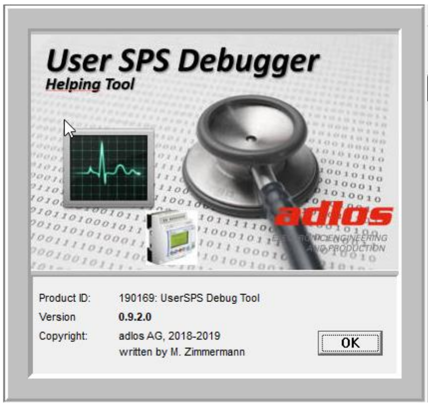
AN100639-000: Customized KannMOTION Coding & Debug guidance

Introduction

This document shall enable Users to work with KannMOTION drives Generation-2. This Generation allows to customize drive function by USERS ANSI-C code.
KannMOTION is based on ISO/IEC 9899:1999 standard, this standard is commonly referred to as C99.

KannMOTION -adlos customizing approach, allows to implement own functionallity in a very code und runtime efficient way. During developing of customized firmware it might be very helpful to have a kind of Debugger tool.
This document shows you a way, how you can make your data visible.



Needs Comwatch 190077 Toolset >= V2.1.1.1 !

How it works

USER static variables can be accessed over defined communication port. To get advantage of that, we need to know where our variables are mapped in and what is the size and interpretation of it. Comwatch and KannMotion Manager uses for that a dedicated XML-File defining the access to 'online' data. Every device, firmware has its own XML interpretation file. For debug purpose of aur customized software we will show, how we can use the same methodology to get information about our variables during system operation.

Figure 1: Online XML Section of Firmware 190167

```xml
<!-- Online Diagnose Daten [011] -->
<OnlineData IddNr="011">
  <DataTable CMD="R1#0D" Intervall="5" Info="MainApp Data" UserLevel="0x008F">
    <DataRec name="enAppState" type="ENU" factor="1" unit="E" digits="0" list="0:WaitSupply;1:Init_Enc;2:Init_Drv;15:Test;16:Run;21:RUN_SPS;32:Homin
    <DataRec name="ErrorBits" type="BF_8" factor="1" unit="E" digits="0" list="0:DrvOverCurr;1:DrvOvTemp;2:UinToLO;3:Timeout;4:StpDrvInit;5:EncoderI
    <DataRec name="ErrorBits" type="BF_8" factor="1" unit="E" digits="0" list="0:Stall;1:bSPS;2:NC;3:NC;4:NC;5:NC;6:NC;7:NC;" KMMTag="Error"/>
    <DataRec name="Temperature" type="UI_8" factor="1" formula="(x -50)" unit="°C" digits="0" KMMTag="Temperature"/>
    <DataRec name="TargetPos [um]/[0.1°]" type="SI_32" factor="1" unit="um/01" digits="0" />
    <DataRec name="ActualPos [um]/[0.1°]" type="SI_32" factor="1" unit="um/01" digits="0" KMMTag="Position"/>
    <DataRec name="CheckSum" type="UI_8" factor="1" unit="E" digits="0" show="false" UserLevel="0x0080" />
  </DataTable>
```

There is one RAM structure defined in User.c which is static mapped and so the address range is known in User-side and also in base application. This is a must to have access to defined RAM during operation od the KANNmotion drive.

Figure 2: Static Var structure in L2_APPC_SPS_User.c

```
1 L2_APPC_SPS_User.h   2 L2_APPC_SPS_User.c
64   // ---- Globale Variabeln loesen -------
65   LOCATEVARSPS tAPPCSPS stAppCSPS;
```

This RAM section holds predefined variables, to enable data exchange between Main-Application and User-Application.
As next step, we wil have a short view on certain typedef to get the overview of our possibilities.
In *Figure 3* we will find the overall structure having attribute 'packed' what means, that compiler has no permission to do padding. This is marked with 1) in *Figure 3*.
For debugging now it is important to have a deeper view into Marking 2) and 3) of *Figure 3*.
2) union means, that the memory size reserved is given by the 'biggest' following member. A **union** is a special data type available in C that allows to store different data types in the same memory location.

You can define a union with many members, but only one member can contain a value at any given time, respctively they share the same memory location.

Figure 3: Static Var typedef in L2_APPC_SPS_User.h

C:\SW_Works_Git\190167_K17eVAT_USER\Src\L2_APPC_SPS_User.h - SciTE

File Edit Search View Tools Options Language Buffers Help

```
tDepK17EPI;

//! SPS-vordefinierte USER-Variabeln  (1)
typedef struct DO_ATTR_PACK DO_ALLIGN4
{
  unDIN_Infot     u8_Din;                 //!< R: <EVA> Abbild der digitalen Eingänge
  unDIN_Infot     u8_DinChange;           //!< R: <EVA> Änderungen seit dem letzten Mal
  unDOUTS12t      u8_Dout;                //!< R: <EVA> Abbild der digitalen Ausgänge
  tenAPPSTATES    enAppState;             //!< R: <EVA> aktueller Zustand APP
  SI_32           i32_ActualPos_um_01deg;//!< R: <EVA> aktuelle Position
  tERRBITS        u16_Errorbits;          //!< R: <EVA> Abbbild Fehlerbits
  tenMOTSTATES    enDrvState;             //!< R: <EVA> aktueller Zustand DRV
  SI_8            i8_VARX;                //!< R: 1x allgemeine 8-Bit VAR [8-Bit]
  UI_16           u16_Timer5ms[4];        //!< R: 4x 5ms Timer [16-Bit]
  UI_8            u8_StepChain[4];        //!< R: 4x Schrittketten- Vars [8-Bit]
  SI_32           i32_PosVAR_um_01deg[3];//!< RW: 3x zusätzliche Psoitionsmerker ..zB um Endanschläge zu merken
  union  (2)                              //       5x 4Bytes = 20 Byte, Flex User Section, Overlapped !
  {
    tMyData       myData;  (3)                  //!< R: own defined Data Type
    tDepK17A      K17a;                         //!< R: <EVA> vom 17a
    tDepK17C      K17c;                         //!< R: <EVA> vom 17c
    tDepK17E      K17e;                         //!< R: <EVA> vom 17e
    tDepK17F      K17f;                         //!< R: <EVA> vom 17f
    tDepK17G      K17g;                         //!< R: <EVA> vom 17g
    tDepK11A      K11a;                         //!< R: <EVA> vom 11a
    tDepK11B      K11b;                         //!< R: <EVA> vom 11b
    tDepD17A      D17a;                         //!< R: <EVA> vom D17a
    tDepK17EPI    K17ePi;                       //!< R: <EVA> vom 17e Pirani
    UI_32         u32_UserVar[xFLEXSIZE32];     //!< R: je nach Steuerung freies USER Feld 32-Bit UI Zugriff
    SI_32         i32_UserVar[xFLEXSIZE32];     //!< R: je nach Steuerung freies USER Feld 32-Bit SI Zugriff
    UI_16         u16_UserVar[xFLEXSIZE32*2];   //!< R: je nach Steuerung freies USER Feld 16-Bit UI Zugriff
    SI_16         i16_UserVar[xFLEXSIZE32*2];   //!< R: je nach Steuerung freies USER Feld 16-Bit SI Zugriff
    UI_8          u08_UserVar[xFLEXSIZE32*4];   //!< R: je nach Steuerung freies USER Feld 8-Bit UI Zugriff
    SI_8          i08_UserVar[xFLEXSIZE32*4];   //!< R: je nach Steuerung freies USER Feld 8-Bit SI Zugriff
  }
  KxxFlexUser;
  #ifdef fFeCOMCMD
  tCSPSCOMRXDATA st32_ComData;                  //!< RW: from Serial Port received data Command <Df>, shall be cleared by USER !
  #endif
  #ifdef fFeNVVar
  union  (4)                              //       5x 4Bytes = 20 Byte, Flex User Section, Overlapped !
  {
    tMyNVData     myData;  (5)                  //!< eigene Struktur
    UI_32         u32_UserVar[xFLEXSIZE32NV];     //!< R: USER Feld 32-Bit UI Zugriff, nonVolatile
    SI_32         i32_UserVar[xFLEXSIZE32NV];     //!< R: USER Feld 32-Bit SI Zugriff, nonVolatile
    UI_16         u16_UserVar[xFLEXSIZE32NV*2];   //!< R: USER Feld 16-Bit UI Zugriff, nonVolatile
    SI_16         i16_UserVar[xFLEXSIZE32NV*2];   //!< R: USER Feld 16-Bit SI Zugriff, nonVolatile
    UI_8          u08_UserVar[xFLEXSIZE32NV*4];   //!< R: USER Feld 8-Bit UI Zugriff, nonVolatile
    SI_8          i08_UserVar[xFLEXSIZE32NV*4];   //!< R: USER Feld 8-Bit SI Zugriff, nonVolatile
  }
  UserNonVolatile;
  #endif
}
tAPPCSPS_USERVAR;
```

For coding and Debugging we will use for our purpose the member myData 3). This member is by default 'empty' defined, so ist is our freedom to define it as we need.
For that we will edit *L2_APPC_SPS_myUserTypes.h* and fill into the structure tMyData what we need.

Figure 4: MyUserTypes.h Example 1

1 L2_APPC_SPS_myUserTypes.h | 2 Untitled

```


//! eigene Datenstruktur, statisch, MAX Grösse (xFLEXSIZE32 * 4), attribute nicht ändern !
typedef struct DO_PACK_ALLIGN4
{
  UI_32 u32_PressureDif;  //!< Druckdifferenz
  UI_16 u16_SpeedReg;     //!< Geschwindigkeitsregelwert
}
tMyData;

//! eigene NV Datenstruktur, statisch MAX Grösse (xFLEXBLOCKCNTNV * 4), attribute nicht ändern !
typedef struct DO_PACK_ALLIGN4
{
  UI_16 u16_RegKI;        //!< Regler-Konstante
}
tMyNVData;

```

For Debugger's Wizzard it's important that you use known type specifiers like (UI_8,SI_8;UI_16…) or inside the same h-file self defined structures, enums or unions.

known Type qualifiers

Unsigned Integer Types	
UI_8	8 Bit -> [0..255]
UI_16	16 Bit -> [0..65535]
UI_32	32 Bit -> [0..4'294'967'295]
UI_64	64 Bit
Signed Integer Types	
SI_8	8 Bit -> [-128 .. +127]
SI_16	16 Bit -> [-32768 .. +32767]
SI_32	32 Bit -> [-2147483648 .. +2147483647]
SI_64	64 Bit
floating point types	
F_32	32 Bit floating point number
F_64	64 Bit floating point number
special	
BIT(var)	1 Bit where var ist he Bit-Name

tMyData and tMyNVData

you might fill in both structures with variables you will use. Both structures are held while running inside RAM. Difference between tMyData and tMyNVData is, that tMyNVData initail value is loaded at PowerUp from device EEPROM, so means you might use it as Non volatile parameter if you want.

Figure 5:Access inside your code

```
stAppCSPS.SPSUserVar.KxxFlexUser.myData.u16_SpeedReg += 5;
stAppCSPS.SPSUserVar.KxxFlexUser.myData.u32_PressureDif *= stAppCSPS.SPSUserVar.UserNonVolatile.myData.u16_RegKI;
```

There might be a way to shorten your writing inside code by setting 2 definitions inside MyUserTypes.h.

```
// following definitions will shorten my Access inside code
#define myUserData     stAppCSPS.SPSUserVar.KxxFlexUser.myData
#define myUserNVData   stAppCSPS.SPSUserVar.UserNonVolatile.myData
```

So as a consequence of above definitions we can write the same 2 lines much shorter:

Figure 6: Shorten Access

```
myUserData.u16_SpeedReg += 5;
myUserData.u32_PressureDif *= myUserNVData.u16_RegKI;
```

Take care, that your defined structures do not need more RAM than the Flex-Field is offering !
In most KANNmotion firmware the ***xFLEXSIZE32***=32=***128 Byte*** and ***xFLEXSIZE32NV***=4=***16 Byte***
Do not remove DO_ATTR_PACK at ist definition.

Using StepChain- Variables as State identifier

You might use a StepChain Variable as your State-Machine identifier. To have it better readable we will set first a Substitute by a #define.

1 L2_APPC_SPS_myUserTypes.h | 2 Untitled | 3 L2_APPC_SPS_myUserTypes.h

```
   #include "L0_KannCSPS_exTypes.h"
#endif

// ****** Zuweisungen für kürzere Schreibweise der Vorbelegten Werte aus dem SPS-User Block ***
#define eMyState            stAppCSPS.SPSUserVar.u8_StepChain[0]          // [type=tenMyState]
#define My5msDelayTimer     stAppCSPS.SPSUserVar.u16_Timer5ms[0]          // Hilfs-DelayTimer
```

After that we define an enumeration, containing our states:

```
//! My State Machine State Enumeration
typedef enum
{
    MyState_INIT_DRV            = 0,
    MyState_INIT_PIRANI         = 1,
    MyState_INIT_PIRANI_WAIT    = 2,
    MyState_RUN                 = 0x10,
}
tenMyState;
```

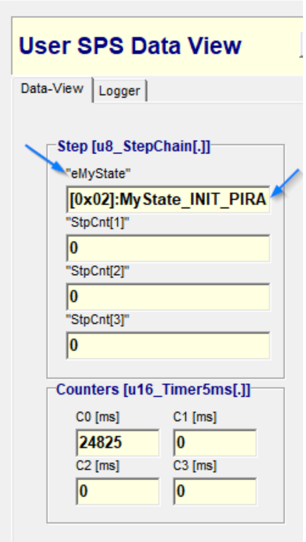
And as a consequence set this type as comment to your substituiton Line!

1 L2_APPC_SPS_myUserTypes.h | 2 Untitled | 3 L2_APPC_SPS_myUserTypes.h

```
    #include "L0_KannCSPS_exTypes.h"
  #endif

  // ****** Zuweisungen für kürzere Schreibweise der Vorbelegten Werte aus dem SPS-User-Block ***
  #define eMyState           stAppCSPS.SPSUserVar.u8_StepChain[0]        // [type=tenMyState]
  #define My5msDelayTimer    stAppCSPS.SPSUserVar.u16_Timer5ms[0]        // Hilfs-DelayTimer
```

If defined like this, Debugger Tool will Rename StepChain by your substitution an will also set your enumeration names into result field.

Figure 7: StepChain as State



In your code, it might look like this:

```
switch(eMyState)
{
   case MyState_INIT_DRV:
   {
      // Motorparameter Konfiguration
      stAppCSPS.SPSCallFunctions.DrvChgSetting(12, 1000);   // Min. Speed     [0.1rpm]
      stAppCSPS.SPSCallFunctions.DrvChgSetting(13, 6000);   // Max. Speed
      stAppCSPS.SPSCallFunctions.DrvChgSetting(14, 0);      // Torque-Hold [0.5.%]
      stAppCSPS.SPSCallFunctions.DrvChgSetting(15, 200);    // Torque-ACC
      stAppCSPS.SPSCallFunctions.DrvChgSetting(16, 100);    // Torque-RUN
      stAppCSPS.SPSCallFunctions.DrvChgSetting(17, 200);    // Torque-DEC
      stAppCSPS.SPSCallFunctions.DrvChgSetting(18, 20);     // Acceleration   [0.5%]
      stAppCSPS.SPSCallFunctions.DrvChgSetting(19, 20);     // Deceleration
      eMyState = MyState_INIT_PIRANI;
      break;
   }
   case MyState_INIT_PIRANI:
   {
```

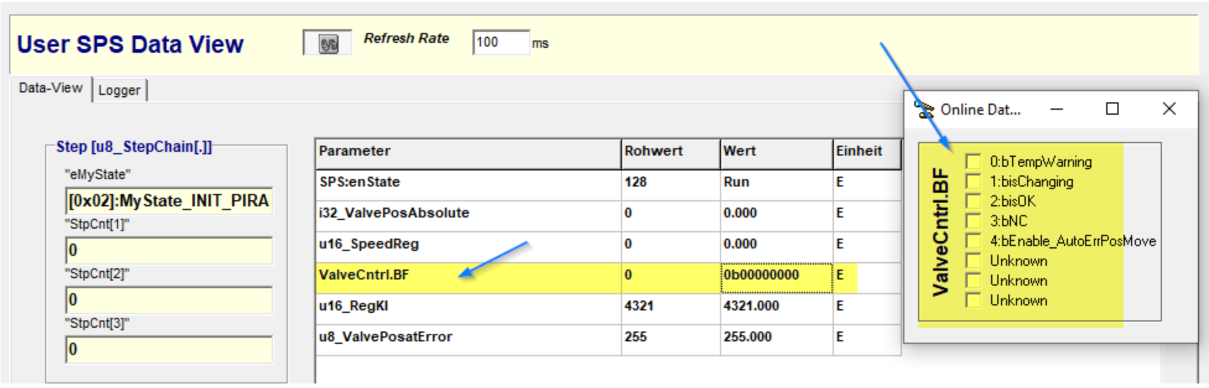
Using Bit-Fields to get it 'readable'

```
//! My VALVE Info & CNTRL-Bit-Struktur
typedef union DO_ATTR_PACK
{
  UI_8 ucAllBits;                                     //!< R:   alle Control& Info Bits
  struct DO_ATTR_PACK
  {
      BIT(bTempWarning);                              //!< Temperature Warning ist Set
      BIT(bisChanging);                               //!< VAlve is actually Change its Position
      BIT(bisOK);                                     //!< Valve is ok, on Position, No Error
      BIT(bNC);
      BIT(bEnable_AutoErrPosMove);                    //!< when set, the Predefined Error-Position is activated
  }b;
}
tVALVE_INFOCNTRL;

//! eigene Datenstruktur, statisch, MAX Grösse (xFLEXSIZE32 * 4), attribute nicht ändern !
typedef struct DO_PACK_ALLIGN4
{
  SI_32 i32_ValvePosAbsolute;     //!< Absolute Position in [0.1°]
  UI_16 u16_SpeedReg;             //!< Geschwindigkeitsregelwert
  tVALVE_INFOCNTRL ValveCntrl;    //!< Gauge Info and Cntrl Bit-Structure
}
tMyData;
```

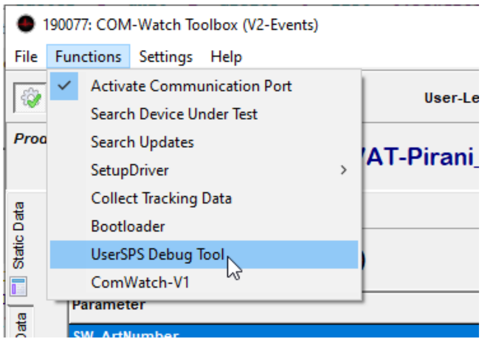
Inside code it might like this

```
        case MyState_RUN:
        {
            // Drive Valve to Safe Position if Enabled
            if ((myUserData.ValveCntrl.b.bEnable_AutoErrPosMove) && (stAppCSPS.SPSUserVar.u16_Errorbits.b.bEncOverTemp))
            {
                SI_32 i32_ValvePosAbsolute = myUserNVData.u8_ValvePosatError * 300;
                UFu_GotoFuncP(i32_ValvePosAbsolute,eGOTO_um_01deg);
                eMyState++;
            }

            // Manipulate Info Bit
            if (stAppCSPS.SPSUserVar.enDrvState &  (eMS_ROTATE | eMS_GOTOPOS))
            {
               myUserData.ValveCntrl.b.bisChanging = 1;
            }
            else myUserData.ValveCntrl.b.bisChanging = 0;
```

Debugger Tools, shows it afterward like this

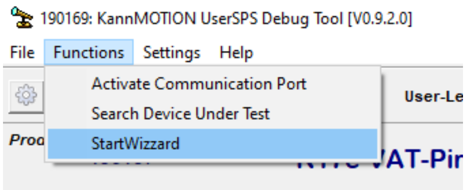


How to Use Debug Tool

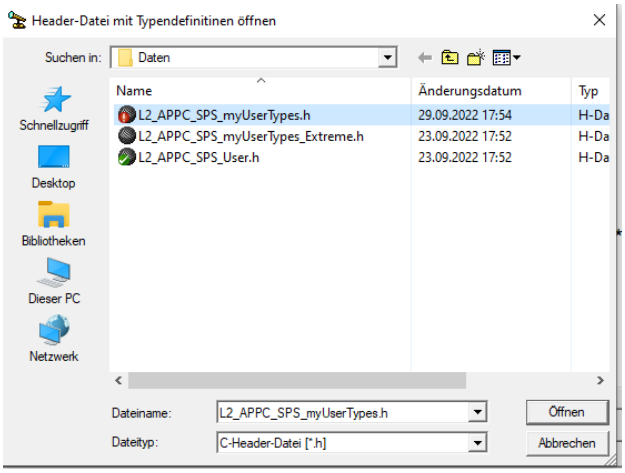


Starting Debug Tool out of COM-Watch

Menu-> Functions->UserSPS Debug Tool

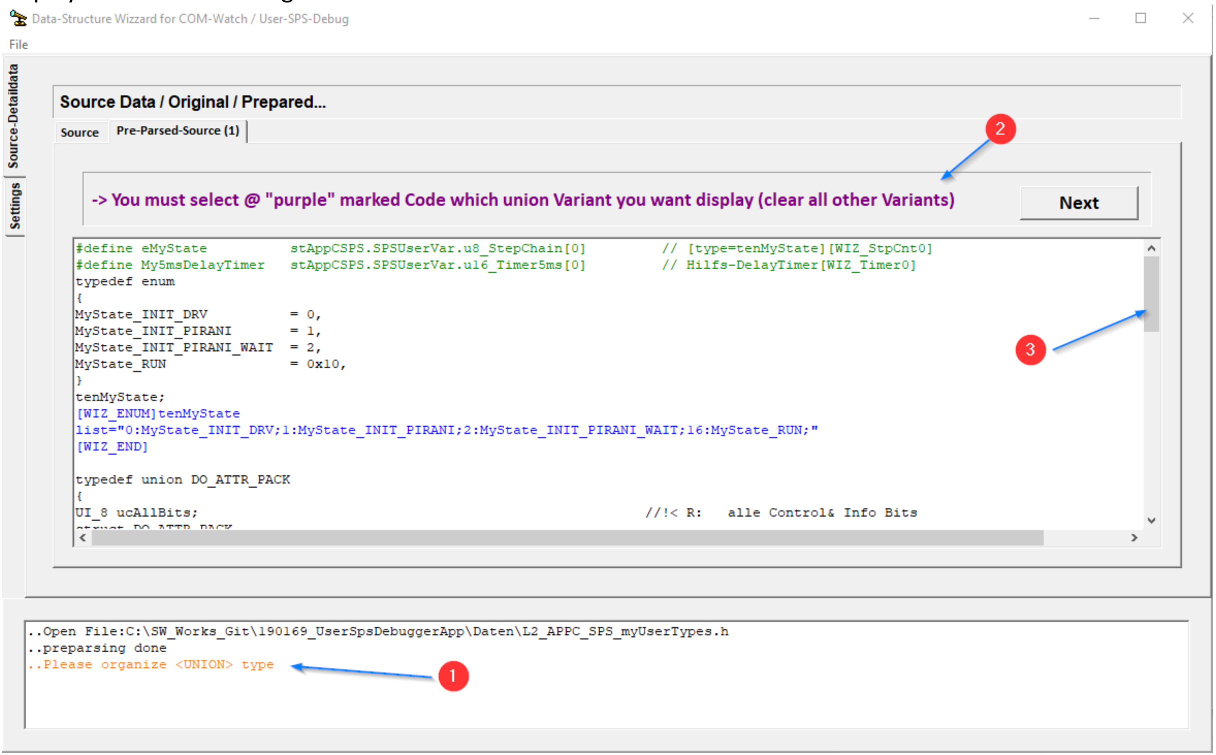


.. as next Step inside Debug-Tool you have to start the XML-Wizzard to actively create an Decoding XML.
Menu->Functions->StartWizzard



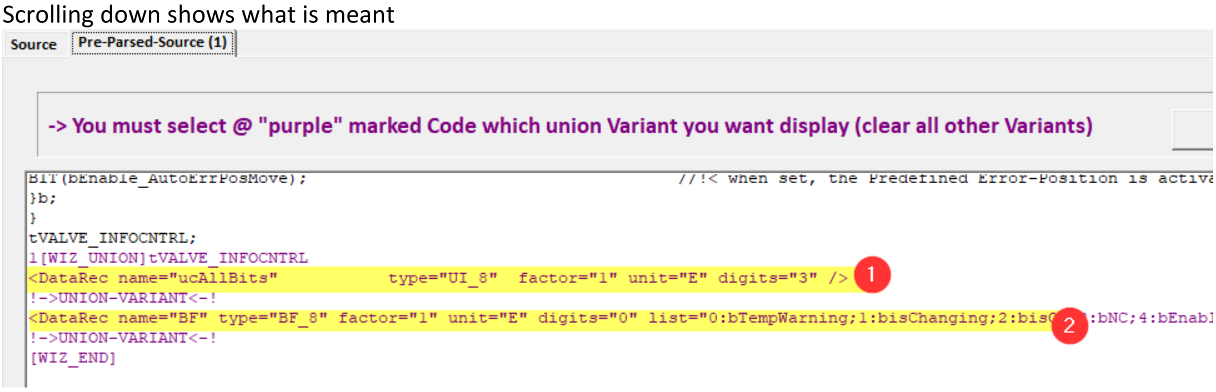
Then choose your h-File, where your type definitions are placed. (tMyData / tMyNVData) shall be present.

If there are some unions implemented, the Wizzard needs your Help to decide which variant you want to be displayed inside the Debug-Tool Gui.



1) Warning in orange, that there was a relevant union found
2) Note that code is marked in purple where you shall do something
3) Scrollbar to scroll througt the pre-parsed code

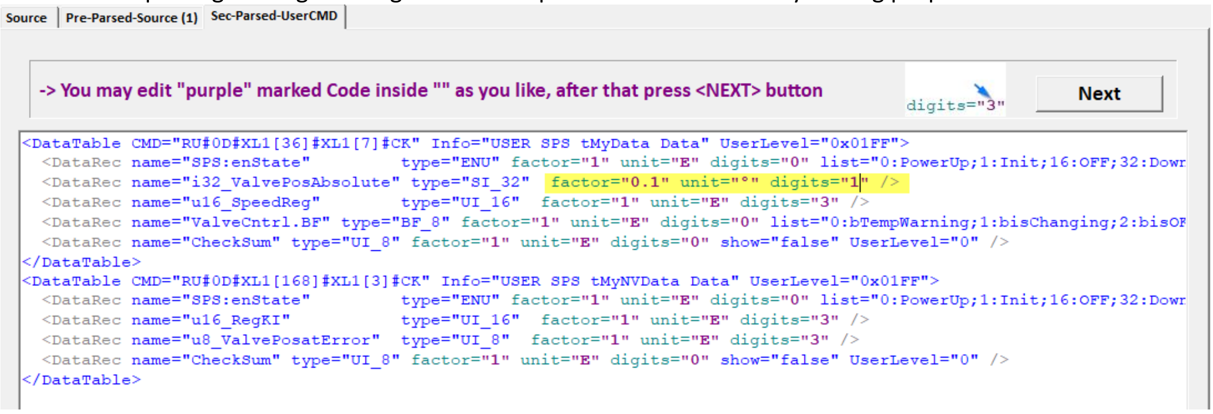
Scrolling down shows what is meant



1) Variant, Debug Tool will show it as UI_8 Value without decoding
2) Variant Bit Field, Debug Tool will show it as 8-Bit data and as Bit-Field

So, we want to see Bit-Field representation, so we delete the other interpretation variant, after doing that, it looks like that:

After second parsing we might change some interpretation of variables by editing purple marked fields.



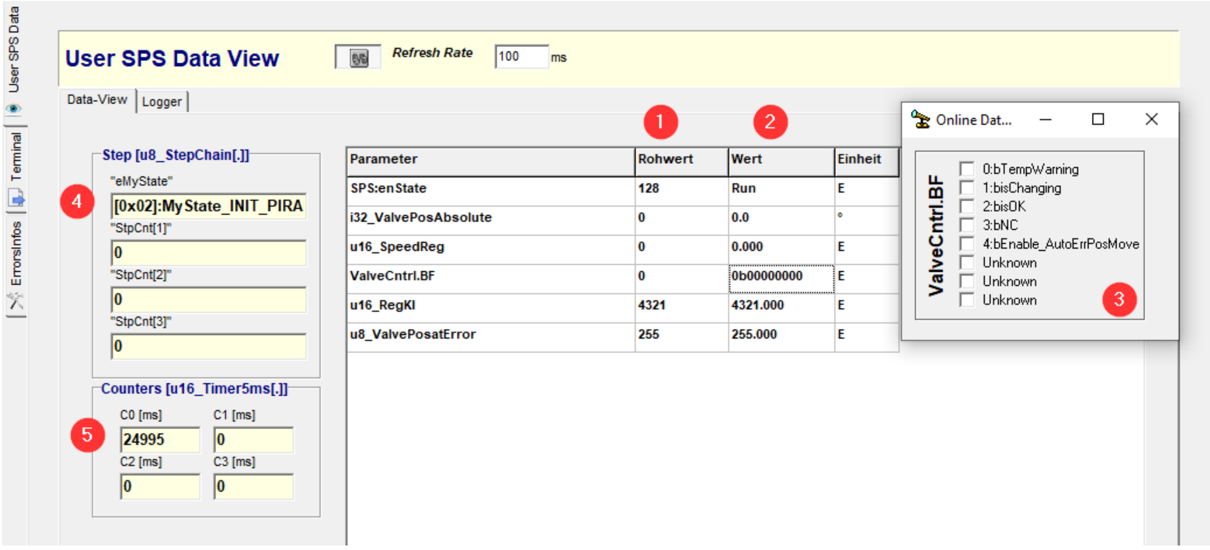
So we cann change factor, what means we can calculate an value… means transmitted RAW data will be multiplied by this factor, Tool will then show RAW-data and calculated value.
Same for Digits, Tool is calculation in Floating Point so you may show digits after comma so you write here an integer number of digits after comma you want to visualize.

Press NEXT…

Final-Output XML will be shown…

Press Close AND Start..

If wizard was successful and our drive is connected we will have a running connection and we can see our variables. Here our example:



1) RAW data column
2) 'calculated' data column
3) Bit-Field detail form
4) Step-chain view, here interpreted as eMyState
5) Timers values

Tools

Adlos Win32-APPs

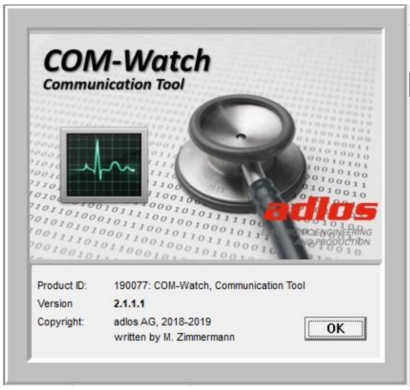
adlos offers for it's customers some Helping and Design-In Tools.

KannMotionManager Tool (190081), manage your drives

KannMOTION Manage is the general tool for our GEN2 drives. This tool comes with an integrated C-coder and a visual drag and drop User interface for customizing your drive.

https://kannmotion.adlos.com/download/kannmotionmanager/application/SetupKannMOTIONManager.zip

ComWatch Communication Tool (190077), for Life values



ComWatch is a helping tool for engineers and technicians to explore device specific parametes, read out tracking data and settings and doing firmware updates.
The software is as it is, and in principle for free for adlos customers, the software is not made for a broad range of standard users, it’s made in principle for technical engineers which are used in working w. windows based software and have some understanding of technical things.

https://kannmotion.adlos.com/download/comwatchtool/ComWatchSetup.zip

Contact information

Adlos AG
Föhrenweg 14
FL-9496 Balzers

Thomas Vogt
Thomas.Vogt@adlos.com
Tel: +423 263 63 63

Countries: CH, A, LI, SK, IT
www.adlos.com

KOCO MOTION GmbH
Niedereschacher Straße 54
D-78083 Dauchingen

Olaf Kämmerling
O.Kaemmerling@kocomotion.de
Tel: +49 7720/995858-0

Countries: DE, BE, NL, LU
www.kocomotion.de