



## **A Framework for Developing SAE J1939 Devices**

Joachim Stolberg, IXXAT Automation

The SAE J1939 standard defines the CAN based data communication between electronic control units (ECU) in heavy duty vehicles like trucks, construction or agricultural vehicles and machines. Although first published in 1998, the SAE J1939 standard became increasingly popular in the last years. Today the suppliers of ECUs for diesel engines, transmissions, brakes etc. are forced to convert their systems to J1939 based communication.

This article describes aspects, requirements and alternatives for implementing J1939 into an ECU and presents solutions based on a commercially available protocol stack.

### ***Demands on a SAE J1939 protocol stack***

When integrating the J1939 protocol stack into devices like an ECU, the first considerations concerns the available resources. ECUs are today equipped with microcontrollers ranging from 8 bit platforms with rather limited resources up to 32 bit systems with plenty of memory, high performance CPUs and running sophisticated real-time operating systems. Regardless of the used micro-controller, a communication stack is always considered as an accessory which should not consume too much resources with respect to processing time or memory. Especially for low-end systems the memory foot-print is critical.

Another important aspect for the implementation of the J1939 protocol are the actual communication requirements of the application. Some applications can be satisfied with only a few messages of up to 8 data bytes length and a fixed device address (reduced network management). Other applications need dynamic device address allocation, the transmission of large data packets (multi-packets) and comprehensive diagnostics functions according to SAE J1939/73. Consequently an J1939 protocol stack must be highly scalable and easily adaptable to the actual needs of an application.

## ***Interfacing application and protocol software***

Primarily, the J1939 protocol stack should provide functions for the transmission and reception of J1939 messages. Generally two different methods for interfacing the application and protocol software are common:

- **Function Interface:** With this type of interface application and protocol software are linked by means of appropriate function calls.
- **Data Interface:** With this type of interface the application and protocol software exchange directly data values (signals) across a shared memory.

To transmit data across a function interface, the application is using a transmit function for sending a complete message. Received messages are either delivered without delay via a call-back function from the protocol software to the application or intermediately stored in a ring buffer, from which the application can read the messages. The function interface provides a tight connection between application and protocol software. Across the function interface a complete CAN-message is exchanged and requires assembling respectively disassembling of the message data by the application software.

Transfer of data via a data interface is performed by means of a shared memory which can be accessed by the application and the protocol software independently. Application and communication software are therefore decoupled to the greatest possible extent. The application can run its own processing cycle and use the data interface to asynchronously transmit and receive data only when required. With this solution assembling and disassembling of the message data is the task of the communication stack. With a data interface the application directly accesses application data.

### **Standard and proprietary messages**

The J1939 standard defines a message catalogue in which standard message formats are specified. This is the base for providing message-specific pack( ) and unpack() functions for any of the used standard messages by the protocol stack.

Additionally, J1939 permits the exchange of vendor-specific proprietary messages. In order to assemble respectively disassemble proprietary messages, the protocol software needs to know the rules for mapping respectively demapping the application data into respective from a proprietary message. Therefore the protocol software has to be configurable with respect to proprietary messages.

### ***A scalable J1939 protocol stack***

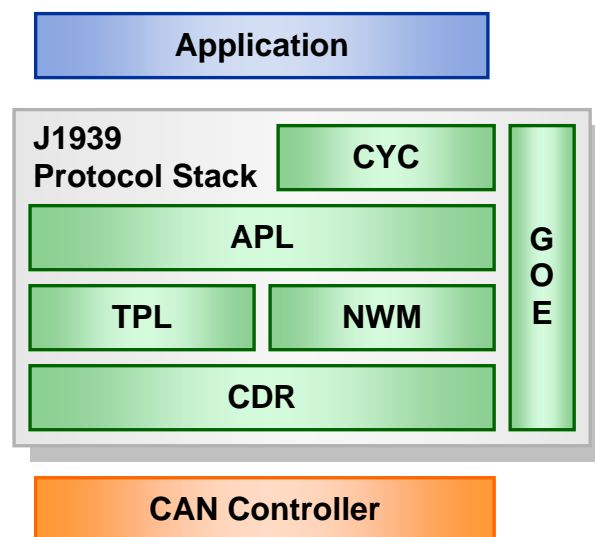
The J1939 protocol stack from IXXAT represents a highly scalable solution which offers various options for adaptation to the actual communication requirements of an application, to reduce resource consumption and to provide easy software integration.

The J1939 protocol software includes the following modules (Fig. 1):

- A CAN Driver (CDR) for accessing of the CAN controller
- A Transport Layer (TPL) module for transfer of segmented messages
- An Application Layer (APL) for providing a function oriented programming interface with additional mapping/demapping macros
- The Cycle module (CYC) for handling and supervision of cyclically transferred messages and for providing a data interface
- A Network Management (NWM) module for the device address claiming procedure
- A Generic Operation System Environment (GOE) for providing an operating system abstraction layer

In order to cover a wide range of target systems, IXXAT offers two versions of the J1939 protocol software:

- The **standard version** for target systems with sufficient data memory (more than 2 kByte).
- The **micro version** for systems with limited data memory (minimum 200 bytes).



*Fig 1: Block diagram of the J1939 protocol software*

The standard version allows the dynamic configuration of the SAE J1939 software via the function interface during runtime. Thus, for example filter rules for receive messages can be modified during runtime. This version also supports more than one software instance (CAN channel) and is designed for use in a real-time operating system environment due to an abstraction layer especially provided for this purpose. The standard version can also be used in systems without an operating system.

The micro version is optimized for use on 8/16-bit CPU systems with limited performance and memory resources. In this case the protocol software has to be configured during compile time in order to have a defined usage of the limited resources. Changing parameters during

run-time is not possible. The C-files representing the desired J1939 configuration are automatically generated by the IXXAT J1939 configuration tool. Finally the complete configuration is statically integrated in the code which considerably reduces the RAM requirements compared to a dynamic configuration used in the standard version. Both software versions have a similar function interface for the transmission and reception of J1939 messages. In the standard version, the function interface is designed in such a way that it can be used by more than one task in the environment of an real-time executive. The standard version can therefore be referred to as “thread-safe”.

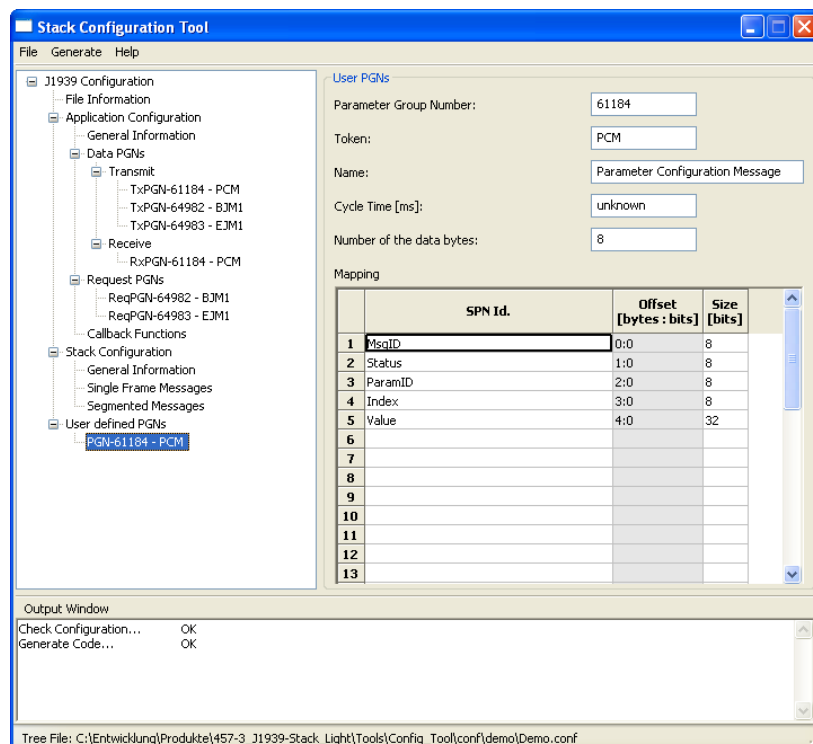
In addition to the function interface which provides a message-oriented interfacing additionally a signal-oriented data interface is provided with the IXXAT J1939 stack. This is accomplished by the so-called cycle module which provides the data interface between protocol stack and application besides monitoring and sending of cyclically messages.

The protocol software has to be called cyclically in order to monitor time-outs and to process transmission protocols. In a system without a real-time executive system (super loop), this is usually performed by calling a process() function in the cyclically processed main program.

In systems controlled by a real-time executive , a separate cyclic task should be installed for calling the process() function.

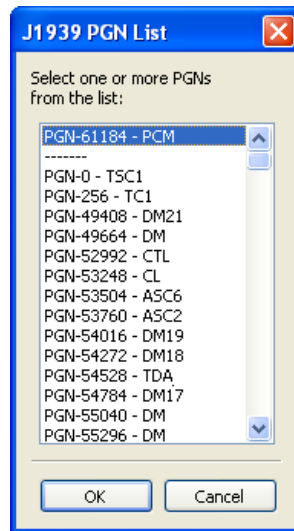
### ***J1939 Protocol software configuration tool***

The graphical J1939 configuration tool which comes with the J1939 protocol stack considerably facilitates the customization and configuration of the micro and standard version (Fig 2).



*Fig 2: J1939 Configuration Tool – Definition of user specific messages*

In addition to setting standard parameters such as device names or the size of ring buffers, the J1939 configuration tool also allows the selection of standard messages from a pre-defined message catalogue (Fig. 3). The filter and distributor functions in the protocol software are configured and access macros are generated in form of C-files to be integrated directly into the software project based on these settings. These access macros are defining the mapping rules between the J1939 messages and the process parameters or variables used in the application for receiving and transmitting. Besides the configuration of standard messages, the J1939 configuration tool also allows the definition of proprietary messages.



*Fig 3: J1939 Configuration Tool – Standard message catalogue based on Parameter Group Numbers PGNs*

The most important advantage of the J1939 configuration tool is the fact that a complete configuration is generated by one tool therefore guaranteeing the consistency of all parameters. Thus problems like incorrect sizes of FIFO memories can be avoided. The J1939 configuration tool offers furthermore the option of generating a complete C-code framework for an J1939 application. This feature is especially useful during the “first steps” with J1939 because it supports a very quick entry into the J1939 communication.

### ***An example application***

The following lines of code show the initialization and operation of the J1939 protocol software. In this example, a received parameter is read from a buffer via the data interface. The message has been configured with the J1939 configuration tool as a receive message with the required parameter (engine oil pressure, SPN100). Since the configuration of the protocol software has been performed via the J1939 configuration tool, all configuration parameters are automatically been taken into account by calling the function `APL_Init()`. Only the local time in milliseconds has to be passed (via function `time()`) to the protocol software.

```

void main(void)
{
    // Initialize the J1939
    // protocol software
    APL_Init(time());

    while(1)
    {
        // Keep the J1939 protocol
        // software running
        APL_Main(time());
        do_other_tasks();
    }
}

```

The cycle module monitors the reception of the configured message and stores the data in a buffer, which has been generated by the J1939 configuration tool for this purpose. The actual parameter can be read by the application via the macro UNPACK\_EFL\_P1\_SPN100() which is also been generated by the J1939 configuration tool.

```

// read SPN100 (engine oil pressure)
// from PGN65263 (EFL/P1)
val = UNPACK_EFL_P1_SPN100();

```

## **Conclusion**

The IXXAT J1939 protocol software is available in a version which is fully configurable at runtime and a version which provides minimum footprint. Depending on the application requirements a function- or a data-based application interface can be used. By means of a configuration tool the generation of a consistent parameter set and of a complete C-Code framework is supported.

IXXAT furthermore offers a J1939 analyzing module for its canAnalyser which especially supports the developer in verifying and testing the communication capabilities of J1939 devices and systems.