2.3 “Step 3”.
- At this stage, it will be necessary to get in contact with Opternus Components to be able to Replace the Evaluation Board with a BIM M130/M131/M132 or M135 device and integrate it with the AM.
- Check whether the same Telegrams are sent as during the evaluation phase.
- Certify the entire product, that is:
  - the HW, being the BIM M13x with the integrated AM, and
  - the AP.

3 In practice

3.1 Group Object vs. Communication Object

- First of all, the terms “Group Object” and “Communication Object” are synonyms.
- “Communication Object” is used in MT, ETS and other tools.
- “Group Object” is the only term used in the rest of the KNX Specifications and is therefore considered as the only correct one.
- Both terms will however be used in this Cookbook because it is here were practice and theory meet. “Communication Object” will only be used when absolutely necessary, e.g. when the context is MT.

3.2 Preparation

3.2.1 Install the Minicube2 driver, NEC Debugger and IAR Embedded Workbench

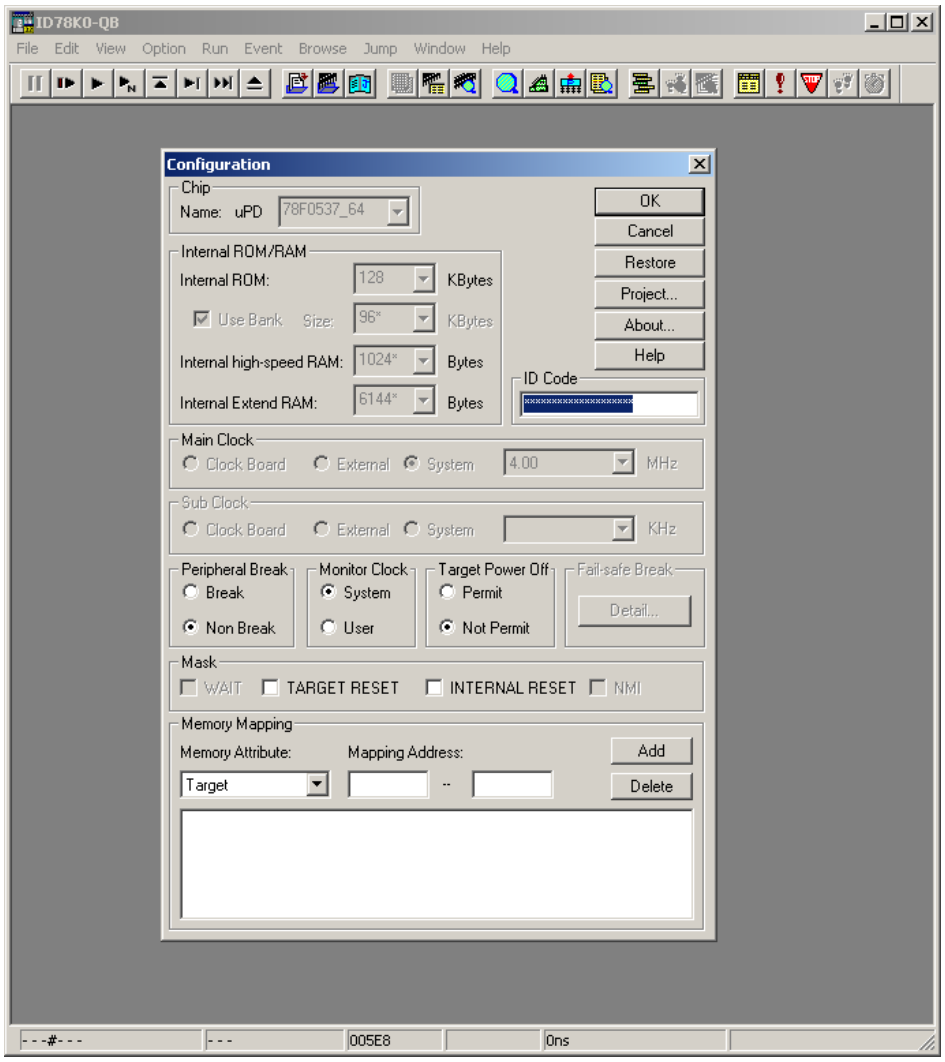
- Link: http://www.opternus.com/uploads/media/EVB_Getting_Started_V1.1.pdf.
- Install instructions for Minicube2 including NEC Debugger, see section I. of EVB_Getting_Started_V1.1.pdf.
- Install instructions for IAR Embedded Workbench, see section II. of EVB_Getting_Started_V1.1.pdf.

3.2.2 Install the BIM Tools

- Run BIM_Tools_v1_1_50.msi.

3.2.3 Connect & Prepare Device

- Connect: see section III. of EVB_Getting_Started_V1.1.pdf.
- Prepare: see section IV. of EVB_Getting_Started_V1.1.pdf.
- In this particular case, use BIM_M_130__M_135_Debug_v1_08.hex.



Chip
Name: uPD 78F0537_64

Internal ROM/RAM
Internal ROM: 128 KBytes
[✓ Use Bank] Size: 96* KBytes
Internal high-speed RAM: 1024* Bytes
Internal Extend RAM: 6144* Bytes

ID Code
xxxxxxxxxxxxxxxxxxxx

Main Clock
Clock Board External [✓ System] 4.00 MHz

Sub Clock
Clock Board External System KHz

Peripheral Break
Break [✓ Non Break]

Monitor Clock
[✓ System] User

Target Power Off
Permit [✓ Not Permit]

Fail-safe Break

Mask
WAIT TARGET RESET INTERNAL RESET NMI

Memory Mapping
Memory Attribute: Target
Mapping Address: --

---#--- --- 005E8 0ns

Figure 1 - NEC Debugger screenshot

3.3 KNX bus traffic in relation to the KNX stack and APs

3.3.1 When receiving a group Telegram from the bus

See Figure 2.

The system software (KNX Stack) does the following.

- It analyses the Group Address and the value of the Telegram.
- It determines which GO is involved, by consulting the following Resources:
  - the Group Address Table – this is the list of GAs -, and
  - the Group Object Association Table, which maps between the GAs and the GOs, and
  - the Group Object Table, which is the list of GOs and their descriptions.
- It updates the value for the involved GO in RAM. The consulted Resources are:
  - the Application Info Block to retrieve the pointer to CObjects, and
  - CObjects that contain the pointers for all GO values.
- Sets the Update Flag for the involved GO in the RAM Flags

The AP continuously polls the Update Flags. If at least one is set then it will

- fetch the GO value as stored in RAM, and
- control the PEI (if required and/or applicable), and
- reset the Update Flag in the RAM Flags.

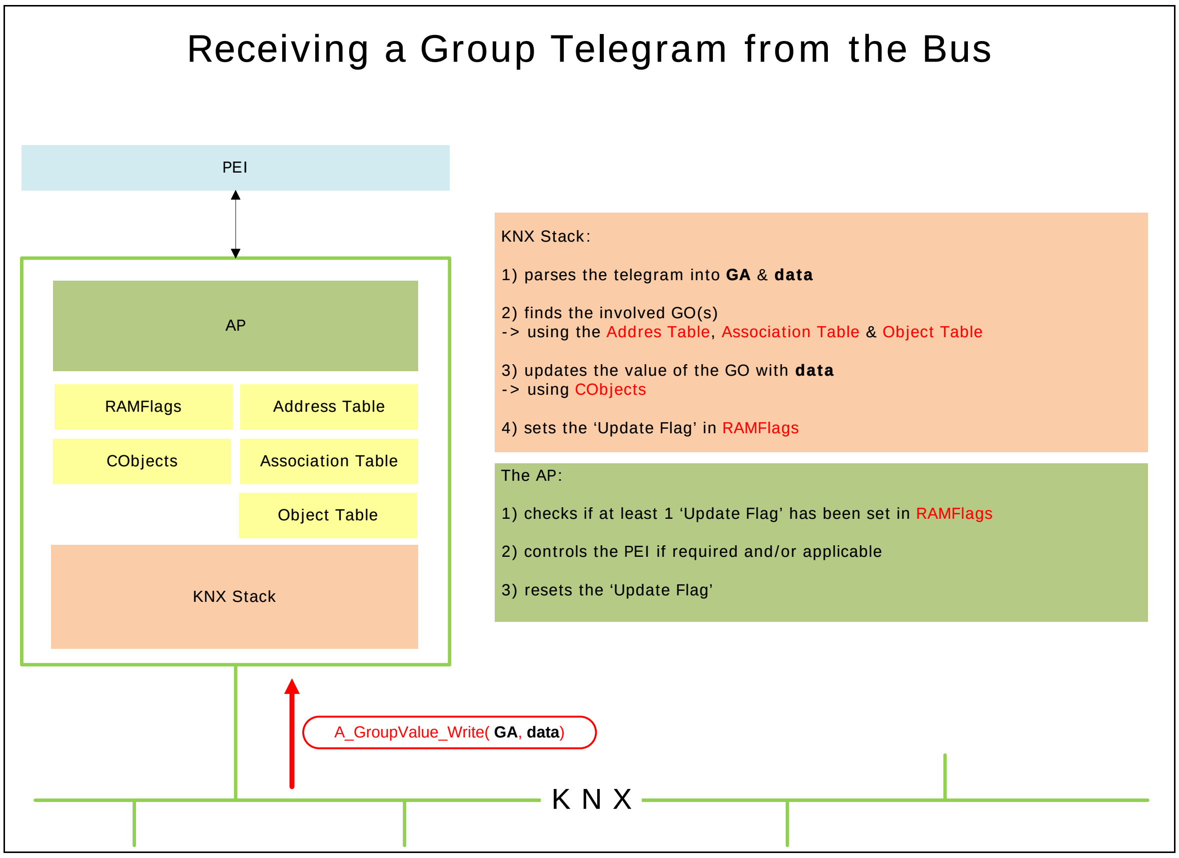

Receiving a Group Telegram from the Bus
PEI
AP
RAMFlags
Address Table
CObjects
Association Table
Object Table
KNX Stack
KNX Stack:
1) parses the telegram into GA & data
2) finds the involved GO(s)
-> using the Addres Table, Association Table & Object Table
3) updates the value of the GO with data
-> using CObjects
4) sets the 'Update Flag' in RAMFlags
The AP:
1) checks if at least 1 'Update Flag' has been set in RAMFlags
2) controls the PEI if required and/or applicable
3) resets the 'Update Flag'
A_GroupValue_Write(GA, data)
K N X

Figure 2 - Receiving a Group Telegram from the bus

3.3.2 When sending a group Telegram to the bus

See Figure 3

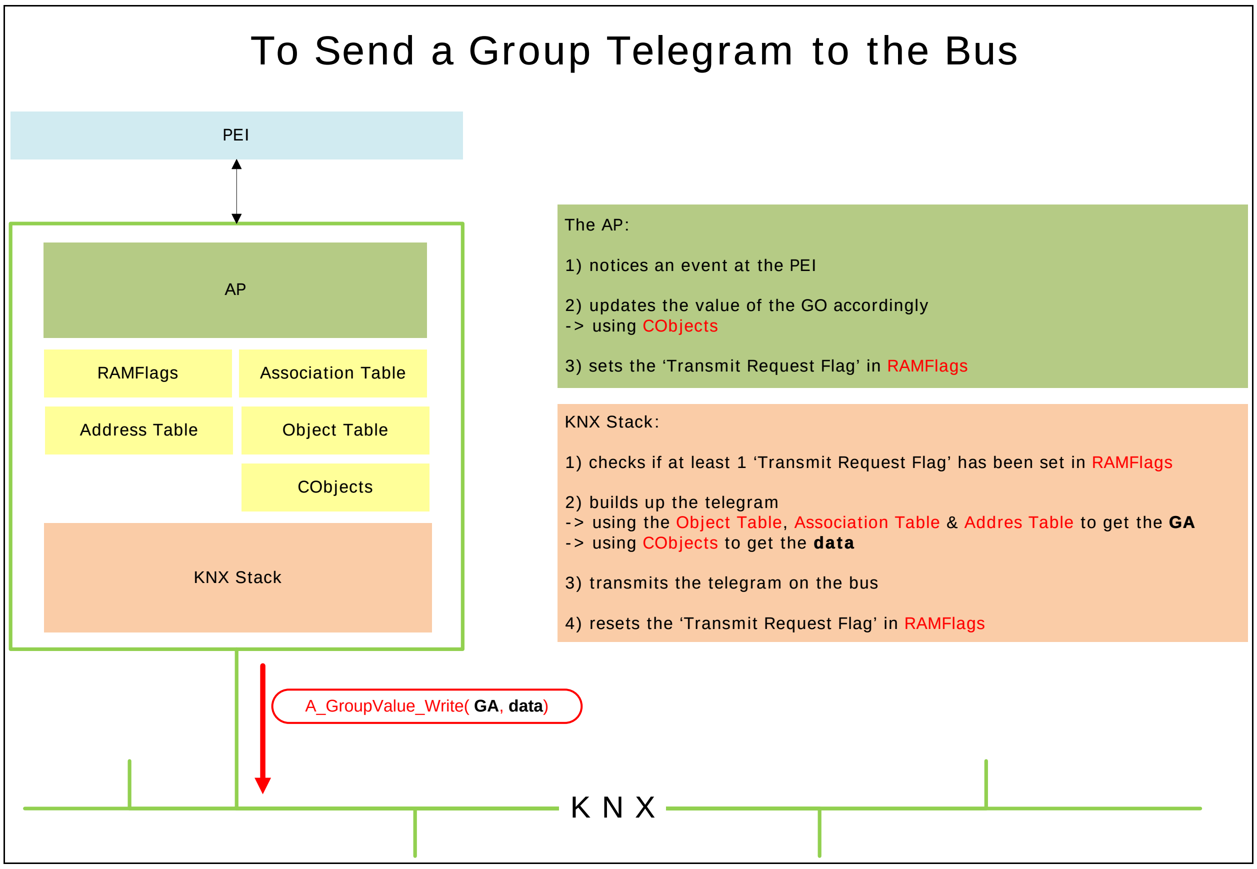
The AP:

- notices an event at the PEI
- updates the value for the involved GO in RAM, the consulted resources are:
  - the **Application Info Block** to retrieve the pointer to CObjects
  - **CObjects** contains the pointers for all GO values
- sets the Transmit Request Flag for the involved GO in the RAM Flags

The system software (KNX Stack) continuously polls the Transmit Request Flags, if at least one is set, then it will:

- generate the (group) Telegram, the consulted resources in order to find the appropriate GA are:
  - the **Object Table** = list with GO descriptions
  - the **Association Table** = mapping between GAs & GOs
  - the **Address Table** = list with GAs

- In order to find the appropriate data
  - the **Application Info Block** to retrieve the pointer to CObjects
  - **CObjects** contains the pointers for all GO values
- transmit the Telegram on the bus
- reset the Transmit Request Flag for the involved GO in the RAM Flags


To Send a Group Telegram to the Bus
PEI
AP
RAMFlags
Association Table
Address Table
Object Table
CObjects
KNX Stack
The AP:
1) notices an event at the PEI
2) updates the value of the GO accordingly
-> using CObjects
3) sets the 'Transmit Request Flag' in RAMFlags
KNX Stack:
1) checks if at least 1 'Transmit Request Flag' has been set in RAMFlags
2) builds up the telegram
-> using the Object Table, Association Table & Addres Table to get the GA
-> using CObjects to get the data
3) transmits the telegram on the bus
4) resets the 'Transmit Request Flag' in RAMFlags
A_GroupValue_Write(GA, data)
K N X

-

Figure 3 - To send a Group Telegram to the bus

3.4 AP Development with IAR

3.4.1 Introduction

- Start IAR.
- Download opternus01.zip.
- Extract the file and open template BIM M130 example by double clicking BIM_M_130_Example_BasicKit.eww

3.4.2 The complete file main.c

```
/*******************************************************************************
*
* File: main.c
*
*
*******************************************************************************/
#include "BIM_M13x.h"

// ram flags for communication objects
BYTE RAMFlags[(NUM_OF_GOM_OBJ / 2) + 1];

// communication object values
BYTE   GO_Value_0;
BYTE   GO_Value_1;
BYTE   GO_Value_2;
BYTE   GO_Value_3;
BYTE   GO_Value_4;
BYTE   GO_Value_5;
BYTE   GO_Value_6;
BYTE   GO_Value_7;
BYTE   GO_Value_8;

/*******************************************************************************/

// data received for object8 (3 bytes)
BYTE byte0;
BYTE byte1;
BYTE byte2;

// 'keys' for the individual objects
#pragma constseg = PARAM
__root const BYTE Key[8];

void AppInit(void)
{
   // after initialization a taskswitch must be done because
   // the task switch is disabled at cstartup
   FORCE_TASKSWITCH();
}

//-----------------------------------------------------------------------
//  This is the main function of the application program
//  It is called at the end of cstartup
//-----------------------------------------------------------------------
void main (void)
{
   // Do some initializations
   AppInit();


   // Start the main loop of the application program
   // it will be interrupted every 3.3ms and
   // continued after a complete system cycle

   BYTE Data_Obj8[3];
```

```
    for(;;)
    {
        if (U._TestAndCopyObject(GOM_OBJ8, Data_Obj8, 3) == TRUE)
        {
          byte0 = Data_Obj8[0];
          byte1 = Data_Obj8[1];
          byte2 = Data_Obj8[2];

          if (byte1 == Key[byte0])
              U._SetAndTransmitObject(byte0, &byte2, 1);
        }
    }
}

//------------------------------------------------------------------------
//  This function is called on loss of power
//------------------------------------------------------------------------
void save()
{
}

//------------------------------------------------------------------------
//  This function is called if there was an
//  unload of the application program
//------------------------------------------------------------------------
void unload()
{
}

const BYTE NbComsOb = NUM_OF_GOM_OBJ;
const GObjPtr GObjects[] =      // set table for com objects
{
    (void*)&NbComsOb,           // ptr to number of cobjects
    &GO_Value_0,
    &GO_Value_1,
    &GO_Value_2,
    &GO_Value_3,
    &GO_Value_4,
    &GO_Value_5,
    &GO_Value_6,
    &GO_Value_7,
    &GO_Value_8
};

//------------------------------------------------------------------------
//  In the application info block the application program gives some
//  necessary information for the operation system
//------------------------------------------------------------------------
#pragma constseg = APPINFOBLOCK
const AppInfoBlock AIB =
{
    Swap(0x0001),                 // AIBVersion
    0x01, 0x01,                 // ApplFirmwareVersion, ApplFirmwareSubVersion
```

```
   __program_start,              // AppMain
   save,                         // AppSave
   unload,                       // AppUnload
   GObjects,                     // pGObjects
   RAMFlags,                     // pRAMFlags
   NULL,                         // pUserTimerTab
   NULL,                         // pUsrIntObjRoot
   NULL,                         // pUsrParamMgmt
   0x0000                        // WatchDogTime
};
#pragma constseg = default
```

3.4.3 Detailed explanation of main.c

```
// ram flags for communication objects
BYTE RAMFlags[(NUM_OF_GOM_OBJ / 2) + 1];

// communication object values
BYTE   GO_Value_0;
BYTE   GO_Value_1;
BYTE   GO_Value_2;
BYTE   GO_Value_3;
BYTE   GO_Value_4;
BYTE   GO_Value_5;
BYTE   GO_Value_6;
BYTE   GO_Value_7;
BYTE   GO_Value_8;
```

- This reserves the required RAM for the GOs of this AP; both flags as well as values.
- This is the operational (live) data for the GOs.
- The actual definitions of the GOs are stored in EEPROM and will be set up in tables.c (see further down) in a resource called **Object Table**.
- The address of RAMFlags needs to be specified in the **Application Info Block**.

```
// data received for object8 (3 bytes)
BYTE byte0;
BYTE byte1;
BYTE byte2;
```

- This reserves 3 bytes in RAM in order to store the data received via object 8.
- The exact memory locations depend on settings in BIM_M_130.xcl.

```
#pragma constseg = PARAM
__root const BYTE Key[8];
```

- This reserves 8 bytes in EEPROM in order to store the GO key parameters
- The exact memory locations depend on settings in BIM_M_130.xcl.

```
void main (void)
```

- Contains the actual logic of the AP.

```
  for(;;)
  {
    if (U._TestAndCopyObject(GOM_OBJ8, Data_Obj8, 3) == TRUE)
    {
      byte0 = Data_Obj8[0];
      byte1 = Data_Obj8[1];
      byte2 = Data_Obj8[2];

      if (byte1 == Key[byte0])
        U._SetAndTransmitObject(byte0, &byte2, 1);
    }
  }
```

This is the actual functionality of the AP: in this example is kept fairly simple.

- It waits for un update on GO 8 (GOM_OBJ8) – i.e. TestAndCopyObject checks if (in this case) the RAMFlag for GO 8 has been set. If so, then it will copy 3 bytes into Data_Obj8.
- The received value is then stored in the three RAM variables: byte0, byte1 and byte2.
- Byte1 is then evaluated against the GO key, which is stored in EEPROM as parameter (Key[]).
- If there is a match a Telegram for the GO in question (byte0) will be sent; this is; SetAndTransmitObject sets the RAMFlag for the GO indicated by byte0 and send its value to 1 byte of byte2.

```
const BYTE NbComsOb = NUM_OF_GOM_OBJ;
```

- This reserves 1 byte in EEPROM in order to store the number of GOs.

```
const GObjPtr GObjects[] =      // set table for com objects
{
  (void*)&NbComsOb,          // ptr to number of cobjects
  &GO_Value_0,
  &GO_Value_1,
  &GO_Value_2,
  &GO_Value_3,
  &GO_Value_4,
  &GO_Value_5,
  &GO_Value_6,
  &GO_Value_7,
  &GO_Value_8
};
```

- This reserves the necessary EEPROM in order to store/set up a table, which consists of:
    - a pointer to where the number of GOs is stored, this is a pointer to EEPROM, and
    - pointers to where the **GO values** are stored, these are pointers to RAM.
- The address of GObjects needs to be specified in the **Application Info Block**.

```
#pragma constseg = APPINFOBLOCK
const AppInfoBlock AIB =
{
   Swap(0x0001),                // AIBVersion
   0x01, 0x01,                  // ApplFirmwareVersion, ApplFirmwareSubVersion
   __program_start,             // AppMain
   save,                        // AppSave
   unload,                      // AppUnload
   GObjects,                    // pGObjects
   RAMFlags,                    // pRAMFlags
   NULL,                        // pUserTimerTab
   NULL,                        // pUsrIntObjRoot
   NULL,                        // pUsrParamMgmt
   0x0000                       // WatchDogTime
};
```

- This reserves the necessary EEPROM in order to store the **Application Info Block**.
- This information stores amongst other
    - the starting address of the AP
    - pointers to the save & unload routines
    - pointer to the RAM flags
    - pointer to the table containing the GO value pointers (= GObjects)
- Remark: this AP does not require any timers, therefore ‘pUserTimerTab’ has been set to NULL.

3.4.4 The complete file tables.c

```
/******************************************************************************
*
* File: tables.c
*
* Description: This module contains the address table, association table and
*              communication object table.
*
******************************************************************************/
#include “BIM_M13x.h”

#pragma constseg = ADDRTAB
__root const BYTE AdrTab[] =
{    NUM_OF_GOM_OBJ + 1,   // for debug: one Group Address per communication object
                      // (Group Addresses must be sorted in ascending order!)
     0xFF, 0xFF,          // physical address
```

```
    // Group Addresses
    0x48, 0x01,
    0x48, 0x02,
    0x48, 0x03,
    0x48, 0x04,
    0x49, 0x01,
    0x49, 0x02,
    0x49, 0x03,
    0x49, 0x04,
    0x50, 0xFF,
    0x00                // padding
};
#pragma constseg = ASSOCTAB
__root const BYTE AssTab[] =
{   NUM_OF_GOM_OBJ + 0,   // for debug: one association per communication object
                      // (sorted in ascending order by the second byte of the
                      //  association. If a communication object has no
                      //  association then association has to be (0xFE, [objnum]).
                      //  A communication object has only one sending Group Address.
                      //  This is the Group Address that is specified by the
                      //  association at the “communication object number”-position
                      //  in the association table.)
//   grp.addr.,  comobj. Num.,
       0x01,        0x00,
       0x02,        0x01,
       0x03,        0x02,
       0x04,        0x03,
       0x05,        0x04,
       0x06,        0x05,
       0x07,        0x06,
       0x08,        0x07,
       0x09,        0x08,
       0x00                // padding
};
#pragma constseg = GOMTAB
__root const BYTE EE_CommsTab[] =
{   NUM_OF_GOM_OBJ,      // number of communication objects,
          0x00,      // pointer to RAMFlags is not longer used
                      // but must be specified for compatibility
/*Obj0*/ NULL, (GOMM_ENABLE + WRITE_ENABLE + TRANSMIT_ENABLE + READ_ENABLE + TX_LOW),
       UINT1,
/*Obj1*/ NULL, (GOMM_ENABLE + WRITE_ENABLE + TRANSMIT_ENABLE + READ_ENABLE + TX_LOW),
       UINT1,
/*Obj2*/ NULL, (GOMM_ENABLE + WRITE_ENABLE + TRANSMIT_ENABLE + READ_ENABLE + TX_LOW),
       UINT1,
/*Obj3*/ NULL, (GOMM_ENABLE + WRITE_ENABLE + TRANSMIT_ENABLE + READ_ENABLE + TX_LOW),
       UINT1,
/*Obj4*/ NULL, (GOMM_ENABLE + WRITE_ENABLE + TRANSMIT_ENABLE + READ_ENABLE + TX_LOW),
       UINT4,
/*Obj5*/ NULL, (GOMM_ENABLE + WRITE_ENABLE + TRANSMIT_ENABLE + READ_ENABLE + TX_LOW),
       UINT4,
/*Obj6*/ NULL, (GOMM_ENABLE + WRITE_ENABLE + TRANSMIT_ENABLE + READ_ENABLE + TX_LOW),
       UINT4,
/*Obj7*/ NULL, (GOMM_ENABLE + WRITE_ENABLE + TRANSMIT_ENABLE + READ_ENABLE + TX_LOW),
          UINT4,
/*Obj8*/ NULL, (GOMM_ENABLE + WRITE_ENABLE + READ_ENABLE + TX_LOW), 0x09 /* 3 bytes */

};
#pragma constseg = default
```

3.4.5 Detailed explanation table.c

```
#pragma constseg = ADDRTAB
__root const BYTE AdrTab[] =
{     NUM_OF_GOM_OBJ + 1,   // for debug: one Group Address per communication object
                        // (Group Addresses must be sorted in ascending order!)
      0xFF, 0xFF,          // physical address
      // Group Addresses
      0x48, 0x01,
      0x48, 0x02,
      0x48, 0x03,
      0x48, 0x04,
      0x49, 0x01,
      0x49, 0x02,
      0x49, 0x03,
      0x49, 0x04,
      0x50, 0xFF,
      0x00                // padding
};
```

- This reserves the required EEPROM for the **Address Table** of this AP.
  - The first two bytes are reserved for the Individual Address IA of the BCU.
  - The last byte is always 0x00 in order to indicate the end of this table.
  - The GAs for this AP are stored between the IA and the ‘end of table’ byte.
  - In case the AP requires more Gas, they need to be added between the IA and the last byte, two bytes per extra GA.

```
#pragma constseg = ASSOCTAB
__root const BYTE AssTab[] =
{     NUM_OF_GOM_OBJ + 0,   // for debug: one association per communication object
                        // (sorted in ascending order by the second byte of the
                        //  association. If a communication object has no
                        //  association then association has to be (0xFE, [objnum]).
                        //  A communication object has only one sending Group Address.
                        //  This is the Group Address that is specified by the
                        //  association at the “communication object number”-position
                        //  in the association table.)
//    grp.addr.,  comobj. Num.,
          0x01,         0x00,
          0x02,         0x01,
          0x03,         0x02,
        0x04,       0x03,
        0x05,       0x04,
        0x06,       0x05,
        0x07,       0x06,
        0x08,       0x07,
        0x09,       0x08,
        0x00            // padding
};
```

- This reserves the required EEPROM for the **Association Table** of this AP.
- This table defines the links between the GAs and the GOs.
- The last byte is always 0x00 in order to indicate the end of this table.
- Each ‘association’ contains two bytes:
  - first = index of the GA in the **Address Table**, and
  - second = index of the GO in the **Object Table**.
- In case the AP requires more associations, they need to be added before the ‘end of table’ byte, two bytes per extra association.

```
#pragma constseg = GOMTAB
__root const BYTE EE_CommsTab[] =
{    NUM_OF_GOM_OBJ,      // number of communication objects,
           0x00,      // pointer to RAMFlags is not longer used
                      // but must be specified for compatibility
/*Obj0*/ NULL, (GOMM_ENABLE + WRITE_ENABLE + TRANSMIT_ENABLE + READ_ENABLE + TX_LOW),
        UINT1,
/*Obj1*/ NULL, (GOMM_ENABLE + WRITE_ENABLE + TRANSMIT_ENABLE + READ_ENABLE + TX_LOW),
        UINT1,
/*Obj2*/ NULL, (GOMM_ENABLE + WRITE_ENABLE + TRANSMIT_ENABLE + READ_ENABLE + TX_LOW),
        UINT1,
/*Obj3*/ NULL, (GOMM_ENABLE + WRITE_ENABLE + TRANSMIT_ENABLE + READ_ENABLE + TX_LOW),
        UINT1,
/*Obj4*/ NULL, (GOMM_ENABLE + WRITE_ENABLE + TRANSMIT_ENABLE + READ_ENABLE + TX_LOW),
        UINT4,
/*Obj5*/ NULL, (GOMM_ENABLE + WRITE_ENABLE + TRANSMIT_ENABLE + READ_ENABLE + TX_LOW),
        UINT4,
/*Obj6*/ NULL, (GOMM_ENABLE + WRITE_ENABLE + TRANSMIT_ENABLE + READ_ENABLE + TX_LOW),
        UINT4,
/*Obj7*/ NULL, (GOMM_ENABLE + WRITE_ENABLE + TRANSMIT_ENABLE + READ_ENABLE + TX_LOW),
        UINT4,
/*Obj8*/ NULL, (GOMM_ENABLE + WRITE_ENABLE + READ_ENABLE + TX_LOW), 0x09 /* 3 bytes */

};
```

- This reserves the required EEPROM for the **Object Table** of this AP.
- This table contains the actual definitions of the individual GOs.
  - The first byte contains the number of GOs.
  - The second byte is reserved and needs to be set to 0x00 (because of compatibility reason).

- Each GO definition contains 3 bytes
  - The first byte is not used (set to NULL).
  - The second byte is the config byte; it is used to define the communication flags (CRWTU) and the priority (system, high, low, normal).
  - Third byte defines the type (length) of the GO.

3.4.6 Detailed explanation user.h

```
/******************************************************************************
*
* File: user.h
*
* Description: The defines in this file must be adjusted to the
             application program
*
******************************************************************************/


#ifndef __USER_H
#define __USER_H

#define NUM_OF_TIMERS   0
#define NUM_OF_GOM_OBJ  9

#define GOM_OBJ0 0
#define GOM_OBJ1 1
#define GOM_OBJ2 2
#define GOM_OBJ3 3
#define GOM_OBJ4 4
#define GOM_OBJ5 5
#define GOM_OBJ6 6
#define GOM_OBJ7 7
#define GOM_OBJ8 8

#endif
```

- As mentioned before this AP does not have any timers. NUM_OF_TIMERS is therefore set 0.
- This AP has 9 GOs. NUM_OF_GOM_OBJ is therefore set to 9.

3.4.7 Reserved Memory

The reserved memory locations in RAM (variables) and EEPROM (parameters) are as mentioned before defined in the file BIM_M_130.xcl.

```
/*******************************************************************\
|*  D A T A                                                       *|
\*******************************************************************/
-Z(DATA)CSTACK+_CSTACK_SIZE#FBCF
-Z(DATA)NEAR_I,NEAR_Z,NEAR_N=FB00-FBCF

/*******************************************************************\
|*  C O D E                                                       *|
\*******************************************************************/
```

```
-Z(GODE)PARAM=9C00-9FFF
-Z(GODE)NEAR_ID,GONST,RGODE,GODE=8416-9BFF

-Z(GODE)GOMTAB=8316-8415
-Z(GODE)ASSOCTAB=8216-8315
-Z(GODE)ADDRTAB=8116-8215

-Z(GODE)APPINFOBLOCK=8000-8115
-Z(GODE)ROOTGODE=7800-7FFF
/*********************************************************************/
-Z(GODE)BCU2_JMP=7700-77FF
```

One example of how these variables are used in e.g. main.c:

```
#pragma constseg = PARAM
__root const BYTE Key[8];
```

3.4.8 Debug

Make sure that the 'Debug' option is selected in the Workspace window.

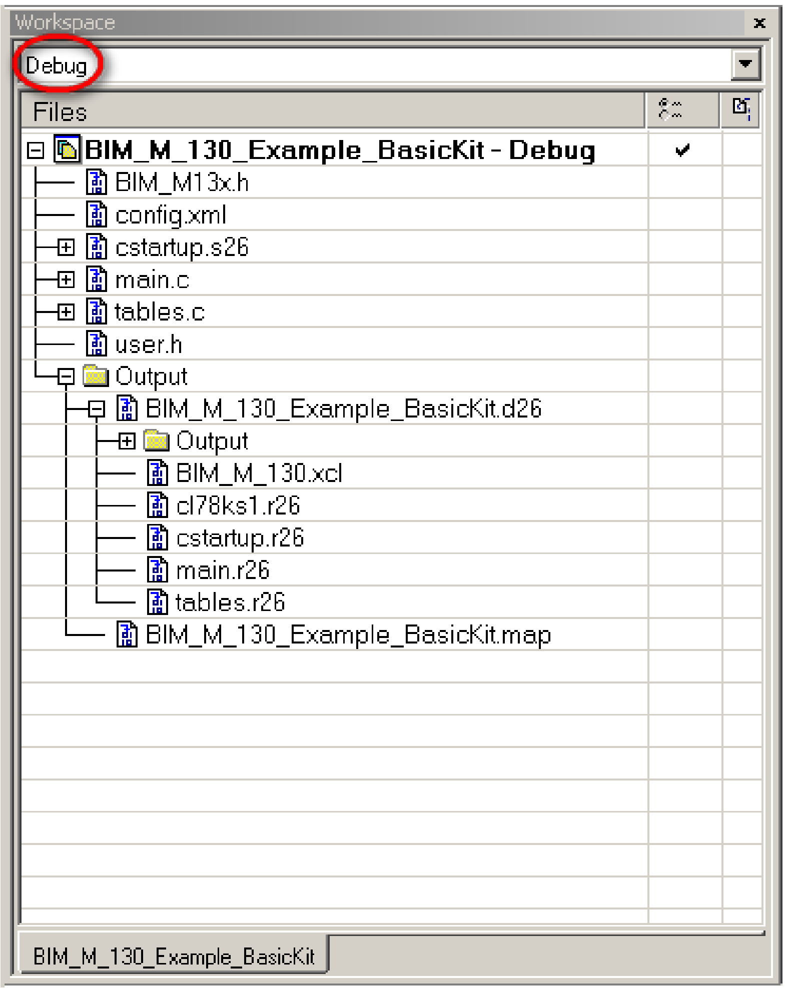


Figure 4 - Selecting debug in Workspace window

Select Project/Make via the menu to compile and link the project.

Select Project/Debug via the menu in order to start the preparations for a debug session.

Select Debug/Go via the menu to actually start the AP.

This AP should as such work and does not require any real debugging unless you would like to test it with other (hard-coded) GAs as the ones specified in the Address Table.

3.4.9 Release

Make sure that the 'Release' option is selected within the Workspace window.

Select Project/Make via the menu to build the project; this will create the file BIM_M_130_Example_BasicKit.map in the Release\List folder of this project.

As its content is too large to be copied here, only the most important part is shown here.

```
   -----------------------------------------------------------------------
PARAM
  Relative segment, address: 9C00 - 9C07 (0x8 bytes), align: 0
  Segment part 18. ROOT.      Intra module refs:   main
         ENTRY                 ADDRESS        REF BY
         =====                 =======        ======
         Key                   9C00
   -----------------------------------------------------------------------
```

This means that the 8 GO keys of the AP will be stored from address 9C00 onwards, this is from 0x9C00 to 0x9C07.

Project/Make will also create the file BIM_M_130_Example_BasicKit.s19 in the Release\Exe folder of this project.

This is its content:

```
S00C00003F435354415254455503E
S1138000000101011684A784A8840A9C00FB0000D7
S109801000000000000066
S11381160AFFFF4801480248034804490149024945
S109812603490450FF00B0
S1138216090100020103020403050406050707060812
S10782260709080038
S11383160900005F00005F00005F00005F00005F6F
S110832603005F03005F03005F03001F09F5
S1138416712AE661D01600FB101100D23261613276
S1138426AD0143617197868AFC8BFA0200809A51EA
S113843684B7AFFAFE710B6F710A6F712AE2712B62
S1138446E6000000000000000000AFB3B5B7B79A1D
S11384563B84A503891CD2100800B3D2020477D644
S1138466C2B29A378451BDEA891CD6879E0EFB8612
S1138476879E0FFBAE019E10FB8E0EFB7316009CAF
S1138486AB708E0FFB6148BDC9A5011210FB8E0EA1
S1138496FB70A100B3D2020677D6C2B29A3784FA29
S10684A6B1AFAFC0
S1139C0000000000000000000900089C05FB06FBA2
S1119C1007FB08FB09FB0AFB0BFB0CFB0DFB1F
```

```
S903841662
```

As such this file is correct but not usable for the MT, as it needs to be modified first.

This can be done automatically via a Post-build command line, by doing the following.

- Click Project/Options
- Click Build Actions
- Copy this text: "C:\Program Files\Siemens AG\BIM Tools\aioc.exe" -t modifier --targs, into Post-build command line.

Building the project will from now on create two extra files in the Release\Exe folder of this project:

- BIM_M_130_Example_BasicKit_mod.log, and
- BIM_M_130_Example_BasicKit_mod.s19.

BIM_M_130_Example_BasicKit_mod.s19 is the actual modified s19 file and is directly usable in the MT, with the following content.

```
S01D000042494D5F4D5F3133305F4578616D706C655F42617369634B6974D7
S113000000F00000000000000000000000000000FC
S113E800000101011684A784A8840A9C00FB00006F
S109E810000000000000FE
S10401160ADA
S1130119480148024803480449014902490349047A
S106012950FF0080
S11302160901000201030204030504060507060892
S107022607090800B8
S11303160900005F00005F00005F00005F00005FEF
S110032603005F03005F03005F03001F0975
S1130416712AE661D01600FB101100D232616132F6
S1130426AD0143617197868AFC8BFA0200809A516A
S113043684B7AFFAFE710B6F710A6F712AE2712BE2
S1130446E6000000000000000000AFB3B5B7B79A9D
S11304563B84A503891CD2100800B3D2020477D6C4
S1130466C2B29A378451BDEA891CD6879E0EFB8692
S1130476879E0FFBAE019E10FB8E0EFB7316009C2F
S1130486AB708E0FFB6148BDC9A5011210FB8E0E21
S1130496FB70A100B3D2020677D6C2B29A3784FAA9
S10604A6B1AFAF40
S1131C0000000000000000000900089C05FB06FB22
S1111C1007FB08FB09FB0AFB0BFB0CFB0DFB9F
S113F0000E000000000000000000000000000000EE
S113F01014000000000000000000000000000000D8
S113F02024000000000000000000000000000000B8
S113F0303400000000000000000000000000000098
S113F04011000000000000000000000000000000AB
S113F05013000001160155330300000000000000F6
S113F0601302000116000000000000000000000070
S113F070210000000000000000000000000000006B
S113F0802300000156019333030000000000000038
S113F09023020001560000000000000000000000F0
S113F0A0310000000000000000000000000000002B
```

```
S113F0B0330000E000E915330300000000000000005
S113F0C033000001941FFF330300000000000000020
S113F0D03302000000FF000112340100000000000B0
S113F0E03305000000019400000000000000000004F
S113F0F053020001940001000100000000000000020
S113F10012000000000000000000000000000000E9
S113F11022000000000000000000000000000000C9
S113F12032000000000000000000000000000000A9
S113F1300F000000000000000000000000000000BC
S113F140FF000000000000000000000000000000BC
S9030000FC
```

Not only the content of the first part has indeed been modified, it also has become longer, as the necessary load controls have been added.

BIM_M_130_Example_BasicKit_mod.log gives a summary of the modifications applied compared to BIM_M_130_Example_BasicKit.s19, with the following content:

```
BIM-Tools v1.1.50.27121

Start reading MEMORYGONVERSION entries ...
MEMORYGONVENTRY: 7800 <-> 8115 Offset: +6800
MEMORYGONVENTRY: 8116 <-> 8116 Offset: -8000
MEMORYGONVENTRY: 8119 <-> 9FFF Offset: -8000
... finished reading MEMORYGONVERSION!

Start reading LOADGONTROLS entries ...
Read load control: GONNECT
Read load control: UNLOAD
Read load control: UNLOAD
Read load control: UNLOAD
Read load control: LOAD
Read load control: DATASEGMENT
Read load control: TABLEPOINTER
Read load control: LOAD
Read load control: DATASEGMENT
Read load control: TABLEPOINTER
Read load control: LOAD
Read load control: DATASEGMENT
Read load control: DATASEGMENT
Read load control: APPDATA
Read load control: TABLEPOINTER
Read load control: LOADGOMPLETED
Read load control: LOADGOMPLETED
Read load control: LOADGOMPLETED
Read load control: DISGONNECT
... finished reading LOADGONTROLS!

Start parsing "Z:\tasks\Cookbook\opternus\CP\test03\Release\Exe\BIM_M_130_Example_BasicKit.s19" ...
...finished!

Start creating "Z:\tasks\Cookbook\opternus\CP\test03\Release\Exe\BIM_M_130_Example_BasicKit_mod.s19" ...
```

```
        Start converting memory ...
        ... finished!
        Start creating load controls ...
        ... finished!
... finished!
```

This is important as it is necessary to know where the parameters for the AP will located in EEPROM

The log file shows that the offset for the memory locations have been changed:

```
MEMORYGONVENTRY: 7800 <-> 8115 Offset: +6800
MEMORYGONVENTRY: 8116 <-> 8116 Offset: -8000
MEMORYGONVENTRY: 8119 <-> 9FFF Offset: -8000
```

From BIM_M_130_Example_BasicKit.map it is known that the original location for Key = 9C00 (see above). This means, because of

```
MEMORYGONVENTRY: 8119 <-> 9FFF Offset: -8000
```

that the actual location for these (ETS) parameters shall be calculated as 0x9C00 – 0x8000, this is 0x1C00.

3.5 Creating an ETS4 product entry with MT4

3.5.1 Start

- Add Project
  - KNX Manufacturer Tool Project
  - Name = KNXproduct100203-01
  - Target ETS Version = ETS4
- Add Hardware
  - Serial Number = SN17.06.2010-01
  - Name = PCB SN17.06.2010-01
  - Version = 1
- Define Product (= commercial realization of HW)
  - Order Number = 4f-PB SN17.06.2010-01
  - Product Name = 4 fold Push Button V1.0
  - Language
- Add Application Program
  - Number = 1 & Version = 1
  - Name = 4f Push Button
  - Mask Version = 2.5
- Open Hardware

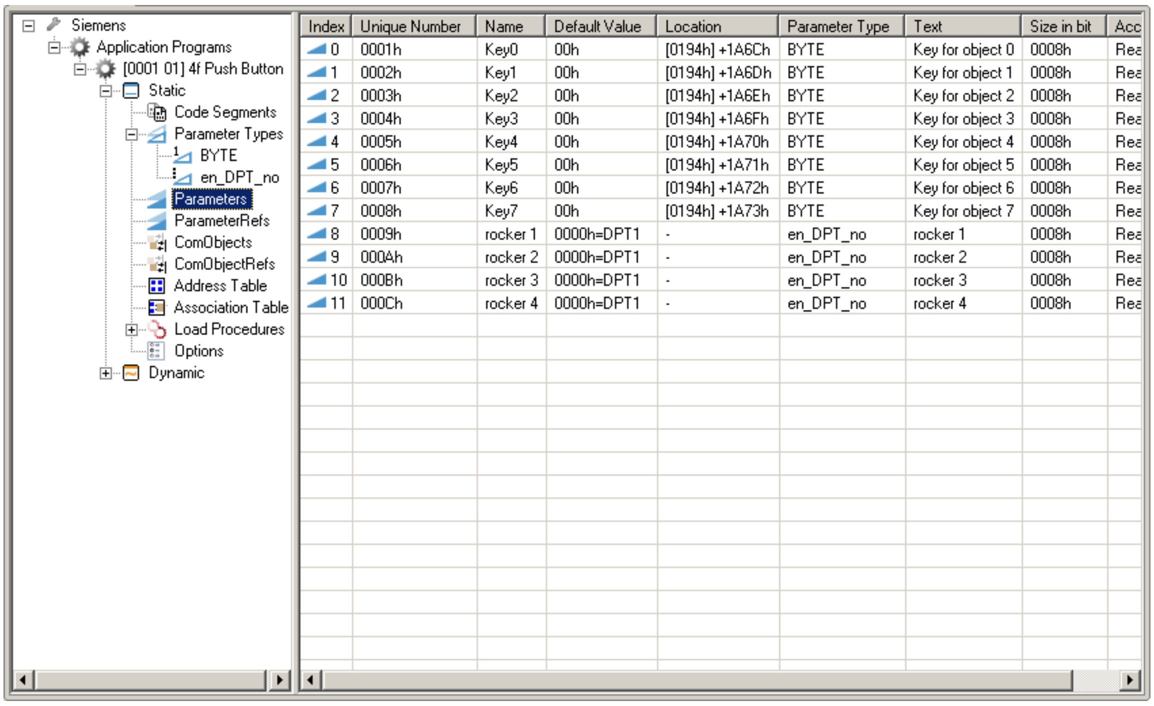
  - select Application Programs
  - add new Hardware2Program
  - Select 4f Push Button
- AP Properties
  - Dynamic Table Management = True
- Open the AP, in static
  - Import binary data -> BIM_M_130_Example_BasicKit_mod.s19, this will:
- Add Code Segments
  - Segment 0: Address = 0116h, Size = 0040h
  - Segment 1: Address = 0156h, Size = 003Eh
  - Segment 2: Address = E000h, Size = 0916h
  - Segment 3: Address = 0194h, Size = 1E6Ch
- Add Address Table
  - Code Segment -> [0116h], Max Entries = 001Eh, Offset=0
- Add Association Table
  - Code Segment -> [0156h], Max Entries = 001Eh, Offset=0
- Add Load Controls
  - LC00: Connect
  - LC01: Unload, LSM Index=1
  - LC02: Unload, LSM Index=2
  - LC03: Unload, LSM Index=3
  - LC04: Load, LSM Index=1
  - LC05: AbsSegment, Access=33h , Address=0116h, LSM Index=1, MemType=3, SegFlags=0, SegType=0, Size=0040h
  - LC06: TaskSegment, Address=0116h, ApplicationNumber=0, ApplicationVersion=0, LSM Index=1, Manufacturer ID=0, PEI Type=0
  - LC07: Load, LSM Index=2
  - LC08: AbsSegment, Access=33h , Address=0156h, LSM Index=2, MemType=3, SegFlags=0, SegType=0, Size=003Eh
  - LC09: TaskSegment, Address=0156h, ApplicationNumber=0, ApplicationVersion=0, LSM Index=2, Manufacturer ID=0, PEI Type=0
  - LC10: Load LSM Index=3
  - LC11: AbsSegment, Access=33h , Address=E000h, LSM Index=3, MemType=3, SegFlags=0, SegType=0, Size=0916h
  - LC12: AbsSegment, Access=33h , Address=0194h, LSM Index=3, MemType=3, SegFlags=0, SegType=0, Size=1E6Ch
  - LC13: TaskSegment, Address=0, ApplicationNumber=1, ApplicationVersion=1, LSM Index=3, Manufacturer ID=1, PEI Type=FFh

- LC14: TaskCtrl2, Address=0194h, LSM Index=3, Seg0=0, Seg1=0
- LC15: LoadCompleted LSM Index=1
- LC16: LoadCompleted LSM Index=2
- LC17: LoadCompleted LSM Index=3
- LC18: Disconnect

- Add Parameter Types
  - Restriction : name = en_No_DPT, range: DPT1 (on/off) = 0, DPT3 (dimming) = 3, No Function = 99
  - Number : name = BYTE, type unsignedInt
- Add Parameters
  - Memory: name = Key0..7, Type = BYTE, CodeSegment = 0194h, Absolute Address = 1C00 to 1C07, Text = Key for object 0..7

    NOTE 1 Create the parameter firstly, then use the properties window for its configuration, this is more convenient than using the 'Add new Memory Parameter' window: it allows to specify the Absolute Address directly, the Offset will be calculated automatically

  - Virtual : name = rocker 1..4, en_No_DPT , default = DPT 1.xxx.
- Add Communication Objects
  - Code segment = 0194h, offset = 0
  - Object #0..3 : Name = object 0..3, Object Size = 1 Bit, Function Text = send
  - Object #4..7 : Name = object 0..3, Object Size = 4 Bit, Function Text = send
  - Object #8 : Name = object 8, Object Size = 3 bytes, Function Text = receive/test
- Add Communication Objects References
- Result



Index	Unique Number	Name	Default Value	Location	Parameter Type	Text	Size in bit	Acc
0	0001h	Key0	00h	[0194h] +1A6Ch	BYTE	Key for object 0	0008h	Rea
1	0002h	Key1	00h	[0194h] +1A6Dh	BYTE	Key for object 1	0008h	Rea
2	0003h	Key2	00h	[0194h] +1A6Eh	BYTE	Key for object 2	0008h	Rea
3	0004h	Key3	00h	[0194h] +1A6Fh	BYTE	Key for object 3	0008h	Rea
4	0005h	Key4	00h	[0194h] +1A70h	BYTE	Key for object 4	0008h	Rea
5	0006h	Key5	00h	[0194h] +1A71h	BYTE	Key for object 5	0008h	Rea
6	0007h	Key6	00h	[0194h] +1A72h	BYTE	Key for object 6	0008h	Rea
7	0008h	Key7	00h	[0194h] +1A73h	BYTE	Key for object 7	0008h	Rea
8	0009h	rocker 1	0000h=DPT1	-	en_DPT_no	rocker 1	0008h	Rea
9	000Ah	rocker 2	0000h=DPT1	-	en_DPT_no	rocker 2	0008h	Rea
10	000Bh	rocker 3	0000h=DPT1	-	en_DPT_no	rocker 3	0008h	Rea
11	000Ch	rocker 4	0000h=DPT1	-	en_DPT_no	rocker 4	0008h	Rea

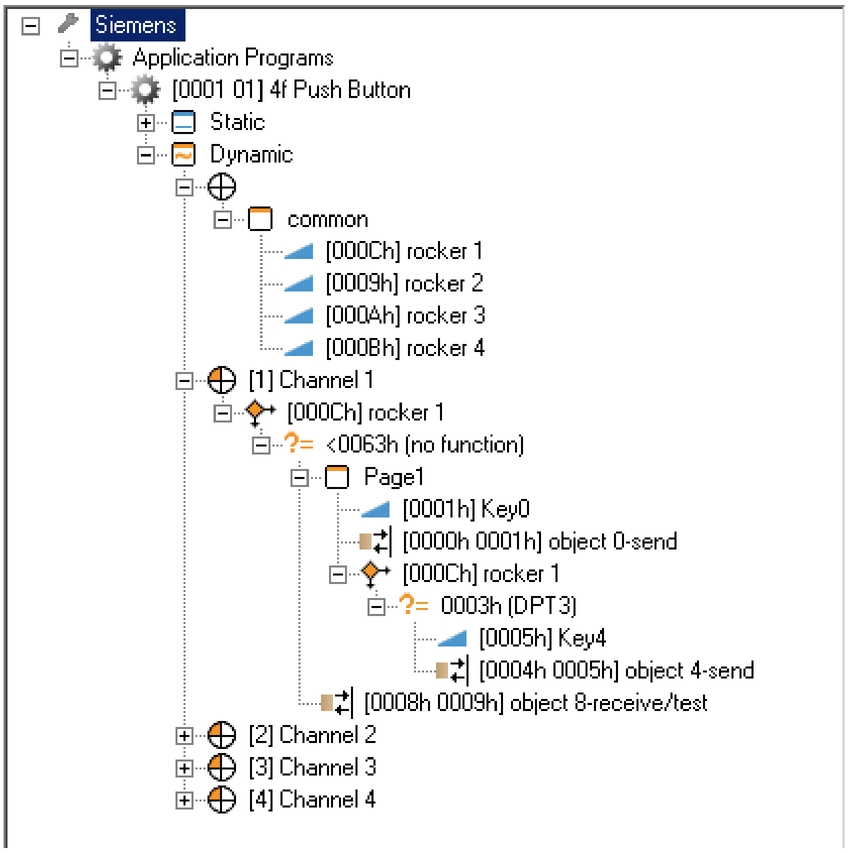
Figure 5 - First steps creating an ETS product entry in the KNX MT

In the dynamic part of the AP:

- Add ChannelIndependentBlock:
  - Add ParameterBlock: Name = common, Text = common page
    - Add ParameterRefRef rocker 1
    - Add ParameterRefRef rocker 2
    - Add ParameterRefRef rocker 3
    - Add ParameterRefRef rocker 4
- Add Channel: Name = Channel 1, Text = rocker 1
  - Add Choose: Parameter = rocker 1
  - Add When: default = False, test: <99

    NOTE 2 ‘99’ is the value for ‘No Function’
- Add ParameterBlock: Name = Page1, Text = parameter page
  - Add ParameterRefRef Key0
  - Add ComObjectRefRef: object 0
  - Add Choose: Parameter = rocker 1
    - Add When: default = False, test: 3
    - Add Parameter Key4
    - Add ComObjectRefRef: object 4
  - Add ComObjectRefRef object 8
- Add Channel: Name = Channel 2, Text = rocker 2
  - Add Choose: Parameter = rocker 2
  - Add When: default = False, test: <99
  - Add ParameterBlock: Name = Page1, Text = parameter page
    - Add ParameterRefRef Key1
    - Add ComObjectRefRef: object 1
    - Add Choose: Parameter = rocker 2
      - Add When: default = False, test: 3
      - Add Parameter Key5
      - Add ComObjectRefRef: object 5
    - Add ComObjectRefRef object 8
- Add Channel: Name = Channel 3, Text = rocker 3
  - Add Choose: Parameter = rocker 1
  - Add When: default = False, test: <99
  - Add ParameterBlock: Name = Page1, Text = parameter page
  - Add ParameterRefRef Key2
    - Add ComObjectRefRef: object 2

    - Add Choose: Parameter = rocker 3
      - Add When: default = False, test: 3
      - Add Parameter Key6
      - Add ComObjectRefRef: object 6
    - Add ComObjectRefRef object 8
- Add Channel: Name = Channel 4, Text = rocker 4
  - Add Choose: Parameter = rocker 4
  - Add When: default = False, test: <99
  - Add ParameterBlock: Name = Page1, Text = parameter page
    - Add ParameterRefRef Key3
    - Add ComObjectRefRef: object 3
    - Add Choose: Parameter = rocker 4
      - Add When: default = False, test: 3
      - Add Parameter Key7
      - Add ComObjectRefRef: object 7
    - Add ComObjectRefRef object 8
- Result:

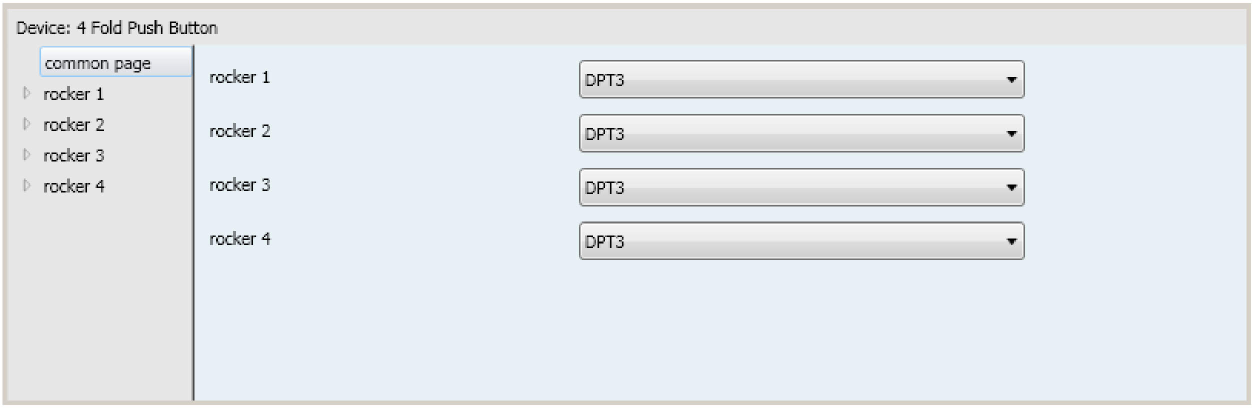


```
Siemens
  Application Programs
    [0001 01] 4f Push Button
      Static
      Dynamic
        
          common
            [000Ch] rocker 1
            [0009h] rocker 2
            [000Ah] rocker 3
            [000Bh] rocker 4
        [1] Channel 1
          [000Ch] rocker 1
            <0063h (no function)
              Page1
                [0001h] Key0
                [0000h 0001h] object 0-send
                [000Ch] rocker 1
                  0003h (DPT3)
                    [0005h] Key4
                    [0004h 0005h] object 4-send
              [0008h 0009h] object 8-receive/test
        [2] Channel 2
        [3] Channel 3
        [4] Channel 4
```

Figure 6 - ETS product entry in the KNX MT after adding the channels

3.5.2 Import into ETS4

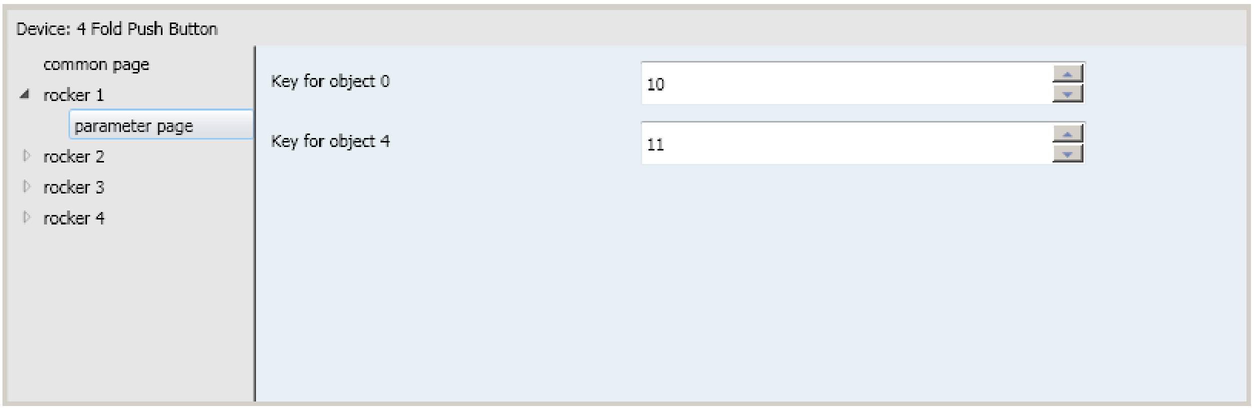
- build the project in MT4
- create a test project via the ‘Edit’ menu in MT4
- import the test project in ETS4
- the device is then visible in the ‘Device’ panel under ’All Devices’
- link its GOs with GAs, in our example:
  - 9/0/1, 9/0/2, 9/0/3, 9/0/4 for object 0..3 (on/off)
  - 9/6/11, 9/6/12, 9/6/13, 9/6/14 for object 4..7 (dimming)
  - 10/1/53 for object 8 (‘simulation’)
- prior to programming with ETS, the evaluation board needs to be put in its factory settings, proceed as follows:
  - start the NEC Tool ID78K0-QB
  - select chip 78F0537_64
  - click File/Download from the menu in order to download BIM_M_130__M_135_Debug_v1_08.hex
  - click Run/Go from the menu in order to activate the device
- program the device with ETS
- ETS4 Screenshots
  - Common parameter page


Device: 4 Fold Push Button

rocker 1 DPT3
rocker 2 DPT3
rocker 3 DPT3
rocker 4 DPT3

Figure7 - Common parameter page

- Parameter for rocker 1

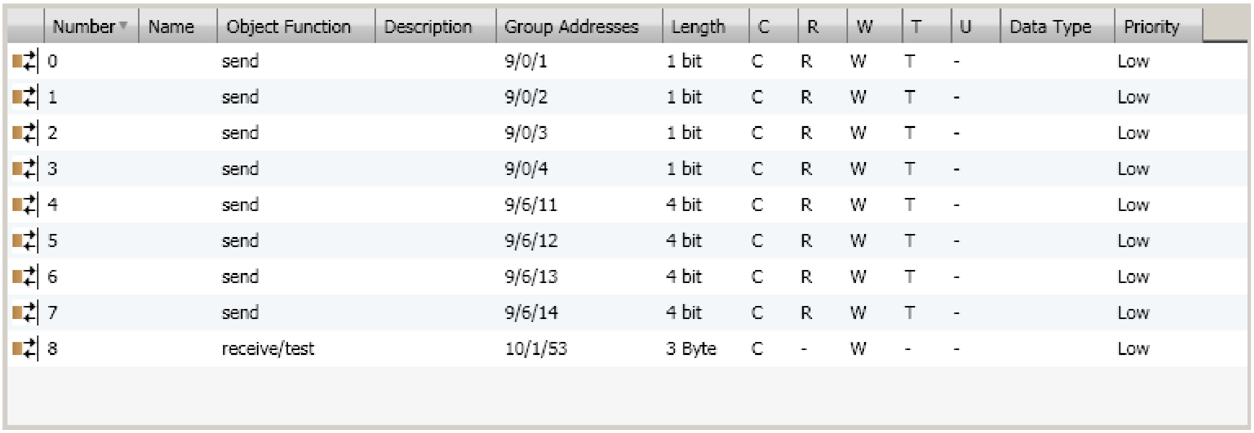


Key for object 0: 10

Key for object 4: 11

Figure 8 - Parameter rocker 1

- Group Objects

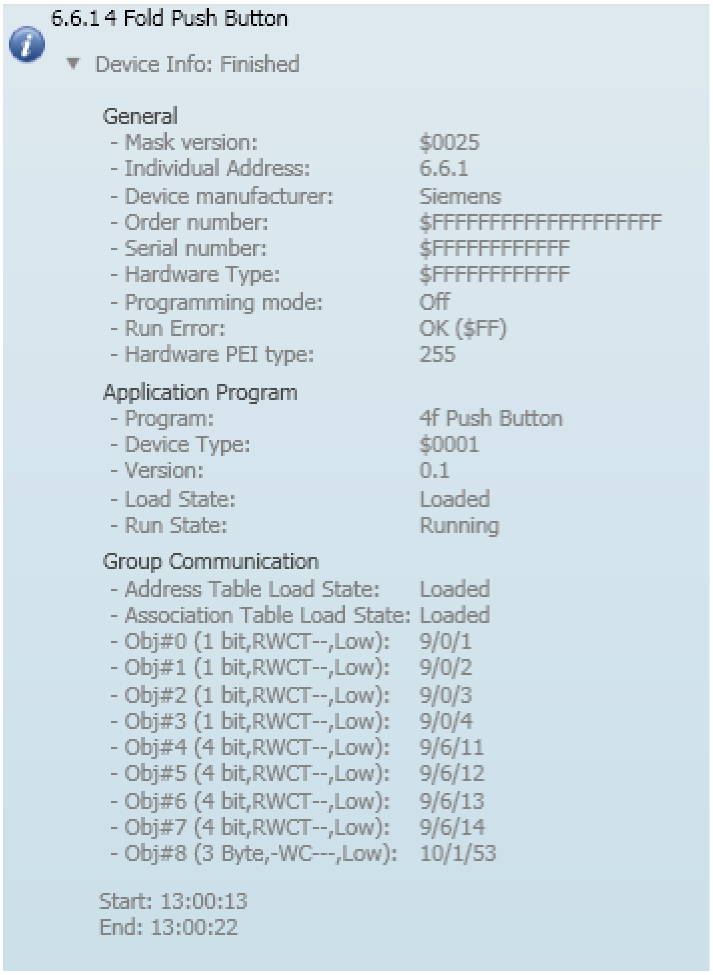


Number	Name	Object Function	Description	Group Addresses	Length	C	R	W	T	U	Data Type	Priority
0		send		9/0/1	1 bit	C	R	W	T	-		Low
1		send		9/0/2	1 bit	C	R	W	T	-		Low
2		send		9/0/3	1 bit	C	R	W	T	-		Low
3		send		9/0/4	1 bit	C	R	W	T	-		Low
4		send		9/6/11	4 bit	C	R	W	T	-		Low
5		send		9/6/12	4 bit	C	R	W	T	-		Low
6		send		9/6/13	4 bit	C	R	W	T	-		Low
7		send		9/6/14	4 bit	C	R	W	T	-		Low
8		receive/test		10/1/53	3 Byte	C	-	W	-	-		Low

Figure 9 - Group Objects

3.5.3 Checking whether ETS4 did configure the device correctly

This can e.g. be done with the device info function of ETS, preferably with ‘group communication’ info.

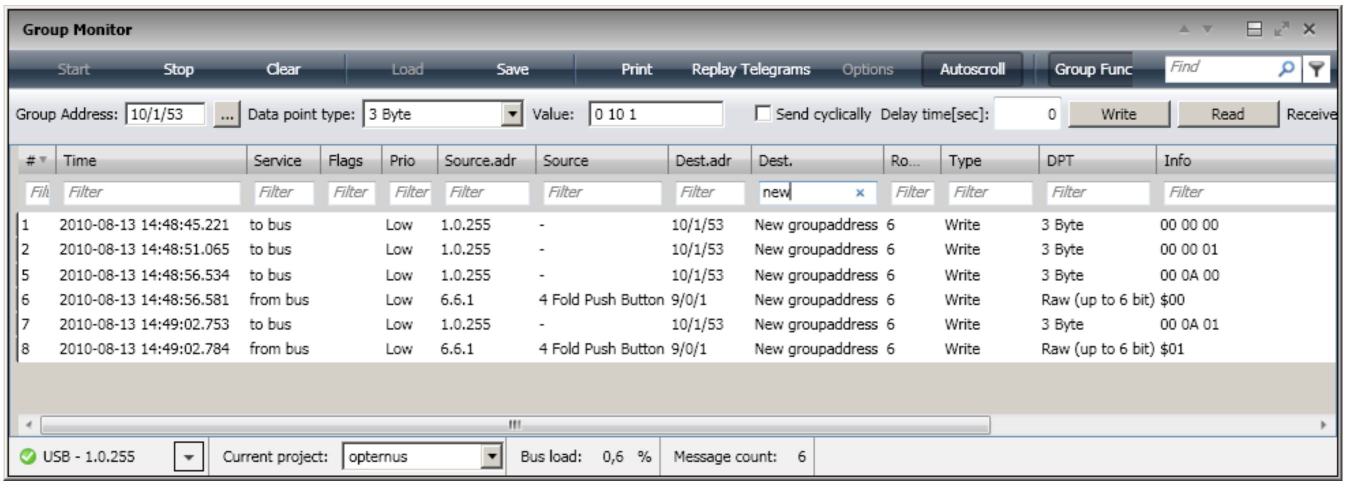


```
6.6.1 4 Fold Push Button
▼ Device Info: Finished

General
 - Mask version:                 $0025
 - Individual Address:           6.6.1
 - Device manufacturer:          Siemens
 - Order number:                 $FFFFFFFFFFFFFFFFFFFF
 - Serial number:                $FFFFFFFFFFFF
 - Hardware Type:                $FFFFFFFFFFFF
 - Programming mode:             Off
 - Run Error:                    OK ($FF)
 - Hardware PEI type:            255
Application Program
 - Program:                      4f Push Button
 - Device Type:                  $0001
 - Version:                      0.1
 - Load State:                   Loaded
 - Run State:                    Running
Group Communication
 - Address Table Load State:     Loaded
 - Association Table Load State: Loaded
 - Obj#0 (1 bit,RWCT--,Low):     9/0/1
 - Obj#1 (1 bit,RWCT--,Low):     9/0/2
 - Obj#2 (1 bit,RWCT--,Low):     9/0/3
 - Obj#3 (1 bit,RWCT--,Low):     9/0/4
 - Obj#4 (4 bit,RWCT--,Low):     9/6/11
 - Obj#5 (4 bit,RWCT--,Low):     9/6/12
 - Obj#6 (4 bit,RWCT--,Low):     9/6/13
 - Obj#7 (4 bit,RWCT--,Low):     9/6/14
 - Obj#8 (3 Byte,-WC---,Low):    10/1/53

Start: 13:00:13
End: 13:00:22
```

Figure 10 - Checking correct configuration with ETS

Use ETS4 Group Monitor to simulate the AM

- This is, send A_GroupValue_Write Telegrams for GA = 10/1/53 (object 8).
- The results can be checked in two ways:
  - with an actual actuator: then the lamp switches on, or
  - within the same the Group Monitor session.
    - Send the following data for GA = 10/1/53:
      - 0 0 0
      - 0 0 1
      - 0 10 0
      - 0 10 1
    - the results can be seen in this screenshot:



Group Address: 10/1/53 Data point type: 3 Byte Value: 0 10 1 Send cyclically Delay time[sec]: 0

#	Time	Service	Flags	Prio	Source.adr	Source	Dest.adr	Dest.	Ro...	Type	DPT	Info
1	2010-08-13 14:48:45.221	to bus		Low	1.0.255	-	10/1/53	New groupaddress	6	Write	3 Byte	00 00 00
2	2010-08-13 14:48:51.065	to bus		Low	1.0.255	-	10/1/53	New groupaddress	6	Write	3 Byte	00 00 01
5	2010-08-13 14:48:56.534	to bus		Low	1.0.255	-	10/1/53	New groupaddress	6	Write	3 Byte	00 0A 00
6	2010-08-13 14:48:56.581	from bus		Low	6.6.1	4 Fold Push Button	9/0/1	New groupaddress	6	Write	Raw (up to 6 bit)	$00
7	2010-08-13 14:49:02.753	to bus		Low	1.0.255	-	10/1/53	New groupaddress	6	Write	3 Byte	00 0A 01
8	2010-08-13 14:49:02.784	from bus		Low	6.6.1	4 Fold Push Button	9/0/1	New groupaddress	6	Write	Raw (up to 6 bit)	$01

USB - 1.0.255 Current project: opternus Bus load: 0,6 % Message count: 6

Figure 11 - Stimulating AP with ETS Group Monitor

- 0 0 0: no reaction from the device because Key0 = 10 (0Ah)
- 0 0 1: idem
- 0 10 0: a A_GroupValue_Write Telegram with value = 0 for GA = 9/0/1 was sent on the bus
- 0 10 1: a A_GroupValue_Write Telegram with value = 1 for GA = 9/0/1 was sent on the bus
- repeat for all other GOs