Technical Document - Confidential

GSM PROTOCOL STACK

G23

SOCKET – SOCKET

DRIVER INTERFACE

Document Number:	6392.199.99.101
Version:	0.4
Status:	Draft
Approval Authority:	
Creation Date:	1999-Aug-19
Last changed:	2015-Mar-08 by XINTEGRA
File Name:	6392_199_socket.doc

Important Notice

Texas Instruments Incorporated and/or its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products, software and services at any time and to discontinue any product, software or service without notice. Customers should obtain the latest relevant information during product design and before placing orders and should verify that such information is current and complete.

All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment. TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI products, software and/or services. To minimize the risks associated with customer products and applications, customers should provide adequate design, testing and operating safeguards.

Any access to and/or use of TI software described in this document is subject to Customers entering into formal license agreements and payment of associated license fees. TI software may solely be used and/or copied subject to and strictly in accordance with all the terms of such license agreements.

Customer acknowledges and agrees that TI products and/or software may be based on or implement industry recognized standards and that certain third parties may claim intellectual property rights therein. The supply of products and/or the licensing of software does not convey a license from TI to any third party intellectual property rights and TI expressly disclaims liability for infringement of third party intellectual property rights.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products, software or services are used.

Information published by TI regarding third–party products, software or services does not constitute a license from TI to use such products, software or services or a warranty, endorsement thereof or statement regarding their availability. Use of such information, products, software or services may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Change History

Date	Changed by	Approved by	Version	Status	Notes
1999-Aug-19	LM et al.		0.1		1
1999-Nov-1	MS et al.		0.2	HJS	2
1999-Dec-13	TSE		0.3		3
2003-May-13	XINTEGRA		0.4	Draft	4

Notes:

1. Initial/Submitted
2. Accepted
3. restructed

Table of Contents

List of Figures and Tables

List of References

[ISO 9000:2000] International Organization for Standardization. Quality management systems - Fundamentals and vocabulary. December 2000

1 Introduction

This document describes the functional interface of the GPF Socket driver, API. This driver can be used for common communication purposes.

The driver can either be configured as a server or as a client. The server configuration accepts only a single connection. Therefore only single point to point connections can be established. This is to keep the behavior of the driver similar to the behavior of serial port driver (USART).

The driver does not support Write Signals.

This API of the driver is derived from the generic driver interface specification [C_8415.0026].

NOTE: The driver needs to be configured after initialization. Only the following functions can be called while the driver is not configured: socket_Clear, socket_SetSignal and socket_GetSignal. All other functions return DRV_NOTCONFIGURED.

2 Interface description of the SOCKET driver

2.1 Data types

Name	Description
USHORT	unsigned 16 bit integer data type
SHORT	signed 16 bit integer data type
T_SOCKET_DCB	Device Control Block

2.1.1 T_SOCKET_DCB – Device Control Block

Definition:

```
typedef struct
{
  USHORT   port;
  USHORT   tx_buffer_size ;
  USHORT   rx_buffer_size ;
  ULONG    tx_timeout_msec ;
  ULONG    rx_timeout_msec ;
  char     hostname[SOCKET_MAX_LEN_HOSTNAME+1] ;
 } T_SOCKET_DCB;
```

Description:

The device control block data type contains all parameters used to configure the serial device. The following table contains a list of the data elements and brief descriptions of them.

Data element	Description
port	TCPIP port
tx_buffer_size	Size of the transmission buffer
rx_buffer_size	Size of the reception buffer
tx_timeout_msec	transmission timeout
rx_timeout_msec	reception timeout
hostname	Name of the host the server recedes on. Set the hostname to 0 in order to configure the driver to act as a server

2.2 Constants

Name	Description
SOCKET_MAX_LEN_HOSTNAME	Maximal length of the hostname
DRV_SIGTYPE_CONNECT	Used to specify connection established event signaling
DRV_SIGTYPE_DISCONNECT	Used to specify connection release event signaling
DRV_SIGTYPE_READ	Used to specify data reception event signaling
SOCKET_ERRUNSPEC	Return value specifying a general socket driver failure
SOCKET_NOCONNECT	Return value indicating that no connection exists to a peer entity

2.3 Signals

Signals are used to inform the using process about selected events asynchronously. Signaling is done by passing a signal call-back function to the driver at the time of initialization (see "2.4.1 socket_Init – Driver Initialization"). When no call-back is defined event signaling cannot be performed. A signal can be set using the function *drv*_SetSignal() which can be found in 2.4.7. Event signaling can be disabled by calling the function *drv*_ResetSignal(), for more details on this function refer to Chapter 2.4.8.

The following chapters describe the contents of the T_DRV_SIGNAL information structures defined for the socket driver as an extension to the common driver signals, refer to C_8415.0026.

2.3.1 SOCKET_SIGTYPE_CONNECT

This signal is indicated if the driver is setup as a server and a client has established a connection to the server. From now on the server application can send data to the client (peer entity). A prerequisite to being informed asynchronously about this event is that the signal has been set using the socket_SetSignal() function. The event will only be signaled each time new connection is established..

Paramter	Value
SignalType	SOCKET_SIGTYPE_CONNECT
DataLength	Not used
UserData	not used

2.3.2 SOCKET_SIGTYPE_RELEASE

This signal will be indicated when a connection to the peer entity is lost. Reasons for this can be that the server was shut down (socket_Close()) or the client has disconnected (socket_Close()). Calls to the socket_Write() and socket_Read() function will fail because no connection is available.

Paramter	Value
SignalType	SOCKET_SIGTYPE_RELEASE
DataLength	not used
UserData	not used

2.4 Functions

Name	Description
socket_Init	Initialization of the standard USART driver
socket_Exit	De-initialization of standard USART driver
socket_Open	Open a connection to a server
socket_Close	Close a connection
socket_Write	Send data to the connected client
socket_Read	Read data received from the remote connection
socket_SetSignal	Define a signal used to indicated special events
socket_ResetSignal	Un-define a signal the driver uses to indicate an event
socket_SetConfig	Set a device configuration
socket_GetConfig	Retrieve a device configuration

2.4.1 socket_Init – Driver Initialization

Definition:

```
USHORT socket_Init
(
    USHORT              DrvHandle
    T_DRV_CB_FUNC       CallbackFunc
    T_DRV_EXPORT **     DrvInfo
) ;
```

Parameters:

Name	Description
DrvHandle	unique handle for this driver
CallbackFunc	This parameter points to the function that is called at the time an event that is to be signaled occurs. This value can be set to NULL if event signaling should not be possible.
DrvInfo	pointer to the driver parameters (see GDI specification document for a description of T_DRV_EXPORT).

Return values:

Name	Description
DRV_OK	Initialization successful
DRV_INITIALIZED	Driver already initialized
DRV_INITFAILURE	Initialization failed

Description

The function initializes the internal data of the driver. The function returns DRV_INITIALIZED if the driver has already been initialized and is ready to be used or is already in use. In the case of an initialization failure, i.e. the driver cannot be used, the function returns DRV_INITFAILURE.

The driver exports its properties like its name and the functions to access the driver.

The driver stores the *DrvHandle* and passes it over the SignalID to the calling process every time the callback function is called.

2.4.2 socket_Exit – Termination of the driver

Definition:

```
void socket_Exit
(
 void) ;
```

Parameters:

Name	Description
-	

Return values:

Name	Description
-	-

Description

The function is called when the driver functionality is no longer required. The function “de-allocates” the resources (interrupts, buffers, etc.). The driver terminates regardless of any outstanding data to be sent.

2.4.3 socket_Open - Open a connection

Definition:

USHORT socket_Open(void) ;

Parameters:

Name	Description
-	

Return values:

Name	Description
DRV_OK	Function successful
DRV_INPROCESS	The driver is currently reading data. The data is incomplete.
DRV_NOTCONFIGURED	The driver is not yet configured
SOCKET_ERRUNSPEC	An unspecified error occurred

Description

This function is used to establish a connection to server or activate the driver to act as a server, using the settings of the socket_DCB. A hostname must be specified to open a connection to a server, in this case the driver runs in the client mode. If no hostname is specified the driver will run in server mode. In the server mode it serves a single client. As soon as a client is connected the DRV_SIGTYPE_CONNECT signal is generated, if it was enabled by calling socket_SetSignal().

In case of a successful completion the driver is able to transmit data to (socket_write()) and receive (socket_read()) data from the peer entity. To get notified about the reception of data the appropriate signal DRV_SIGTYPE_READ has to be set (socket_SetSignal()).

In the case of a successful completion, the driver returns DRV_OK.

If the driver is already busy, DRV_INPROCESS is returned.

If the driver is not configured, the function returns DRV_NOTCONFIGURED.

If an error occurs while establishing the requested mode, the function returns SOCKET_ERRUNSPEC.

2.4.4 socket_Close – Close connection

Definition:

void socket_Close(void) ;

Parameters:

Name	Description
-	

Return values:

Name	Description
-	-

Description

This function is used by a client to close the connection or by server to shut down the server functionality.

In case of a successful completion, the connection is shut down and neither socket_Read nor socket_Write will be successful. To get notified about the termination of a connection, the appropriate signal DRV_SIGTYPE_DISCONNECT must be set (socket_SetSignal()).

2.4.5 socket_Read - Read data from the driver

Definition:

```
USHORT socket_Read
(
        void*           Buffer,
        USHORT*         Length
) ;
```

Parameters:

Name	Description
Buffer	This parameter points to the buffer wherein the data is to be copied
Length	On call: number of characters to read. If the function returns DRV_OK, it contains the number of characters read. If the function returns DRV_INPROCESS, it contains 0.

Return values:

Name	Description
DRV_OK	Function successful
DRV_INPROCESS	The driver is currently reading data. The data is incomplete.
DRV_NOTCONFIGURED	The driver is not yet configured
SOCKET_NOCONNECT	Connection not available

Description

This function is used to read data from the socket driver. The data is copied into the buffer to which *Buffer* points. The parameter **Length* contains the size of the buffer in characters.

If the driver is not configured, the function returns DRV_ NOTCONFIGURED.

If no connection exists the driver returns SOCKET_NOCONNECT.

NOTE: When calling the function with **Length* set to 0, the function will return DRV_OK. The size of the buffer needed to store the available data is stored in the parameter *Length. In this case, the parameter *Buffer* can be set to NULL.

2.4.6 socket_Write – Write data to the driver

Definition:

```
USHORT socket_Write
    (
        void*           Buffer,
        USHORT*         Length
    ) ;
```

Parameters:

Name	Description
Buffer	This parameter points to the buffer that is passed to the driver for further processing
Length	On call: number of characters to write. If the function returns DRV_BUFFER_FULL, it contains the maximum number of characters that can be written. If the function returns DRV_OK, it contains the number of characters written. If the function returns DRV_INPROCESS, it contains 0.

Return values:

Name	Description
DRV_OK	Function successful
DRV_BUFFER_FULL	Not enough space
DRV_INPROCESS	Driver is busy writing data
DRV_NOTCONFIGURED	The driver is not yet configured
SOCKET_NOCONNECT	Connection not available

Description

This function is used to write data to the driver. The parameter **Length* contains the number of characters to write.

In the case of a successful completion, the function returns DRV_OK.

If the data cannot be written because the storage capacity of the driver has been exhausted, the function returns DRV_BUFFER_FULL and the maximum number of characters that can be written in **Length*.

If the driver is currently busy writing data and therefore cannot accept further data to be written, it returns DRV_INPROCESS and sets the parameter **Length* to 0.

If the driver is not configured, the function returns DRV_ NOTCONFIGURED.

If no connection exists the driver returns SOCKET_NOCONNECT.

NOTE: When calling the function with **Length* set to 0, the function will return the number of characters that can be written in the parameter **Length*. In this case, the parameter *Buffer* can be set to NULL.

2.4.7 socket_SetSignal – Setup a Signal

Definition:

```
USHORT socket_SetSignal
(
        USHORT                  SignalType
) ;
```

Parameters:

Name	Description
SignalType	Signal type to be set

Return values:

Name	Description
DRV_OK	Function completed successfully
DRV_INVALID_PARAMS	One or more parameters are out of range or invalid
DRV_SIGFCT_NOTAVAILABLE	Event signaling functionality is not available

Description

This function is used to define a single signal or multiple signals that is/are indicated to the process when the event identified by *SignalType* occurs.

To remove a signal, call the function socket_ResetSignal(), refer 2.4.8.

If the signal type is not implemented the driver returns DRV_INVALID_PARAMS.

If no signal call-back function has been defined at the time of initialization, the driver returns DRV_SIGFCT_NOTAVAILABLE.

The DRV_SIGTYPE_WRITE signal is not supported.

2.4.8 socket_ResetSignal – Remove a Signal

Definition:

```
U16 socket_ResetSignal
(
    USHORT          SignalType
) ;
```

Parameters:

Name	Description
SignalType	Signal type to be reset

Return values:

Name	Description
DRV_OK	Function completed successfully
DRV_INVALID_PARAMS	One or more parameters are out of range or invalid
DRV_SIGFCT_NOTAVAILABLE	Event signaling functionality is not available

Description

This function is used to remove a previously set single or multiple signals.

If the passed signal type is not implemented the driver returns DRV_INVALID_PARAMS.

If no signal call-back function has been defined at the time of initialization, the driver returns DRV_SIGFCT_NOTAVAILABLE.

This function is normally not called directly by the process but indirectly via the frame function vsi_d_resetsignal(): For dynamic disabling of signals the process calls vsi_d_resetsignal() to enable the frame to store the new signal type mask for the calling process (refer to C_8415.033).

2.4.9 socket_SetConfig – Setup a driver configuration

Definition:

```
USHORT socket_SetConfig
(
    T_SOCKET_DCB*       DCBPtr
) ;
```

Parameters:

Name	Description
DCBPtr	Pointer to the driver control block

Return values:

Name	Description
DRV_OK	Function successfully completed
DRV_INVALID_PARAMS	One or more values are out of range or invalid in that combination

Description

This function is used to configure the driver. The parameters that can be configured are included in the device control block T_SOCKET_DCB. For detailed information about the contents of the device control block, refer to Chapter 2.1.1.

If any value of the configuration is out of range or invalid in combination with any other value of the configuration, the function returns DRV_INVALID_PARAMS.

Call the socket_GetConfig() function to retrieve the driver's configuration.

The driver needs to be configured after initialization. Only the following functions can be called while the driver is not configured: socket_Clear, socket_SetSignal and socket_GetSignal. All other functions return DRV_NOTCONFIGURED.

2.4.10 socket_GetConfig – Retrieve a driver configuration

Definition:

```
USHORT socket_GetConfig
(
    T_SOCKET_DCB *      DCBPtr
);
```

Parameters:

Name	Description
DCBPtr	Pointer to the driver control block

Return values:

Name	Description
DRV_OK	Function successfully completed
DRV_NOTCONFIGURED	The driver is not yet configured

Description

This function is used to retrieve the configuration of the driver. The configuration is returned in the driver control block to which the pointer *DCBPtr* points. For detailed information about the contents of the device control block, refer to Chapter 2.1.1. The configuration can be requested at any one time.

If the driver is not configured, the function returns DRV_NOTCONFIGURED.

Appendices

A. Acronyms

DS-WCDMA	Direct Sequence/Spread Wideband Code Division Multiple Access

B. Glossary

International Mobile Telecommunication 2000 (IMT-2000/ITU-2000)	Formerly referred to as FPLMTS (Future Public Land-Mobile Telephone System), this is the ITU's specification/family of standards for 3G. This initiative provides a global infrastructure through both satellite and terrestrial systems, for fixed and mobile phone users. The family of standards is a framework comprising a mix/blend of systems providing global roaming. <URL: http://www.imt-2000.org/>