Connect Tech Inc.

Xtreme MIO J1708 API Manual

API Software Reference

Connect Tech Inc.
42 Arrow Road, Guelph, ON CANADA N1K 1S6
Tel: 519.836.1291 • **North America:** 800.426.8979 • **Fax:** 519.836.4878
sales@connecttech.com • http:www.connecttech.com

1 Revision History

Revision	Date	Author(s)	Change
0.00	October 23, 2012	HJ	Added content for J1708 API

2 Table of Contents

3 Contents and Audience

This manual describes an API (Application Programming Interface) for interface to a DLL which facilitates operations with the PCI card designed according to the requirements of XTREME MIO. Its content is intended for a technical audience, and is oriented toward providing information for software and/or hardware engineering people.

4 Introduction

Throughout this document there are references to the **XTREME MIO Technical Users Manual**, which describes the operation of the J1708 port.

This DLL is a collection of function calls that ease the application software development effort.

The application will need to include library **prf636.lib** which is in the DLL directory of the package. Header file **prf636dll.h** has all the API declarations which are described in the API Declarations section, below.

It is recommended to first read the hardware manual Section 6 to understand J1708 operation.

5 API Declarations

OpenJ1708

Opens a J1708 port returning a handle for use with other J1708 functions.

```
HANDLE _stdcall OpenJ1708(int index );
```

Parameters
index index to device number (1 = first port)

Returns Values
INVALID_HANDLE_VALUE on failure otherwise device handle on success.

Remarks
Non-blocking.

CloseJ1708

Closes a J1708 port previously opened using OpenJ1708 and frees resources.

```
BOOL _stdcall CloseJ1708(HANDLE  handle);
```

Parameters
handle handle to device returned on OpenJ1708.

Returns Values
nonzero on success.

Remarks
Non-blocking.

SetupJ1708

Configures a J1708 port parameters.

```
int _stdcall SetupJ1708(
                        HANDLE devHandle,
                        J1708_CONFIG *j1708Config
                       );
```

Parameters

devHandle	handle to device returned on OpenJ1708
j1708Config	pointer to configuration structure

Returns Values

J1708_STATUS_OK on success, otherwise statuses defined in api return statuses.

Remarks

Non-blocking. Following is the J1708_CONFIG structure (see hardware manual section 7.2)

```
typedef struct _J1708_CONFIG {
        ULONG mask_good_tx;         // Control Register bit 0
        ULONG Tx_Priority;          // TX Priority register, valid ranges 0-7
        ULONG Tx_Problem_Limit;     // TX Problem Limit register, valid ranges 2-254
        ULONG Rx_EOM_Level;         // RX EOM Level register, valid ranges 0-15
        ULONG Rx_Afull_Level;       // RX Afull Level register, valid ranges 1-31
}J1708_CONFIG;
```

ReadJ1708

Reads from a J1708 port.

```
int _stdcall ReadJ1708(
                       HANDLE devHandle,
                       char *buffer,
                       int bufferSize,
                       int *numRead
                      );
```

Parameters

devHandle	handle to device returned on OpenJ1708
buffer	pointer to buffer to read to
bufferSize	size of buffer in bytes
numRead	on return contains the actual number of bytes read

Returns Values

J1708_STATUS_OK on success, otherwise statuses defined in api return statuses.

Remarks

Blocking.

The application should pass a buffer of sufficient size to this API call, so that there is enough space for the read data, otherwise it will return **J1708_STATUS_INSUFFICIENT_BUFFER** and the application will get the number of data bytes available by the **GetReadDataAvailJ1708** API call.

WriteJ1708

Writes to a J1708 port.

```
int _stdcall WriteJ1708(
                        HANDLE devHandle,
                        char *buffer,
```

```
                    int bufferSize
                   );
```

Parameters

devHandle	handle to device returned on OpenJ1708
buffer	pointer to buffer to write
bufferSize	number of bytes in buffer to write

Returns Values
J1708_STATUS_OK on success, otherwise statuses defined in api return statuses.

Remarks
Blocking.

GetReadDataAvailJ1708

Get number of bytes available to read on a J1708 port.

```
int _stdcall GetReadDataAvailJ1708(
                                    HANDLE devHandle,
                                    int *dataAvail
                                    );
```

Parameters

devHandle	handle to device returned on OpenJ1708
dataAvail	on return contains number of bytes ready to read

Returns Values
J1708_STATUS_OK on success, otherwise statuses defined in api return statuses.

Remarks
Non-blocking.

GetReadStatusJ1708

Get read status of a J1708 port.

```
int _stdcall GetReadStatusJ1708(
                                  HANDLE devHandle,
                                  char *buffer,
                                  int bufferSize
                                 );
```

Parameters

devHandle	handle to device returned on OpenJ1708
buffer	pointer to buffer to receive status information status values include BUF_OVERRUN
bufferSize	size in bytes of buffer (must be at least 4)

Note in future this function may be extended to return more information
if necessary, so this is why the buffer is not a pointer to just a ULONG.

Returns Values
J1708_STATUS_OK on success, otherwise statuses defined in api return statuses

Remarks
Non-blocking

GetWriteStatusJ1708

Get write status of a J1708 port.

```
int _stdcall GetWriteStatusJ1708(
                                    HANDLE devHandle,
                                  char *buffer,
                                  int bufferSize
                                 );
```

Parameters

devHandle	handle to device returned on OpenJ1708
buffer	pointer to buffer to receive status information status values include STATUS_J1708_TX_ABORTED and STATUS_J1708_TX_PROB
bufferSize	size in bytes of buffer (must be at least 4)

Note in future this function may be extended to return more information
if necessary, so this is why the buffer is not a pointer to just a ULONG

Returns Values
J1708_STATUS_OK on success, otherwise statuses defined in api return statuses.

Remarks
Non-blocking.

AbortJ1708TX

Abort a transmit in progress on a port.

```
int _stdcall AbortJ1708TX(HANDLE devHandle);
```

Parameters

devHandle	handle to device returned on OpenJ1708

Returns Values
J1708_STATUS_OK on success, otherwise statuses defined in api return statuses.

Remarks
Non-blocking.

Write_USER_TA_J1708

Set USER_TA values on a J1708 port.

```
int _stdcall Write_USER_TA_J1708(
                                    HANDLE devHandle,
                                    USER_TA_DLL *userTA
                                   );
```

Parameters

devHandle	handle to device returned on OpenJ1708
userTA	pointer to USER_TA values to set

Returns Values

J1708_STATUS_OK on success, otherwise statuses defined in api return statuses.

Remarks

Non-blocking. Following is the USER_TA_DLL structure (see hardware manual section 7.2.9)

```
typedef struct _USER_TA_DLL {
        UCHAR Value;        // User-Ta value, valid ranges (0-31)
        UCHAR In_Effect;    // User Ta in effect, valid ranges (0-1)
        UCHAR Ignore        // Ignore Ta in effect valid ranges (0-1)
}USER_TA_DLL;
```

6 API Return Values

The following definition is from driver to application via DLL.

```
//
// receive buffer overrun in the driver
//
#define BUF_OVERRUN                    0xb0000001

// GetWriteStatusJ1708 values. For more detail refer to hardware manual section 7.1.2.1
//
// this indicates that there is a TX abort error on J1708
//
#define STATUS_J1708_TX_ABORTED        0xb0000040
//
// this indicates that there is a TX problem error on J1708
//
#define STATUS_J1708_TX_PROB           0xb0000080
```

The following definition is from dll to application

```
#define J1708_STATUS_OK                                0x0000
#define J1708_STATUS_INVALIDPARAM                      0x0001
#define J1708_STATUS_IOFAILURE                         0x0002
#define J1708_STATUS_NOT_INIT                          0x0003
#define J1708_STATUS_INSUFFICIENT_BUFFER               0x0004
```

7 J1708 DLL API Operation

7.1 Opening J1708 Port

The application will use **OpenJ1708** to open J1708 Port, according to the following sample code.

```
HANDLE hDriver = OpenJ1708( unit );
```

Where unit is board index starting from 1, enumerated as J1708_PORT1, J1708_PORT2.

7.2 Set Up J1708 Port

Please read hardware manual section 7.1 to understand the operation of J1708. The application will need to setup J1708 port using API **SetupJ1708**, no read or write operation is permitted until setup is done. This API takes **J1708_CONFIG** structure as an argument. The following code will setup the port.

```
J1708_CONFIG j1708Config = {0};
// see section 7.2 of hardware manual for details of parameters
j1708Config.mask_good_tx        = 0;
j1708Config.Tx_Priority         = 7;
j1708Config.Tx_Problem_Limit    = 111;
j1708Config.Rx_Afull_Level      = 11;
j1708Config.Rx_EOM_Level        = 11;
SetupJ1708( hDriver, &j1708Config);
```

Then the application can read and write to the J1708 port.

7.3 Read from J1708 Port

The application will use **ReadJ1708** API to read data from J1708 port. The following code shows usage of this API call.

```
charbuffer[MAX_BUF_SIZE];
ULONG       ret, bufReq;
int         numRead, i;
ULONG       readStatus;

ReadTryAgain:
ret = ReadJ1708(hDriver, buffer, bufReq, &numRead);
```

If the returned status is ‘J1708_STATUS_OK’ then the application should check the status of the read. The following code shows it

```
ret = GetReadStatusJ1708( hDriver, (char )&readStatus, sizeof readStatus );
if( ret == J1708_STATUS_OK )
{
    if( (readStatus & BUF_OVERRUN) == BUF_OVERRUN )
    {
            printf("J1708 BUF_OVERRUN error\n");
            goto ReadTryAgain;
    }
}
```

If read operation succeeds, then it will get one J1708 complete packet.

The application should pass a buffer of sufficient size to this API call, so that there is enough space for the read data, otherwise it will return **J1708_STATUS_INSUFFICIENT_BUFFER** and the application will get the number of data bytes available by the **GetReadDataAvailJ1708** API call.

If the driver buffer has overrun then the application should find out how much data is available and read all data. The data received is good even though overrun has happened.

```
if (ret == J1708_STATUS_INSUFFICIENT_BUFFER || readStatus & BUF_OVERRUN)
{
	int lastStatus = ret;
	bufReq = 0;
	// buffer was not big enough so find out how much space is needed and try again
	ret = GetReadDataAvailJ1708(pContext->hDriver, &bufReq);
	if( ret == J1708_STATUS_OK )
	{
		// ensure we have enough buffer available to retry with larger size
		if( bufReq > MAX_BUF_SIZE )
		{
			printf("J1708 GetReadDataAvailJ1708 error, buffer required bigger than maximum size\n");
			bufReq = MAX_BUF_SIZE;
		}

		if(lastStatus == J1708_STATUS_INSUFFICIENT_BUFFER)
		{
			goto ReadTryAgain;
		}
	}
	else
		printf("J1708 GetReadDataAvailJ1708 error %d\n", ret);
}
```

7.4 Write to J1708 Port

The application will use **WriteJ1708** API call to write data to J1708 port. The following code shows usage of the API call.

```
charBUF[MAX_BUF_SIZE];
charbuffer;
ULONG		ret;
int		i, bufSize;
ULONG		writeStatus;

ret = WriteJ1708(hDriver, buffer, bufSize);
```

If the returned status is ‘J1708_STATUS_OK’ then the application should check the status of the write. If the write operation status is ‘STATUS_J1708_TX_PROB’ then the application should abort the current transmission and retransmit the last packet. The following code shows it

```
ret = GetWriteStatusJ1708( hDriver, (char )&writeStatus, sizeof writeStatus );
if( ret == J1708_STATUS_OK )
{
	if ((writeStatus & STATUS_J1708_TX_ABORTED) == STATUS_J1708_TX_ABORTED)
	{
		printf("J1708 current packet transmission aborted\n");
	}
	if((writeStatus & STATUS_J1708_TX_PROB) == STATUS_J1708_TX_PROB)
	{
		printf("J1708 Tx problem limit, abort current packet transmission\n");
		 abort the current packet transmission
		if( !AbortJ1708TX( hDriver ) )
		{
			printf("J1708 AbortJ1708TX failed");
		}
	}
}
else
	printf("J1708 GetStatusJ1708 failed error code = %d\n", ret);
```

8 Device Listing

After driver is installed the list of devices is available for viewing in device manager (Control Panel -> System -> Device Manager). The following picture shows the listing of J1708 port installed, marked in blue.