UM1583
User manual

STSW-STM8A-J2602 software package

Introduction

This user manual provides application programmers with detailed information on how to use the SAE J2602 software package for master and slave nodes.

The STSW-STM8A-J2602 software package has been built starting from the LIN2.1 software package (STSW-STM8A-LIN), which supports LIN 1.3, 2.0 and 2.1.

Most of the source code and APIs are common to LIN 2.0 and J2602. Therefore, the software architecture is similar, and this document describes only the J2602-specific variables, macros and function prototypes. The document also distinguishes the variables, macros and function prototypes which are internal to the driver from the variables, macros and function prototypes which can be customized and used by the application software.

Before reading this document, the user must be familiar with User Manual *LIN2.1 software package* (UM0941).

Scope

STMicroelectronics implementation is compliant with the J2602 specification. It also includes some additional functions to cover the Ford car maker specification.

This version supports the STM8 Cosmic compiler.

User profile

The users should be familiar with the concept of networks and in particular LIN networks.

Software package availability

STMicroelectronics J2602 software package (STSW-STM8A-J2602) is available on the company website at *www.st.com*.

References

- SAE J2602-1 LIN Network for Vehicle Applications, dated September 2005
- SAE J2602-2 LIN Network for Vehicle Application Conformance Test, dated September 2005
- SAE J2602-3 LIN Network for Vehicle Application NCF File Definition, dated January 2010
- LIN 2.0 standard

Contents

List of tables

1 Getting started

Once the package is installed, two folders are displayed:

- Lingen
- Project

The user must first configure the .lgn file in the Lingen directory, and then launch the .bat file in order to generate the source code corresponding to the LIN description file (.ldf file).

The second step is to integrate the library in the application. The project folder contains a demonstration (application example) in the Project/demo subfolder, and the driver in the Project/src subfolder.

1.1 Source code generation according to LIN description file

1.2 Project example

While the project example of the LIN software package is based on a makefile, the project example of the J2602 software package can be built in ST Visual Develop.

The ST toolset and the Cosmic compiler must previously be installed.

Then user can open the J2602_Package_workspace.stw workspace from ST Visual Develop.

This workspace contains four projects:

- J2602_Package_Master_project
- J2602_Package_Slave_project
- J2602_Package_Ford_Master_project
- J2602_Package_Ford_Slave_project

The user can set the active project and rebuild it.

2 Lingen tool

The lingen.exe tool can detect a LIN description file compliant with the J2602 specification and generate configuration files for the J2602 software package with the help of the .lgn file (see example in *Section 2.1*).

In order to enable the Ford specification, the user must add a keyword in the .lgn file (see example in *Section 2.2*).

2.1 .lgn file example

```
// lingen control file for lin basic demo
Interfaces
{
  SCI1: "C:\STM8A_J2602_Package_1.1\Lingen\J2602_basic_demo.ldf";
}
LIN_use_default_frame_ids;
```

2.2 .lgn file example covering the Ford specification

```
// lingen control file for lin basic demo
Interfaces
{
  SCI1: "C:\tmp\STM8A_J2602_Package_1.1\Lingen\J2602_basic_ford_demo.ldf";
}
Ford_Standard;
LIN_use_default_frame_ids;
```

2.3 .ldf file example

```
/**************************************************************************
//
// file:        lin_basic_demo.ldf
//
// description: LIN decription file (LDF) for the LIN 2.1
//              driver basic demo
//
// author:      Giuseppe Stefano Fazio
//
// date:        2008-03-17
//
//
// (c) 2008 STMicroelectronics
//
//
```

```
//*********************************************************************/
//
// History
//
// 2008-03-12 v1.00   gsf  created
// 2008-03-12 v1.00   gsf  added Diagnostic tables
//
//


//
// header
//
LIN_description_file;
LIN_protocol_version = "J2602_1_1.0";
LIN_language_version = "J2602_3_1.0";

//
// cluster definitions
//
LIN_speed = 10.417 kbps;

Nodes
{
  Master: master, 10 ms, 0.1 ms;
  Slaves: slave1;
}


//
// definition of the signals
//
Signals
{
  //
  // master node signals
  //

  //
  // the DIP state of the master node board
  // masterDIPState1: DIPs 1 and 2
  // masterDIPState2: DIPs 3 to 6
  // masterDIPState3: DIP 7
  // masterDIPSTate4: DIP 8
  masterDIPState1: 2, 0, master, slave1;
```

```
  masterDIPState2: 4, 0, master, slave1;
  masterDIPState3: 1, 0, master, slave1;
  masterDIPState4: 1, 0, master, slave1;


  //
  // slave1 signals
  //

  //
  // the DIP state of the slave1 node board
  // slave1DIPState1: DIPs 1 and 2
  // slave1DIPState2: DIPs 3 to 6
  // slave1DIPState3: DIP 7
  // slave1DIPSTate4: DIP 8
  slave1DIPState1: 2, 0, slave1, master;
  slave1DIPState2: 4, 0, slave1, master;
  slave1DIPState3: 1, 0, slave1, master;
  slave1DIPState4: 1, 0, slave1, master;


  // error signal
  errorSignalSlave1: 1, 0, slave1, master;
  Status_J2602_1_Bit1: 1,0,slave1,master;
  Status_J2602_1_Bit2: 1,0,slave1,master;
  Status_J2602_1_Bit3: 1,0,slave1,master;
  Status_J2602_1_Bit4: 1,0,slave1,master;
  Status_J2602_1_Bit5: 1,0,slave1,master;
  Comm_error_J2602_1 : 3,0,slave1,master;

  BC_Frame1 : 64, {1,2,3,4,5,6,7,8}, master,slave1;
  BC_Frame2 : 64, {0x0a,0x0b,0x0c,0x0d,0x0e,0xf,0x09,0x10}, master,slave1;
  BC_Frame3 : 64, {8,7,6,5,4,3,2,1}, master,slave1;
  BC_Frame4 : 64, {0,0,0,0xb,0xc,0xd,0xe,0xf}, master,slave1;

  test_sig : 56, {0,0,0,0,0,0,0}, slave1,master;

}

//
// defintion of diagnostic signals
// (optional but recommended)
//
Diagnostic_signals
{
```

```
  MasterReqB0:8,0;
  MasterReqB1:8,0;
  MasterReqB2:8,0;
  MasterReqB3:8,0;
  MasterReqB4:8,0;
  MasterReqB5:8,0;
  MasterReqB6:8,0;
  MasterReqB7:8,0;
  SlaveRespB0:8,0;
  SlaveRespB1:8,0;
  SlaveRespB2:8,0;
  SlaveRespB3:8,0;
  SlaveRespB4:8,0;
  SlaveRespB5:8,0;
  SlaveRespB6:8,0;
  SlaveRespB7:8,0;
}


//
// frame definitions
//
Frames
{
  //
  // frames published by the master
  //
  frmM1: 0, master, 2
  {
    masterDIPState1, 0;
  }

  frmM2: 1, master, 1
  {
    masterDIPState2, 0;
  }

  frmM3: 2, master, 1
  {
    masterDIPState3, 0;
    masterDIPState4, 1;
  }

     Demo_LED_Broadcast1 : 56,master,8{
               BC_Frame1,0;
```

```
    }

    Demo_LED_Broadcast2 : 57,master,8{
              BC_Frame2,0;
    }

    Demo_LED_Broadcast3 : 58,master,8{
              BC_Frame3,0;
    }

    Demo_LED_Broadcast4 : 59,master,8{
              BC_Frame4,0;
    }

//
// frame spublished by slave1
//
frmS1: 3, slave1, 2
{
 Status_J2602_1_Bit1 ,0;
  Status_J2602_1_Bit2 ,1;
  Status_J2602_1_Bit3 ,2;
  Status_J2602_1_Bit4 ,3;
  Status_J2602_1_Bit5 ,4;
  Comm_error_J2602_1 ,5;

  slave1DIPState1, 8;
}

frmS2: 4, slave1, 2
{
 Status_J2602_1_Bit1 ,0;
  Status_J2602_1_Bit2 ,1;
  Status_J2602_1_Bit3 ,2;
  Status_J2602_1_Bit4 ,3;
  Status_J2602_1_Bit5 ,4;
  Comm_error_J2602_1 ,5;

  slave1DIPState2, 8;
}

frmS3: 5, slave1, 2
{
   Status_J2602_1_Bit1 ,0;
  Status_J2602_1_Bit2 ,1;
```

```
    Status_J2602_1_Bit3 ,2;
    Status_J2602_1_Bit4 ,3;
    Status_J2602_1_Bit5 ,4;
    Comm_error_J2602_1 ,5;

    slave1DIPState3, 8;
    slave1DIPState4, 9;
    errorSignalSlave1, 10;


  }

  frmS4: 6, slave1, 8
  {
   Status_J2602_1_Bit1 ,0;
    Status_J2602_1_Bit2 ,1;
    Status_J2602_1_Bit3 ,2;
    Status_J2602_1_Bit4 ,3;
    Status_J2602_1_Bit5 ,4;
    Comm_error_J2602_1 ,5;

     test_sig, 8;
  }
}
//
// definition of diagnostic frames
// (optional but recommended)
//
Diagnostic_frames
{
  MasterReq : 60
  {
    MasterReqB0,0;
    MasterReqB1,8;
    MasterReqB2,16;
    MasterReqB3,24;
    MasterReqB4,32;
    MasterReqB5,40;
    MasterReqB6,48;
    MasterReqB7,56;
  }
  SlaveResp : 61
  {
    SlaveRespB0,0;
    SlaveRespB1,8;
```

```
    SlaveRespB2,16;
    SlaveRespB3,24;
    SlaveRespB4,32;
    SlaveRespB5,40;
    SlaveRespB6,48;
    SlaveRespB7,56;
  }
}

//
// node attributes
// relevant for slave nodes only
//
Node_attributes
{
  slave1
  {
    LIN_protocol = "J2602_1_1.0";

    // the configured diagnostic address
    configured_NAD = 0x60;


    // product id is used to uniquely identify a slave node
    // within a cluster
    product_id = 0x1234, 0x5678, 0x03;

    // definition of the error signal of the slave
    response_error = Comm_error_J2602_1;

    // definition of the timeouts of the slave
    P2_min = 100 ms ;
    ST_min = 20 ms ;

    //
    // the list of configurable frames
    // all frames to be processed by the slave node
    // must get a message id in this section
    //
    configurable_frames
    {
        frmM1 = 0;
        frmM2 = 1;
        frmM3 = 2;
        frmS1 = 3;
```

```
            frmS2 = 4;
            frmS3 = 5;
            frmS4 = 6;
        }
        response_tolerance = 40%;
        wakeup_time = 100 ms;
        poweron_time = 100 ms;
    }
}

//
// definition of schedule tables
//
Schedule_tables
{
    //
    // the normal signals are transferred using this schedule
    // table
    //
    schTab1
    {
        frmM1 delay 20 ms;
        frmS1 delay 20 ms;


        frmM2 delay 20 ms;
        frmS2 delay 20 ms;

        frmM3 delay 20 ms;
        frmS3 delay 20 ms;

        frmS4 delay 20 ms;

        Demo_LED_Broadcast1 delay  40 ms;
        Demo_LED_Broadcast2 delay  40 ms;
        Demo_LED_Broadcast3 delay  40 ms;
        Demo_LED_Broadcast4 delay  40 ms;
    }
    schTab2
    {
     frmS1 delay 20 ms;
    }
}
```

2.4 Ford .ldf file example

```
LIN_description_file;
LIN_protocol_version = "J2602_1_1.0";
LIN_language_version = "J2602_3_1.0";

LIN_speed = 10.417 kbps;

Nodes
{
  Master: BCM, 10 ms, 0.1 ms;
  Slaves: Sensor;
}

//
// definition of the signals
//
Signals
{
  //
  // master node signals
  //

  // the DIP state of the master node board
  masterDIPState1: 2, 0, BCM, Sensor;
  masterDIPState2: 4, 0, BCM, Sensor;
  masterDIPState3: 1, 0, BCM, Sensor;
  masterDIPState4: 1, 0, BCM, Sensor;

  SensorConfigIndex: 8, 0, BCM, Sensor;
  SensorConfigData0: 8, 0, BCM, Sensor;
  SensorConfigData1: 8, 0, BCM, Sensor;
  SensorConfigData2: 8, 0, BCM, Sensor;
  SensorConfigData3: 8, 0, BCM, Sensor;
  SensorConfigData4: 8, 0, BCM, Sensor;
  SensorConfigData5: 8, 0, BCM, Sensor;
  SensorConfigData6: 8, 0, BCM, Sensor;

  BC_Frame1 : 64, {1,2,3,4,5,6,7,8}, BCM, Sensor;
  BC_Frame2 : 64, {0x0a,0x0b,0x0c,0x0d,0x0e,0xf,0x09,0x10}, BCM, Sensor;
  BC_Frame3 : 64, {8,7,6,5,4,3,2,1}, BCM, Sensor;
  BC_Frame4 : 64, {0,0,0,0xb,0xc,0xd,0xe,0xf}, BCM, Sensor;

  //
  // slave1 signals
```

```
  //

  // the DIP state of the slave1 node board
  slave1DIPState1: 2, 0, Sensor, BCM;
  slave1DIPState2: 4, 0, Sensor, BCM;
  slave1DIPState3: 1, 0, Sensor, BCM;
  slave1DIPState4: 1, 0, Sensor, BCM;

   // error signal
   SensorLINStatus : 8, 0, Sensor, BCM;
   SensorPartNumIndex : 8, 0, Sensor, BCM;
   SensorPartNumData0 : 8, 0, Sensor, BCM;
   SensorPartNumData1 : 8, 0, Sensor, BCM;
   SensorPartNumData2 : 8, 0, Sensor, BCM;
   SensorPartNumData3 : 8, 0, Sensor, BCM;
   SensorPartNumData4 : 8, 0, Sensor, BCM;
   SensorPartNumData5 : 8, 0, Sensor, BCM;
}

//
// defintion of diagnostic signals
// (optional but recommended)
//
Diagnostic_signals
{
  MasterReqB0:8,0;
  MasterReqB1:8,0;
  MasterReqB2:8,0;
  MasterReqB3:8,0;
  MasterReqB4:8,0;
  MasterReqB5:8,0;
  MasterReqB6:8,0;
  MasterReqB7:8,0;
  SlaveRespB0:8,0;
  SlaveRespB1:8,0;
  SlaveRespB2:8,0;
  SlaveRespB3:8,0;
  SlaveRespB4:8,0;
  SlaveRespB5:8,0;
  SlaveRespB6:8,0;
  SlaveRespB7:8,0;
}


//
```

```
// frame definitions
//
Frames
{
  //
  // frames published by the BCM
  //
  frmM1: 0, BCM, 2
  {
    masterDIPState1, 0;
  }

  frmM2: 1, BCM, 1
  {
    masterDIPState2, 0;
  }

  frmM3: 2, BCM, 1
  {
    masterDIPState3, 0;
    masterDIPState4, 1;
  }

   frmM4: 6, BCM, 8
  {
    SensorConfigIndex, 0;
    SensorConfigData0, 8;
    SensorConfigData1, 16;
    SensorConfigData2, 24;
    SensorConfigData3, 32;
    SensorConfigData4, 40;
    SensorConfigData5, 48;
    SensorConfigData6, 56;
  }

  Demo_LED_Broadcast1 : 56,BCM,8{
   BC_Frame1,0;
  }

  Demo_LED_Broadcast2 : 57,BCM,8{
   BC_Frame2,0;
  }

  Demo_LED_Broadcast3 : 58,BCM,8{
   BC_Frame3,0;
```

```
  }

  Demo_LED_Broadcast4 : 59,BCM,8{
   BC_Frame4,0;
  }

  //
  // frame spublished by Sensor
  //
  frmS1: 3, Sensor, 2
  {
    SensorLINStatus, 0;
    slave1DIPState1, 8;
  }

  frmS2: 4, Sensor, 2
  {
    SensorLINStatus, 0;
    slave1DIPState2, 8;
  }

  frmS3: 5, Sensor, 2
  {
    SensorLINStatus, 0;
    slave1DIPState3, 8;
    slave1DIPState4, 9;
  }

  frmS4: 7, Sensor, 8
  {
    SensorLINStatus,          0;
    SensorPartNumIndex, 8;
    SensorPartNumData0, 16;
    SensorPartNumData1, 24;
    SensorPartNumData2, 32;
    SensorPartNumData3, 40;
    SensorPartNumData4, 48;
    SensorPartNumData5, 56;
  }
}

//
// definition of diagnostic frames
// (optional but recommended)
//
```

```
Diagnostic_frames
{
  MasterReq : 60
  {
    MasterReqB0,0;
    MasterReqB1,8;
    MasterReqB2,16;
    MasterReqB3,24;
    MasterReqB4,32;
    MasterReqB5,40;
    MasterReqB6,48;
    MasterReqB7,56;
  }
  SlaveResp : 61
  {
    SlaveRespB0,0;
    SlaveRespB1,8;
    SlaveRespB2,16;
    SlaveRespB3,24;
    SlaveRespB4,32;
    SlaveRespB5,40;
    SlaveRespB6,48;
    SlaveRespB7,56;
  }
}

//
// node attributes
// relevant for slave nodes only
//
Node_attributes
{
  Sensor
  {
    LIN_protocol = "J2602_1_1.0";

    // the configured diagnostic address
    configured_NAD = 0x60;

    // product id is used to uniquely identify a slave node
    // within a cluster
    product_id = 0x1234, 0x5678, 0x03;

    // definition of the error signal of the slave
    response_error = SensorLINStatus;
```

```
    // definition of the timeouts of the slave
    P2_min = 100 ms ;
    ST_min = 20 ms ;

    //
    // the list of configurable frames
    // all frames to be processed by the slave node
    // must get a message id in this section
    //
    configurable_frames
    {
        frmM1 = 0;
        frmM2 = 1;
        frmM3 = 2;
        frmS1 = 3;
        frmS2 = 4;
        frmS3 = 5;
        frmM4 = 6;
        frmS4 = 7;
    }
    response_tolerance = 40%;
    wakeup_time = 100 ms;
    poweron_time = 100 ms;
  }
}

//
// definition of schedule tables
//
Schedule_tables
{
  LIN11
  {
    frmM1 delay 20 ms;
    frmS1 delay 20 ms;

    frmM2 delay 20 ms;
    frmS2 delay 20 ms;

    frmM3 delay 20 ms;
    frmS3 delay 20 ms;

    Demo_LED_Broadcast1 delay  40 ms;
    Demo_LED_Broadcast2 delay  40 ms;
```

```
    Demo_LED_Broadcast3 delay  40 ms;
    Demo_LED_Broadcast4 delay  40 ms;
  }

  LINConfig1
  {
   frmM4 delay 20 ms;
   frmM4 delay 20 ms;
   frmM4 delay 20 ms;
   frmM4 delay 20 ms;
   frmM4 delay 20 ms;

   frmS4 delay 20 ms;
   frmS4 delay 20 ms;
   frmS4 delay 20 ms;
   frmS4 delay 20 ms;
   frmS4 delay 20 ms;
  }
}
```

3 Software package source files and header files

The J2602 software package includes the same list of files as the LIN2.1 software package.

\src\lin_version_control.h
\src\ lin.h

\src\arch\ lin_def_arch_include.h
\src\arch\ lin_arch_include.h

\src\arch\stm8\ lin_stm8.h
\src\arch\stm8\ lin_stm8.c
\src\arch\stm8\ lin_def_stm8_gen.h

\src\config\ lin_def.h
\src\config\ lin_def.c
\src\config\ lin_def_stm8.h

\src\diag\lin_diag.c
\src\diag\lin_diag.h
\src\diag\ lin_diag_api.h
\src\diag\ lin_diag_master.c
\src\diag\ lin_diag_master.h
\src\diag\ lin_diag_slave.c
\src\diag\ lin_diag_slave.h

\src\general\ lin_types.h
\src\general\ lin_def_gen.h
\src\general\lin_general.c
\src\general\lin_general.h

\src\master\lin_master.c
\src\master\lin_master.h

\src\slave\lin_slave.c

\src\slave\lin_slave.h

\src\timer\lin_timer.c
\src\timer\lin_timer.h

For the Ford features, the user must compile two additional files:

\src\ford\ford.c
\src\ford\ford.h

4 New internal variables, macros and prototypes in the J2602 software package compared to the LIN 2.1 software package

4.1 New internal variables

These variables are internal to the driver. The user must not access them.

Table 1. New internal variables

Name	Description
`J2602_BC_id`	PID for broadcast frame, Slave only

`J2602_STATUS`
`{`
`l_u8 reset :1;`
`l_u8 tx_resp :1;`
`l_u8 new_frame0 :1;`
`l_u8 new_frame1 :1;`
`l_u8 sresp :1;`
`l_u8 errorstate :7;`
`}t_j2602_status;`
	Status control bits of the J2602 errors and other information
	Slave only
	– `Reset`: a reset request is received if this bit is set to 1.
	– `tx_resp`: the driver needs to give a response frame to the reset request if this bit is set to 1.
	– `new_frame0`: this bit indicates that data is received for the broadcast frame with ID 0x38 or 0x39, when it is set to 1.
	– `new_frame1`: this bit indicates that data is received for the broadcast frame with ID 0x3A or 0x3B, when it is set to 1.
	– `sresp`: the driver needs to give a response frame to the request frame if this bit is set to 1.
	– `errorstate`: bit7-bit0 represent `Reset Response`, `TX error`, `ID Parity Error`, `Byte Field Framing Error`, `Checksum Error`, `Data Error`, `Reset`

`j2602_statusbyte`	J2602 status byte Slave only
`j2602_broadcast_bytes[2][ ]`	Data buffer for broadcast frame Slave only

4.2 New internal macros

These macros are internal to the driver. The user must not access them.

Table 2. New internal macros

Name	Description
`#define LIN_DIAG_PCI_RESET 0x01`	PCI value of the target reset request
`#define LIN_DIAG_SID_RESET 0xB5`	SID value of the target reset request
`#define LIN_DIAG_RESET_DATA_LEN 0x00`	Data length value of the target reset request
`l_bool_wr_new_bc_data0();`	Indicates that data is received for the broadcast frame with ID 0x38 or 0x39
`l_bool_wr_new_bc_data1();`	Indicates that data is received for the broadcast frame with ID 0x3A or 0x3B

4.3 New internal prototypes

These prototypes are internal to the driver. The user must not access them.

Table 3. New internal prototypes

Name	Description
`l_bool_wr_Reset()`	This is for the driver to set the reset request flag to inform the application about a received target or broadcast reset request. Slave only
`J2602_GET_ERRORBITS()`	This macro converts the communication error into status byte. Slave only
`LIN_STATUS_BYTE()`	This function writes the status byte into LIN signal buffer. Slave only
`J2602_CREATE_STATUSBYTE();`	Calls J2602_GET_ERRORBITS() and LIN_STATUS_BYTE(). Slave only

5 Additional internal variables, macros and prototypes in the J2602 software package to cover the Ford specification

5.1 Additional internal variables

These variables are internal to the driver. The user must not access them.

Table 4. Additional internal variables

Name	Description
`l_u8 LIN_ConfigMode`	LIN mode definition Master and slave `LIN_NULL`: normal LIN mode `LIN_CONFIG`: LIN configuration mode
`l_u8 Slave_ConfigData[255]`	Data buffer for configuration data Slave only
`l_u8 ECU_cfg_flag`	Flag to indicate whether the slave has been configured Slave only `FORD_SLAVE_NO_COFIG`: the slave has not been configured yet

5.2 Additional internal macros

These macros are internal to the driver. The user must not access them.

Table 5. Additional internal macros

Name	Description
`Ford_LIN_ConfigMode_test()`	Check whether the master is in configuration mode. Master only
`Ford_LIN_ConfigMode_set()`	Sets the master in configuration mode. Master only
`ecu_config_set()`	Sets the slave configuration flag. Slave only
`ecu_config_clear()`	Clears the slave configuration flag. Slave only
`ecu_config_status()`	Checks the slave configuration flag. Slave only

5.3 Additional internal prototypes

These prototypes are internal to the driver. The user must not access them.

Table 6. Additional internal prototypes

Name	Description
`void recieve_ConfigData(void);`	This function is used to store the configuration data to buffers. Slave only
`void goto_Config_Mode(void);`	This function is used to set slave in Configuration mode. Slave only
`void load_Partnumber_data(void);`	This function is used to load part number data into the signal buffer. Slave only

6 New user APIs in the J2602 software package compared to the LIN 2.1 software package

6.1 New user API prototypes

Table 7. New user API prototypes

Name	Description
`unsigned char l_callback_Reset_Response (void)`	Callback function to check if reset is done. Slave only User should add his own code to check if the system is reset. The return value means: 0 means that there was no reset request. 1 means that the slave is now correctly and consistently reset and the driver has to send a positive answer. 2 means that the slave will not reset after this reset request and the driver has to send a negative answer.
`l_u8* j2602_get_bc_data(l_u8 frame)`	The application uses this function to read the data from the broadcast frames. Slave only Parameters: frame=0 data buffer for broadcast frames with ID 0x38 or 0x39 frame=1 data buffer for broadcast frames with ID 0x3A or 0x3B return: data point to load the data.
`void j2602_hal_target_reset (void)`	This function resets the controller and, if needed the interrupts. It also resets the J2602 status byte, but not the configuration done. User should add his own code to reset the system. Slave only
`l_flg_tst_new_bc_data0()`	This is for the application to check if data from broadcast frames with ID 0x38 or 0x39 was received. Slave only
`l_flg_clr_new_bc_data0()`	This is for the application to reset the flag after fetching the received data from broadcast frames with ID 0x38 or 0x39. Slave only
`l_flg_tst_new_bc_data1()`	This is for the application to check if data from broadcast frames with ID 0x3A or 0x3B was received. Slave only

Table 7. New user API prototypes (continued)

Name	Description
`l_flg_clr_new_bc_data1()`	This is for the application to reset the flag after fetching the received data from broadcast frames with ID 0x3A or 0x3B. Slave only
`l_flg_tst_Reset()`	This is for the application to check if the driver has received a reset request. Slave only
`l_flg_clr_Reset()`	This is for the application to delete the reset flag after beginning to work out or ignore the reset. Slave only
`l_get_nr_databytes()`	This is for the application to check how many bytes the driver extracted from the received broadcast frame (depending on the number of frames assigned to the slave). Slave only

7 Additional user variables, macros and APIs in the J2602 software package to cover the Ford specification

In order to cover the Ford specification, user must add ford.c and ford.h in the project.

7.1 Additional user variables

Table 8. Additional user variables

Name	Description
`l_u8 PartNum[23]`	Data buffer for slave part number Slave only User should give the initial value of the part number.

7.2 Additional user macros

Table 9. Additional user macros

Name	Description
`Ford_LIN_ConfigMode_clear()`	Sets master back to the normal mode. Master only
`max_PartNumIndex`	The maximum PartNumIndex between 0-3 Slave only User should give a value to the parameter.
`ini_PartNumIndex`	The initial PartNumIndex between 0- max_PartNumIndex Slave only User should give a value to the parameter.

7.3 Additional user prototypes

Table 10. Additional user prototypes

Name	Description
`l_u16 Ford_LIN_DTC_SET(l_bool missing, l_u8 status_byte, l_u16 dtc_base);`	This function is used to set DTC. Master only Called by the application
`void Ford_Schedule_switch(l_ifc_handle LIN_ifc, l_schedule_handle schedule_name, l_u8 entry);`	This function is used to switch to a normal schedule table. Master only Called by the application
`void Ford_goto_ConfigMode(l_ifc_handle LIN_ifc, l_schedule_handle schedule_name, l_u8 entry);`	This function is used to switch to a configuration schedule. Master only Called by the application
`void Ford_LINSlavePartNum_Clear(l_ifc_handle LIN_ifc);`	Sets LINSlavePartNum = NULL for each slave node Master only The user should add code to erase the PartNum buffer for all the slave nodes.
`void Ford_J2602_status_report(l_u8 app_error);`	This function is used to extend J2602 Error Handling. Slave node Called by the application
`void save_cfg_data(l_u8 length, l_u8 *data);`	This function is used to save configuration data in EEPROM. Slave node The user should add his own code to save the data in the buffer into the EEPROM.

8 Revision history

Table 11. Document revision history

Date	Revision	Changes
19-Nov-2012	1	Initial release.
16-May-2014	2	Updated the document title and the *Introduction* on the cover page to include STMicroelectronics J2602 software package reference (STSW-STM8A-J2602). Removed the table “Applicable products” on the cover page. No other changes in the content.

Please Read Carefully:

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST’s terms and conditions of sale.

Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN ST’S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

ST PRODUCTS ARE NOT DESIGNED OR AUTHORIZED FOR USE IN: (A) SAFETY CRITICAL APPLICATIONS SUCH AS LIFE SUPPORTING, ACTIVE IMPLANTED DEVICES OR SYSTEMS WITH PRODUCT FUNCTIONAL SAFETY REQUIREMENTS; (B) AERONAUTIC APPLICATIONS; (C) AUTOMOTIVE APPLICATIONS OR ENVIRONMENTS, AND/OR (D) AEROSPACE APPLICATIONS OR ENVIRONMENTS. WHERE ST PRODUCTS ARE NOT DESIGNED FOR SUCH USE, THE PURCHASER SHALL USE PRODUCTS AT PURCHASER’S SOLE RISK, EVEN IF ST HAS BEEN INFORMED IN WRITING OF SUCH USAGE, UNLESS A PRODUCT IS EXPRESSLY DESIGNATED BY ST AS BEING INTENDED FOR “AUTOMOTIVE, AUTOMOTIVE SAFETY OR MEDICAL” INDUSTRY DOMAINS ACCORDING TO ST PRODUCT DESIGN SPECIFICATIONS. PRODUCTS FORMALLY ESCC, QML OR JAN QUALIFIED ARE DEEMED SUITABLE FOR USE IN AEROSPACE BY THE CORRESPONDING GOVERNMENTAL AGENCY.

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.
Information in this document supersedes and replaces all information previously supplied.
The ST logo is a registered trademark of STMicroelectronics. All other names are the property of their respective owners.

STMicroelectronics group of companies

Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Philippines - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

www.st.com