can\_cfg\_tx-message-definitions.h

AlarmWarningsBattery (359h)

Add:

/\*\* CAN message properties for BMS state message. Required properties are:

 \* - Message ID

 \* - Identifier type (standard or extended)

 \* - Message period and phase in ms

 \* - Endianness of message data @{\*/

#define ALARMS\_WARNINGS\_BATTERY\_ID         (0x359u)

#define ALARMS\_WARNINGS\_BATTERY\_ID\_TYPE    (CAN\_STANDARD\_IDENTIFIER\_11\_BIT)

#define ALARMS\_WARNINGS\_BATTERY\_PERIOD\_ms  (100u)

#define ALARMS\_WARNINGS\_BATTERY\_PHASE\_ms   (0u)

#define ALARMS\_WARNINGS\_BATTERY\_ENDIANNESS (CAN\_BIG\_ENDIAN)

/\*\*@}\*/

#define ALARMS\_WARNINGS\_BATTERY\_MESSAGE                                                \

    {                                                                          \

        .id         = ALARMS\_WARNINGS\_BATTERY\_ID,                                      \

        .idType     = ALARMS\_WARNINGS\_BATTERY\_ID\_TYPE,                                 \

        .dlc        = CAN\_DEFAULT\_DLC,                                         \

        .endianness = ALARMS\_WARNINGS\_BATTERY\_ENDIANNESS,                              \

    },                                                                         \

    {                                                                          \

        .period = ALARMS\_WARNINGS\_BATTERY\_PERIOD\_ms, .phase = ALARMS\_WARNINGS\_BATTERY\_PHASE\_ms \

    }

BatteryLimits (351h)

Add:

/\*\* CAN message properties for BMS limit values. Required properties are:

 \* - Message ID

 \* - Identifier type (standard or extended)

 \* - Message period and phase in ms

 \* - Endianness of message data @{\*/

#define BATTERY\_LIMITS\_ID         (0x351u)

#define BATTERY\_LIMITS\_ID\_TYPE    (CAN\_STANDARD\_IDENTIFIER\_11\_BIT)

#define BATTERY\_LIMITS\_PERIOD\_ms  (100u)

#define BATTERY\_LIMITS\_PHASE\_ms   (30u)

#define BATTERY\_LIMITS\_ENDIANNESS (CAN\_BIG\_ENDIAN)

/\*\*@}\*/

#define BATTERY\_LIMITS\_MESSAGE                                                   \

    {                                                                                \

        .id         = BATTERY\_LIMITS\_ID,                                         \

        .idType     = BATTERY\_LIMITS\_ID\_TYPE,                                    \

        .dlc        = CAN\_DEFAULT\_DLC,                                               \

        .endianness = BATTERY\_LIMITS\_ENDIANNESS,                                 \

    },                                                                               \

    {                                                                                \

        .period = BATTERY\_LIMITS\_PERIOD\_ms, .phase = BATTERY\_LIMITS\_PHASE\_ms \

    }

BatteryMeasurements (356h)

Add:

/\*\* CAN message properties for pack values. Required properties are:

 \* - Message ID

 \* - Identifier type (standard or extended)

 \* - Message period and phase in ms

 \* - Endianness of message data @{\*/

#define BATTERY\_MEASUREMENTS\_ID         (0x356u)

#define BATTERY\_MEASUREMENTS\_ID\_TYPE    (CAN\_STANDARD\_IDENTIFIER\_11\_BIT)

#define BATTERY\_MEASUREMENTS\_PERIOD\_ms  (100u)

#define BATTERY\_MEASUREMENTS\_PHASE\_ms   (60u)

#define BATTERY\_MEASUREMENTS\_ENDIANNESS (CAN\_BIG\_ENDIAN)

/\*\*@}\*/

#define BATTERY\_MEASUREMENTS\_MESSAGE                                                  \

    {                                                                              \

        .id         = BATTERY\_MEASUREMENTS\_ID,                                        \

        .idType     = BATTERY\_MEASUREMENTS\_ID\_TYPE,                                   \

        .dlc        = CAN\_DEFAULT\_DLC,                                             \

        .endianness = BATTERY\_MEASUREMENTS\_ENDIANNESS,                                \

    },                                                                             \

    {                                                                              \

        .period = BATTERY\_MEASUREMENTS\_PERIOD\_ms, .phase = BATTERY\_MEASUREMENTS\_PHASE\_ms \

    }

CellMeasurements (71Ch)

Add:

/\*\* CAN message properties for minimum and maximum values. Required properties are:

 \* - Message ID

 \* - Identifier type (standard or extended)

 \* - Message period and phase in ms

 \* - Endianness of message data @{\*/

#define CELL\_MEASUREMENTS\_ID         (0x71Cu)

#define CELL\_MEASUREMENTS\_ID\_TYPE    (CAN\_STANDARD\_IDENTIFIER\_11\_BIT)

#define CELL\_MEASUREMENTS\_PERIOD\_ms  (100u)

#define CELL\_MEASUREMENTS\_PHASE\_ms   (40u)

#define CELL\_MEASUREMENTS\_ENDIANNESS (CAN\_big\_ENDIAN)

/\*\*@}\*/

#define CELL\_MEASUREMENTS\_MESSAGE                                                             \

    {                                                                                                    \

        .id         = CELL\_MEASUREMENTS\_ID,                                                   \

        .idType     = CELL\_MEASUREMENTS\_ID\_TYPE,                                              \

        .dlc        = CAN\_DEFAULT\_DLC,                                                                   \

        .endianness = CELL\_MEASUREMENTS\_ENDIANNESS,                                           \

    },                                                                                                   \

    {                                                                                                    \

        .period = CELL\_MEASUREMENTS\_PERIOD\_ms, .phase = CELL\_MEASUREMENTS\_PHASE\_ms \

    }

ModuleLimits (71D)

Add:

/\*\* CAN message properties for BMS limit values. Required properties are:

 \* - Message ID

 \* - Identifier type (standard or extended)

 \* - Message period and phase in ms

 \* - Endianness of message data @{\*/

#define MODULE\_LIMITS\_ID         (0x71Du)

#define MODULE\_LIMITS\_ID\_TYPE    (CAN\_STANDARD\_IDENTIFIER\_11\_BIT)

#define MODULE\_LIMITS\_PERIOD\_ms  (100u)

#define MODULE\_LIMITS\_PHASE\_ms   (30u)

#define MODULE\_LIMITS\_ENDIANNESS (CAN\_BIG\_ENDIAN)

/\*\*@}\*/

#define MODULE\_LIMITS\_MESSAGE                                                   \

    {                                                                                \

        .id         = MODULE\_LIMITS\_ID,                                         \

        .idType     = MODULE\_LIMITS\_ID\_TYPE,                                    \

        .dlc        = CAN\_DEFAULT\_DLC,                                               \

        .endianness = MODULE\_LIMITS\_ENDIANNESS,                                 \

    },                                                                               \

    {                                                                                \

        .period = MODULE\_LIMITS\_PERIOD\_ms, .phase = MODULE\_LIMITS\_PHASE\_ms \

    }

ModuleStateEstimation (71Bh)

Add:

/\*\* CAN message properties for pack state estimation values. Required properties are:

 \* - Message ID

 \* - Identifier type (standard or extended)

 \* - Message period and phase in ms

 \* - Endianness of message data @{\*/

#define MODULE\_STATE\_ESTIMATION\_ID         (0x71Bu)

#define MODULE\_STATE\_ESTIMATION\_ID\_TYPE    (CAN\_STANDARD\_IDENTIFIER\_11\_BIT)

#define MODULE\_STATE\_ESTIMATION\_PERIOD\_ms  (1000u)

#define MODULE\_STATE\_ESTIMATION\_PHASE\_ms   (50u)

#define MODULE\_STATE\_ESTIMATION\_ENDIANNESS (CAN\_BIG\_ENDIAN)

/\*\*@}\*/

#define MODULE\_STATE\_ESTIMATION\_MESSAGE                                                            \

    {                                                                                                  \

        .id         = MODULE\_STATE\_ESTIMATION\_ID,                                                  \

        .idType     = MODULE\_STATE\_ESTIMATION\_ID\_TYPE,                                             \

        .dlc        = CAN\_DEFAULT\_DLC,                                                                 \

        .endianness = MODULE\_STATE\_ESTIMATION\_ENDIANNESS,                                          \

    },                                                                                                 \

    {                                                                                                  \

        .period = MODULE\_STATE\_ESTIMATION\_PERIOD\_ms, .phase = MODULE\_STATE\_ESTIMATION\_PHASE\_ms \

    }

can\_cfg\_tx.c

BatteryLimits (351h)

Add:

{CAN\_NODE\_1, BATTERY\_LIMITS\_MESSAGE, &BatteryLimits, NULL\_PTR},

BatteryMeasurements (356h)

Add:

{CAN\_NODE\_1, BATTERY\_MEASUREMENTS\_MESSAGE, &BatteryMeasurements, NULL\_PTR},

CellMeasurements (71Ch)

Add:

{CAN\_NODE\_1, CELL\_MEASUREMENTS\_MESSAGE, &CellMeasurements, NULL\_PTR},

ModuleLimits (71Dh)

Add:

{CAN\_NODE\_1, MODULE\_LIMITS\_MESSAGE, &ModuleLimits, NULL\_PTR},

ModuleStateEstimation (71Bh)

Add:

{CAN\_NODE\_1, MODULE\_STATE\_ESTIMATION\_MESSAGE, &ModuleStateEstimation, NULL\_PTR},

can\_cbs\_tx.h

BatteryLimits (351h)

Add:

/\*\*

 \* @brief can tx callback function for limit values

 \* @param[in] message     contains the message ID, DLC and endianness

 \* @param[in] pCanData    payload of can frame

 \* @param[in] pMuxId      multiplexer for multiplexed CAN messages

 \* @param[in] kpkCanShim  shim to the database entries

 \*/

extern uint32\_t BatteryLimits(

    CAN\_MESSAGE\_PROPERTIES\_s message,

    uint8\_t \*pCanData,

    uint8\_t \*pMuxId,

    const CAN\_SHIM\_s \*const kpkCanShim);

BatteryMeasurements (356h)

Add:

/\*\*

 \* @brief can tx callback function for pack values values

 \* @param[in] message     contains the message ID, DLC and endianness

 \* @param[in] pCanData    payload of can frame

 \* @param[in] pMuxId      multiplexer for multiplexed CAN messages

 \* @param[in] kpkCanShim  shim to the database entries

 \*/

extern uint32\_t BatteryMeasurements(

    CAN\_MESSAGE\_PROPERTIES\_s message,

    uint8\_t \*pCanData,

    uint8\_t \*pMuxId,

    const CAN\_SHIM\_s \*const kpkCanShim);

CellMeasurements (71Ch)

Add:

/\*\*

 \* @brief can tx callback function for min/max values

 \* @param[in] message     contains the message ID, DLC and endianness

 \* @param[in] pCanData    payload of can frame

 \* @param[in] pMuxId      multiplexer for multiplexed CAN messages

 \* @param[in] kpkCanShim  shim to the database entries

 \*/

extern uint32\_t CellMeasurements(

    CAN\_MESSAGE\_PROPERTIES\_s message,

    uint8\_t \*pCanData,

    uint8\_t \*pMuxId,

    const CAN\_SHIM\_s \*const kpkCanShim);

ModuleLimits (71Dh)

Add:

/\*\*

 \* @brief can tx callback function for limit values

 \* @param[in] message     contains the message ID, DLC and endianness

 \* @param[in] pCanData    payload of can frame

 \* @param[in] pMuxId      multiplexer for multiplexed CAN messages

 \* @param[in] kpkCanShim  shim to the database entries

 \*/

extern uint32\_t ModuleLimits(

    CAN\_MESSAGE\_PROPERTIES\_s message,

    uint8\_t \*pCanData,

    uint8\_t \*pMuxId,

    const CAN\_SHIM\_s \*const kpkCanShim);

ModuleStateEstimation (71Bh)

Add:

/\*\*

 \* @brief can tx callback function for state estimation values

 \* @param[in] message     contains the message ID, DLC and endianness

 \* @param[in] pCanData    payload of can frame

 \* @param[in] pMuxId      multiplexer for multiplexed CAN messages

 \* @param[in] kpkCanShim  shim to the database entries

 \*/

extern uint32\_t ModuleStateEstimation(

    CAN\_MESSAGE\_PROPERTIES\_s message,

    uint8\_t \*pCanData,

    uint8\_t \*pMuxId,

    const CAN\_SHIM\_s \*const kpkCanShim);

can\_cbs\_tx\_limit-values.c

BatteryLimits (351h)

Add:

extern uint32\_t BatteryLimits(

    CAN\_MESSAGE\_PROPERTIES\_s message,

    uint8\_t \*pCanData,

    uint8\_t \*pMuxId,

    const CAN\_SHIM\_s \*const kpkCanShim) {

    /\* pMuxId is not used here, therefore has to be NULL\_PTR \*/

    FAS\_ASSERT(pMuxId == NULL\_PTR);

    FAS\_ASSERT(message.id == BATTERY\_LIMITS\_ID);

    FAS\_ASSERT(message.idType == BATTERY\_LIMITS\_ID\_TYPE);

    FAS\_ASSERT(message.dlc == CAN\_FOXBMS\_MESSAGES\_DEFAULT\_DLC);

    FAS\_ASSERT(pCanData != NULL\_PTR);

    FAS\_ASSERT(kpkCanShim != NULL\_PTR);

    uint64\_t messageData = 0u;

    DATA\_READ\_DATA(kpkCanShim->pTableSof);

    /\* AXIVION Disable Style Generic-NoMagicNumbers: Signal data defined in .dbc file. \*/

    /\* maximum charge current \*/

    float\_t signalData = (float\_t)kpkCanShim->pTableSof->recommendedContinuousPackChargeCurrent\_mA;

    float\_t offset     = 0.0f;

    float\_t factor     = 0.004f; /\* convert mA to 250mA \*/

    signalData         = (signalData + offset) \* factor;

    uint64\_t data      = (int64\_t)signalData;

    /\* set data in CAN frame \*/

    CAN\_TxSetMessageDataWithSignalData(&messageData, 23u, 16u, data, message.endianness);

    /\* maximum discharge current \*/

    signalData = (float\_t)kpkCanShim->pTableSof->recommendedContinuousPackDischargeCurrent\_mA;

    offset     = 0.0f;

    factor     = 0.004f; /\* convert mA to 250mA \*/

    signalData = (signalData + offset) \* factor;

    data       = (int64\_t)signalData;

    /\* set data in CAN frame \*/

    CAN\_TxSetMessageDataWithSignalData(&messageData, 39u, 16u, data, message.endianness);

    /\* maximum pack voltage \*/

    signalData = (float\_t)(BS\_NR\_OF\_CELL\_BLOCKS\_PER\_MODULE \* BC\_VOLTAGE\_MAX\_MOL\_mV);

    offset     = 0.0f;

    factor     = 0.01f; /\* convert mV to 1V \*/

    signalData = (signalData + offset) \* factor;

    data       = (int64\_t)signalData;

    /\* set data in CAN frame \*/

    CAN\_TxSetMessageDataWithSignalData(&messageData, 7u, 16u, data, message.endianness);

    /\* AXIVION Enable Style Generic-NoMagicNumbers: \*/

    /\* now copy data in the buffer that will be used to send data \*/

    CAN\_TxSetCanDataWithMessageData(messageData, pCanData, message.endianness);

    return 0;

}

ModuleLimits (71D)

Add:

extern uint32\_t ModuleLimits(

    CAN\_MESSAGE\_PROPERTIES\_s message,

    uint8\_t \*pCanData,

    uint8\_t \*pMuxId,

    const CAN\_SHIM\_s \*const kpkCanShim) {

    /\* pMuxId is not used here, therefore has to be NULL\_PTR \*/

    FAS\_ASSERT(pMuxId == NULL\_PTR);

    FAS\_ASSERT(message.id == MODULE\_LIMITS\_ID);

    FAS\_ASSERT(message.idType == MODULE\_LIMITS\_ID\_TYPE);

    FAS\_ASSERT(message.dlc == CAN\_FOXBMS\_MESSAGES\_DEFAULT\_DLC);

    FAS\_ASSERT(pCanData != NULL\_PTR);

    FAS\_ASSERT(kpkCanShim != NULL\_PTR);

    uint64\_t messageData = 0u;

    DATA\_READ\_DATA(kpkCanShim->pTableSof);

    /\* AXIVION Disable Style Generic-NoMagicNumbers: Signal data defined in .dbc file. \*/

    /\* maximum charge current \*/

    float\_t signalData = (float\_t)kpkCanShim->pTableSof->recommendedContinuousPackChargeCurrent\_mA;

    float\_t offset     = 0.0f;

    float\_t factor     = 0.004f; /\* convert mA to 250mA \*/

    signalData         = (signalData + offset) \* factor;

    uint64\_t data      = (int64\_t)signalData;

    /\* set data in CAN frame \*/

    CAN\_TxSetMessageDataWithSignalData(&messageData, 7u, 16u, data, message.endianness);

    /\* maximum discharge current \*/

    signalData = (float\_t)kpkCanShim->pTableSof->recommendedContinuousPackDischargeCurrent\_mA;

    offset     = 0.0f;

    factor     = 0.004f; /\* convert mA to 250mA \*/

    signalData = (signalData + offset) \* factor;

    data       = (int64\_t)signalData;

    /\* set data in CAN frame \*/

    CAN\_TxSetMessageDataWithSignalData(&messageData, 23u, 16u, data, message.endianness);

    return 0;

}

can\_cbs\_tx\_system-values.c

BatteryMeasurements (356h)

Add:

extern uint32\_t BatteryMeasurements(

    CAN\_MESSAGE\_PROPERTIES\_s message,

    uint8\_t \*pCanData,

    uint8\_t \*pMuxId,

    const CAN\_SHIM\_s \*const kpkCanShim) {

    /\* pMuxId is not used here, therefore has to be NULL\_PTR \*/

    FAS\_ASSERT(pMuxId == NULL\_PTR);

    FAS\_ASSERT(message.id == BATTERY\_MEASUREMENTS\_ID);

    FAS\_ASSERT(message.idType == BATTERY\_MEASUREMENTS\_ID\_TYPE);

    FAS\_ASSERT(message.dlc == CAN\_FOXBMS\_MESSAGES\_DEFAULT\_DLC);

    FAS\_ASSERT(pCanData != NULL\_PTR);

    FAS\_ASSERT(kpkCanShim != NULL\_PTR);

    uint64\_t messageData = 0u;

    /\* Read database entry \*/

    DATA\_READ\_DATA(kpkCanShim->pTablePackValues);

    DATA\_READ\_DATA(kpkCanShim->pTableMinMax);

    int16\_t packMaximumTemperature\_ddegC = INT16\_MIN;

    if (0u == BMS\_GetNumberOfConnectedStrings()) {

        /\* Calculate min/max values of complete pack if all slice switches are open \*/

        for (uint8\_t s = 0u; s < BS\_NR\_OF\_STRINGS; s++) {

            if (kpkCanShim->pTableMinMax->maximumCellVoltage\_mV[s] >= packMaximumVoltage\_mV) {

                packMaximumVoltage\_mV = kpkCanShim->pTableMinMax->maximumCellVoltage\_mV[s];

            }

        }

    } else {

        /\* Calculate min/max values of connected slices \*/

        for (uint8\_t s = 0u; s < BS\_NR\_OF\_STRINGS; s++) {

            if (BMS\_IsStringClosed(s) == true) {

                if (kpkCanShim->pTableMinMax->maximumCellVoltage\_mV[s] >= packMaximumVoltage\_mV) {

                    packMaximumVoltage\_mV = kpkCanShim->pTableMinMax->maximumCellVoltage\_mV[s];

                }

            }

        }

    }

    /\* AXIVION Disable Style Generic-NoMagicNumbers: Signal data defined in .dbc file. \*/

    /\* Battery voltage \*/

    float\_t signalData = kpkCanShim->pTablePackValues->batteryVoltage\_mV;

    float\_t offset     = 0.0f;

    float\_t factor     = 0.01f; /\* convert mV to 100mV \*/

    signalData         = (signalData + offset) \* factor;

    uint64\_t data      = (uint64\_t)signalData;

    /\* set data in CAN frame \*/

    CAN\_TxSetMessageDataWithSignalData(&messageData, 7u, 16u, data, message.endianness);

    /\* System current \*/

    signalData = kpkCanShim->pTablePackValues->packCurrent\_mA;

    offset     = 0.0f;

    factor     = 0.1f; /\* convert mA to 10mA \*/

    signalData = (signalData + offset) \* factor;

    data       = (int64\_t)signalData;

    /\* set data in CAN frame \*/

    CAN\_TxSetMessageDataWithSignalData(&messageData, 23u, 16u, data, message.endianness);

    /\* Maximum cell temperature \*/

    signalData = (float\_t)packMaximumTemperature\_ddegC;

    offset     = 0.0f;

    factor     = 0.1f; /\* convert ddegC to degC \*/

    signalData = (signalData + offset) \* factor;

    data       = (int64\_t)signalData;

    /\* set data in CAN frame \*/

    CAN\_TxSetMessageDataWithSignalData(&messageData, 39u, 16u, data, message.endianness);

    /\* AXIVION Enable Style Generic-NoMagicNumbers: \*/

    /\* now copy data in the buffer that will be used to send data \*/

    CAN\_TxSetCanDataWithMessageData(messageData, pCanData, message.endianness);

    return 0;

}

Can\_cbs\_tx\_minimum-maximum-values.c

CellMeasurements (71Ch)

Add:

extern uint32\_t CellMeasurements(

    CAN\_MESSAGE\_PROPERTIES\_s message,

    uint8\_t \*pCanData,

    uint8\_t \*pMuxId,

    const CAN\_SHIM\_s \*const kpkCanShim) {

    FAS\_ASSERT(message.id == CELL\_MEASUREMENTS\_ID);

    FAS\_ASSERT(message.idType == CELL\_MEASUREMENTS\_ID\_TYPE);

    FAS\_ASSERT(message.dlc == CAN\_FOXBMS\_MESSAGES\_DEFAULT\_DLC);

    FAS\_ASSERT(pCanData != NULL\_PTR);

    FAS\_ASSERT(pMuxId == NULL\_PTR); /\* pMuxId is not used here, therefore has to be NULL\_PTR \*/

    FAS\_ASSERT(kpkCanShim != NULL\_PTR);

    uint64\_t messageData = 0u;

    DATA\_READ\_DATA(kpkCanShim->pTableMinMax);

    int16\_t packMaximumVoltage\_mV        = INT16\_MIN;

    int16\_t packMinimumVoltage\_mV        = INT16\_MAX;

    if (0u == BMS\_GetNumberOfConnectedStrings()) {

        /\* Calculate min/max values of complete pack if all slice switches are open \*/

        for (uint8\_t s = 0u; s < BS\_NR\_OF\_STRINGS; s++) {

            if (kpkCanShim->pTableMinMax->maximumCellVoltage\_mV[s] >= packMaximumVoltage\_mV) {

                packMaximumVoltage\_mV = kpkCanShim->pTableMinMax->maximumCellVoltage\_mV[s];

            }

            if (kpkCanShim->pTableMinMax->minimumCellVoltage\_mV[s] <= packMinimumVoltage\_mV) {

                packMinimumVoltage\_mV = kpkCanShim->pTableMinMax->minimumCellVoltage\_mV[s];

            }

        }

    } else {

        /\* Calculate min/max values of connected slices \*/

        for (uint8\_t s = 0u; s < BS\_NR\_OF\_STRINGS; s++) {

            if (BMS\_IsStringClosed(s) == true) {

                if (kpkCanShim->pTableMinMax->maximumCellVoltage\_mV[s] >= packMaximumVoltage\_mV) {

                    packMaximumVoltage\_mV = kpkCanShim->pTableMinMax->maximumCellVoltage\_mV[s];

                }

                if (kpkCanShim->pTableMinMax->minimumCellVoltage\_mV[s] <= packMinimumVoltage\_mV) {

                    packMinimumVoltage\_mV = kpkCanShim->pTableMinMax->minimumCellVoltage\_mV[s];

                }

            }

        }

    }

    /\* AXIVION Disable Style Generic-NoMagicNumbers: Signal data defined in .dbc file. \*/

    /\* Minimum cell voltage \*/

    float\_t signalData = (float\_t)packMinimumVoltage\_mV;

    float\_t offset     = 0.0f;

    float\_t factor     = 1.0f;

    signalData         = (signalData + offset) \* factor;

    uint64\_t data      = (int64\_t)signalData;

    /\* set data in CAN frame \*/

    CAN\_TxSetMessageDataWithSignalData(&messageData, 39u, 16u, data, message.endianness);

    /\* Maximum cell voltage \*/

    signalData = (float\_t)packMaximumVoltage\_mV;

    offset     = 0.0f;

    factor     = 1.0f;

    signalData = (signalData + offset) \* factor;

    data       = (int64\_t)signalData;

    /\* set data in CAN frame \*/

    CAN\_TxSetMessageDataWithSignalData(&messageData, 7u, 16u, data, message.endianness);

    /\* now copy data in the buffer that will be used to send data \*/

    CAN\_TxSetCanDataWithMessageData(messageData, pCanData, message.endianness);

    return 0;

}

can\_cbs\_tx\_pack-state-estimation.c

ModuleStateEstimation (71Bh)

Add:

extern uint32\_t ModuleStateEstimation(

    CAN\_MESSAGE\_PROPERTIES\_s message,

    uint8\_t \*pCanData,

    uint8\_t \*pMuxId,

    const CAN\_SHIM\_s \*const kpkCanShim) {

    FAS\_ASSERT(message.id == MODULE\_STATE\_ESTIMATION\_ID);

    FAS\_ASSERT(message.idType == MODULE\_STATE\_ESTIMATION\_ID\_TYPE);

    FAS\_ASSERT(message.dlc <= CAN\_MAX\_DLC);

    FAS\_ASSERT(pCanData != NULL\_PTR);

    FAS\_ASSERT(pMuxId == NULL\_PTR); /\* pMuxId is not used here, therefore has to be NULL\_PTR \*/

    FAS\_ASSERT(kpkCanShim != NULL\_PTR);

    uint64\_t messageData = 0u;

    float\_t minimumStringSoc\_perc   = FLT\_MAX;

    float\_t maximumStringSoc\_perc   = FLT\_MIN;

    DATA\_READ\_DATA(kpkCanShim->pTableSox);

    /\* Read database entry \*/

    DATA\_READ\_DATA(kpkCanShim->pTablePackValues);

    /\* Check current direction  \*/

    if (BMS\_GetBatterySystemState() == BMS\_CHARGING) {

        /\* If battery system is charging use maximum values \*/

        for (uint8\_t s = 0u; s < BS\_NR\_OF\_STRINGS; s++) {

            if (BMS\_IsStringClosed(s) == true) {

                if (maximumStringSoc\_perc < kpkCanShim->pTableSox->maximumSoc\_perc[s]) {

                    maximumStringSoc\_perc = kpkCanShim->pTableSox->maximumSoc\_perc[s];

                }

            }

        }

    } else {

        /\* If battery system is discharging or at rest use minimum values \*/

        for (uint8\_t s = 0u; s < BS\_NR\_OF\_STRINGS; s++) {

            if (BMS\_IsStringClosed(s) == true) {

                if (minimumStringSoc\_perc > kpkCanShim->pTableSox->minimumSoc\_perc[s]) {

                    minimumStringSoc\_perc = kpkCanShim->pTableSox->minimumSoc\_perc[s];

                }

            }

        }

    }

    float\_t packSoc\_perc       = 0.0f;

    /\* Calculate pack value \*/

    if (BMS\_GetNumberOfConnectedStrings() != 0u) {

        if (BMS\_GetBatterySystemState() == BMS\_CHARGING) {

            packSoc\_perc = (BMS\_GetNumberOfConnectedStrings() \* maximumStringSoc\_perc) / BS\_NR\_OF\_STRINGS;

        } else {

            packSoc\_perc = (BMS\_GetNumberOfConnectedStrings() \* minimumStringSoc\_perc) / BS\_NR\_OF\_STRINGS;

        }

    } else {

        packSoc\_perc      = 0.0f;

    }

    /\* SOC \*/

    float\_t signalData = packSoc\_perc;

    float\_t offset     = 0.0f;

    float\_t factor     = 100.0f; /\* convert from perc to 0.01perc \*/

    signalData         = (signalData + offset) \* factor;

    uint64\_t data      = (int64\_t)signalData;

    /\* set data in CAN frame \*/

    CAN\_TxSetMessageDataWithSignalData(&messageData, 7u, 16u, data, message.endianness);

    /\* Battery voltage \*/

    float\_t signalData = kpkCanShim->pTablePackValues->batteryVoltage\_mV;

    float\_t offset     = 0.0f;

    float\_t factor     = 0.01f; /\* convert mV to 100mV \*/

    signalData         = (signalData + offset) \* factor;

    uint64\_t data      = (uint64\_t)signalData;

    /\* set data in CAN frame \*/

    CAN\_TxSetMessageDataWithSignalData(&messageData, 39u, 16u, data, message.endianness);

    /\* System current \*/

    signalData = kpkCanShim->pTablePackValues->packCurrent\_mA;

    offset     = 0.0f;

    factor     = 0.1f; /\* convert mA to 10mA \*/

    signalData = (signalData + offset) \* factor;

    data       = (int64\_t)signalData;

    /\* set data in CAN frame \*/

    CAN\_TxSetMessageDataWithSignalData(&messageData, 55u, 16u, data, message.endianness);

    /\* now copy data in the buffer that will be used to send data \*/

    CAN\_TxSetCanDataWithMessageData(messageData, pCanData, message.endianness);

    return 0;

}