The Embedded I/O Company

TEWS TECHNOLOGIES

TDRV002-SW-25

Integrity Device Driver

Multiple Channel Serial Interface

Version 2.0.x

User Manual

Issue 2.0.0

June 2019

TEWS TECHNOLOGIES GmbH
Am Bahnhof 7 25469 Halstenbek, Germany
Phone: +49 (0) 4101 4058 0 Fax: +49 (0) 4101 4058 19
e-mail: info@tews.com www.tews.com

TDRV002-SW-25

Integrity Device Driver

Multiple Channel Serial Interface

Supported Modules:

TPMC37x
TPMC46x
TPMC47x
TXMC37x
TXMC46x
TCP46x
TCP47x

Issue	Description	Date
1.0.0	First Issue	November 23, 2010
2.0.0	Integrity 11.x support added Support of additional serial boards API-functions modified	June 21, 2019

Table of Contents

1 Introduction

The TDRV002-SW-25 Integrity device driver allows the operation of TDRV002 supported boards.

The driver uses a software FIFO for data that is received and for data that should be sent. Both FIFOs have a size of 2048 characters by default.

The TDRV002-SW-25 device driver supports the following features:

- SW-FIFO for transmit and receive
- configuration of the data word (data and stop bits, parity mode)
- setting baud rates (free scalable, no predefined values)
- setting I/O interface (if supported by hardware)
- support of hardware flow control (RTS/CTS) (if supported by hardware)
- support of software flow control (Xon/Xoff)
- support of local loopback mode

The TDRV002-SW-25 supports the modules listed below:

Module	Serial Interfaces	Programmable Interfaces	FIFO-Size (Bytes)	Isolated	Form Factor	Conduction Cooled
TPMC371	8		64		PMC	●
TPMC372	4		64		PMC	●
TPMC375	8	●	64		PMC	●
TPMC376	4	●	64		PMC	●
TPMC377	4	●	64	●	PMC	●
TPMC378	8		64	●	PMC	●
TPMC460	16		64		PMC	
TPMC461	8		64		PMC	
TPMC462	4		64		PMC	
TPMC463	4		64		PMC	
TPMC465	8	●	64		PMC	
TPMC466	4	●	64		PMC	
TPMC467	4	●	64		PMC	
TPMC469	4	●	64	●	PMC	
TPMC470	4	●	64	●	PMC	
TXMC375	8	●	256		XMC	●
TXMC376	4	●	256		XMC	●
TXMC463	4		256		XMC	
TXMC465	8	●	256		XMC	
TXMC466	4	●	256		XMC	

Module	Serial Interfaces	Programmable Interfaces	FIFO-Size (Bytes)	Isolated	Form Factor	Conduction Cooled
TCP460	16		64		cPCI	
TCP461	8		64		cPCI	
TCP462	4		64		cPCI	
TCP463	4		64		cPCI	
TCP465	8	●	64		cPCI	
TCP466	4	●	64		cPCI	
TCP467	4	●	64		cPCI	
TCP468	4		64		cPCI	
TCP469	8	●	64	●	cPCI	
TCP470	4	●	64	●	cPCI	

In this document all supported modules and devices will be called TDRV002. Specials for certain devices will be advised.

To get more information about the features and use of TDRV002 devices, it is recommended to read the manuals listed below.

User Manual of the used Module
Integrity Manuals (e.g. bspguide.pdf)

2 Installation

The following files are located on the distribution media:

Directory path TDRV002-SW-25:

tdrv002.c	TDRV002 device driver source
tdrv002def.h	TDRV002 driver include file
tdrv002.h	TDRV002 include file for driver and application
tdrv002api.c	Application interface, simplifies device access
tdrv002api.h	Include file for API and applications
tdrv002fifo.c	FIFO function source
tdrv002fifo.h	FIFO function include file
examples*.c	Path with example application
TDRV002-SW-25-2.0.0.pdf	PDF copy of this manual
ChangeLog.txt	Release history
Release.txt	Release information

2.1 Driver Installation

Copy the TDRV002 driver files into a desired driver or project path. The driver source files tdrv002.c and tdrv002fifo.c must be included into the kernel project and the BSP paths must be added to the include search paths of the files. Set Options… ➔ Project → Include Directories, then double click and add the new paths:

$(__OS_DIR)/bsp
$(__OS_DIR)/system
$(__OS_DIR)/modules/ghs/bspsrc
$(__OS_DIR)/modules/ghs/bspsrc/support
$(__OS_DIR)/modules/ghs/bspsrc/driver/busspace

Afterwards the project must be rebuilt. The driver will be started automatically after booting the image and the driver will be requested if a matching device is detected in the system.

For further information building a kernel, please refer to Greenhills MULTI and INTEGRITY Documentation.

2.2 TDRV002 Applications

Copy the TDRV002 API files (tdrv002api.c, tdrv002api.h, and tdrv002.h) into a desired application path, and include tdrv002api.c into the application project.

The application source file must include tdrv002api.h. If these steps are done, the TDRV002 API can be used and the devices will be accessible.

3 API Documentation

3.1 tdrv002Open

Name

tdrv002Open() – open a device

Synopsis

```
TDRV002_HANDLE tdrv002Open
(
    char        *name
)
```

Description

Before I/O operations can be performed to a device, a descriptor must be opened with a call to this function.

This function will create and initialize a descriptor for the device. The returned handle must be specified for all other functions accessing the device.

Each channel can currently be opened and used once.

Parameters

name

This parameter specifies the name of the device. Generally the TDRV002 device names look like ‘tdrv002_<major>_<minor>‘, where <major> specifies the module and <minor> specifies the local channel number. <major> and <minor> are both zero based counts.
For example, the name of the third channel of the first board will be ‘tdrv002_0_2’.
If more than one TDRV002 board is used, the order of the board detection and the assigned <major> number is system and BSP dependent.

Example

```
#include “tdrv002api.h”

TDRV002_HANDLE     handle;

/*
** open descriptor for a device
*/
handle = tdrv002Open(“tdrv002_0_0”);
if (handle == NULL)
{
     /* handle open error */
}
```

Returns

A device handle for the device descriptor, or NULL if the function fails.

3.2 tdrv002Close

Name

tdrv002Close() – close a device

Synopsis

```
Error tdrv002Close
(
      TDRV002_HANDLE        handle
)
```

Description

This function closes a previously opened device.

If this function is called, the descriptor for the device will be released and the device is no longer accessible.

Parameters

handle

This value specifies the device handle which identifies the device. The device handle has been previously returned by tdrv002Open (see chapter 3.1 tdrv002Open).

Example

```
#include “tdrv002api.h”

TDRV002_HANDLE     handle;
Error              errVal;

/*
** close the device
*/
errVal = tdrv002Close(handle);
if (errVal != Success)
{
     /* handle close error */
}
```

Returns

Success if device has been closed or Failure if the specified handle has been invalid.

3.3 tdrv002Write

Name

tdrv002Write() – write a buffer to the device

Synopsis

```
int tdrv002Write
(
        TDRV002_HANDLE      handle,
        char                *buffer,
        int                 len
)
```

Description

This function writes a buffer of characters to the device. The content of the specified buffer will be transferred to the device.

Parameters

handle

This value specifies the device handle which identifies the device. The device handle has been previously returned by tdrv002Open (see chapter 3.1 tdrv002Open).

buffer

This argument specifies the start of the output buffer.

len

This value specifies the number of characters that shall be written.

Example

```
#include “tdrv002api.h”

char               *txtBuf = “Hello world!”;
TDRV002_HANDLE     handle;
int                numWritten;

/*
** write a string to the device
*/
numWritten = tdrv002Write(handle, txtBuf, strlen(txtBuf));
if (numWritten < 0)
{
     /* handle write error */
}
else if (numWritten != strlen(txtBuf))
{
     /* not all characters have been written */
}
else
{
     /* write complete */
}
```

Returns

The number of transferred (written) characters, or <0 (negative “Error”) if the write function failed.

3.4 tdrv002Read

Name

tdrv002Read() – read data from the device

Synopsis

```
int tdrv002Read
(
        TDRV002_HANDLE      handle,
        char                *buffer,
        int                 len
)
```

Description

This function reads data from a device. The received characters will be transferred into the specified buffer. The function will return if the buffer is filled, no more data is available at the device, or if the timeout condition occurred. How the timeout condition can be set, is described in chapter 3.11 tdrv002ConfigureTimeout.

Parameters

handle

This value specifies the device handle which identifies the device. The device handle has been previously returned by tdrv002Open (see chapter 3.1 tdrv002Open).

buffer

This argument specifies the start of the input buffer where the received data will be stored.

len

This value specifies the size of the buffer and defines the maximum number of characters that shall be read.

Example

```
#include “tdrv002api.h”

#define   MAXTXTLEN     25

char               txtBuf[MAXTXTLEN];
TDRV002_HANDLE     handle;
int                numRead;

/*
** read data from the device
*/
numRead = tdrv002Read(handle, txtBuf, MAXTXTLEN);
if (numRead < 0)
{
     /* handle read error */
}
else if (numRead == 0)
{
     /* no data read */
}
else
{
     /* read complete */
}
```

Returns

The number of transferred (read) characters, or <0 (negative “Error”) if the read function failed.

3.5 tdrv002Getc

Name

tdrv002Getc() – get the next character from the device

Synopsis

```
char tdrv002Getc
(
      TDRV002_HANDLE        handle
)
```

Description

This function tries to read the next character from the specified device. If a character is available, the function will return this character immediately. If no character is available, the function will wait until a character is received and it will return this character. This function will not issue a timeout.

Parameters

handle

This value specifies the device handle which identifies the device. The device handle has been previously returned by tdrv002Open (see chapter 3.1 tdrv002Open).

Example

```
#include “tdrv002api.h”

char                inChar;
TDRV002_HANDLE      handle;

/*
** get the next character from the device
*/
inChar = tdrv002Getc(handle);
if (inChar == EOF)
{
     /* handle EOF error */
}
```

Returns

The function returns the received character, or EOF if the function has failed.

3.6 tdrv002SetBaud

Name

tdrv002SetBaud() – set baud rate of the device

Synopsis

```
Error tdrv002SetBaud
(
      TDRV002_HANDLE      handle,
      UINT4               newBaud
)
```

Description

This function sets the baud rate for the specified device. The device will be configured to the specified baud rate or to the baud rate that matches best. (Refer to chapter 4.2 Baud Rate Tolerance)

Parameters

handle

This value specifies the device handle which identifies the device. The device handle has been previously returned by tdrv002Open (see chapter 3.1 tdrv002Open).

newBaud

This value specifies the new baud rate.

Example

```
#include “tdrv002api.h”

TDRV002_HANDLE     handle;
Error              errVal;

/*
** set baud rate to 115200 Baud
*/
errVal = tdrv002SetBaud(handle, 115200);
if (errVal != Success)
{
     /* handle error */
}
```

Returns

The function returns Success if baud rate has been set or Failure if the function has failed.

3.7 tdrv002SetDataword

Name

tdrv002SetDataword() – set data bits, stop bit and parity mode for the device

Synopsis

```
Error tdrv002SetDataword
(
    TDRV002_HANDLE      handle,
    UINT1               dataBits,
    SerialStopBitSetting    stopBits,
    SerialParitySetting     parity
)
```

Description

This function sets the number of data bits, the length of the stop bit and the parity mode.

Parameters

handle

This value specifies the device handle which identifies the device. The device handle has been previously returned by tdrv002Open (see chapter 3.1 tdrv002Open).

dataBits

This value specifies the new number of data bits. Allowed values are 5, 6, 7 and 8 data bits.

stopBits

This value specifies the length of the stop bit. The following values are allowed:

Value	Descrition
OneStopBit	The stop bit is set to a length of 1 bit.
OneAndAHalfStopBits	The stop bit is set to a length of 1.5 bits. (This configuration is allowed for 5 data bits only)
TwoStopBits	The stop bit is set to a length of 2 bit. (This configuration is allowed for 6, 7, and 8 data bits only)

parity

This value specifies the parity mode. The following values are allowed:

Value	Descrition
NoParity	No parity will be used.
OddParity	Odd parity will be used
EvenParity	Even parity will be used

Example

```
#include “tdrv002api.h”

TDRV002_HANDLE     handle;
Error              errVal;

/*
** prepare device for configuration with 8 databits, 1 stopbit
** and no parity
*/
errVal = tdrv002SetDataword(handle, 8, OneStopBit, NoParity);
if (errVal != Success)
{
     /* handle error */
}
```

Returns

The function returns Success if all settings were done or Failure if at least one setting failed.

3.8 tdrv002SetFlowControl

Name

tdrv002SetFlowControl() – Configure flow control for the device

Synopsis

```
Error tdrv002SetFlowControl
(
      TDRV002_HANDLE      handle,
      UINT1               newHandshake
)
```

Description

This function configures the flow control mode.

Parameters

handle

This value specifies the device handle which identifies the device. The device handle has been previously returned by tdrv002Open (see chapter 3.1 tdrv002Open).

newHandshake

This value specifies the flow control (handshake) mode. The following values are allowed:

Value	Description
TDRV002_HANDSHAKE_OFF	Flow control off.
TDRV002_HANDSHAKE_XON_XOFF	Use Xon/Xoff flow control.
TDRV002_HANDSHAKE_HARDWARE	Use hardware flow control (RTS/CTS lines).

Example

```
#include “tdrv002api.h”

TDRV002_HANDLE     handle;
Error              errVal;

/*
** configure device using Xon/Xoff flow control
*/
errVal = tdrv002SetFlowControl(handle, TDRV002_HANDSHAKE_XON_XOFF);
if (errVal != Success)
{
     /* handle error */
}
```

Returns

The function returns Success if flow control has been configured or Failure if the function failed.

3.9 tdrv002ConfigureLoopback

Name

tdrv002ConfigureLoopback() – Configure local loopback mode

Synopsis

```
Error tdrv002ConfigureLoopback
(
      TDRV002_HANDLE      handle,
      Boolean             enableLoopback
)
```

Description

This function configures if local (internal) loopback mode is enabled. This feature allows a functional test of the device without an external connection.

Parameters

handle

This value specifies the device handle which identifies the device. The device handle has been previously returned by tdrv002Open (see chapter 3.1 tdrv002Open).

enableLoopback

This value specifies if local loopback will be enabled or disabled. If the value is true, the internal connection will be enabled, otherwise the device will be connected to external I/O interface.

Example

```
#include “tdrv002api.h”

TDRV002_HANDLE     handle;
Error              errVal;

/*
** enable local loopback mode
*/
errVal = tdrv002ConfigureLoopback(handle, true);
if (errVal != Success)
{
     /* handle error */
}
```

Returns

The function returns Success, or Failure if the function failed.

3.10tdrv002SetTrans

Name

tdrv002SetTrans() – Configure programmable transceiver interface

Synopsis

```
Error tdrv002SetTrans
(
        TDRV002_HANDLE      handle,
        UINT4               newTrConf
)
```

Description

This function configures programmable transceiver interfaces. The function will fail for channels which do not offer programmable transceivers.

Parameters

handle

This value specifies the device handle which identifies the device. The device handle has been previously returned by tdrv002Open (see chapter 3.1 tdrv002Open).

newTrConf

This value specifies how the interface shall be configured. The value is a combination of flags, which allows individual settings.

The function of the interface configuration pins can be found in the corresponding hardware User Manual.

The following flags are defined:

Flag	Description
TDRV002_TRANS_RS485_RS232_SEL	RS485/RS232# configuration pin
TDRV002_TRANS_HDPLX_SEL	HDPLX configuration pin
TDRV002_TRANS_RENA_SEL	RENA configuration pin
TDRV002_TRANS_RTERM_SEL	RTERM configuration pin
TDRV002_TRANS_TTERM_SEL	TTERM configuration pin
TDRV002_TRANS_SLEWLIMIT_SEL	SLEWLIMIT configuration pin
TDRV002_TRANS_SHDN_SEL	SHDN configuration pin
TDRV002_AUTO_RS485_ENABLE_SEL	Enable Auto RS485 Operation mode of XR17D15x

For a simpler configuration, the definitions of common configurations can be used. These definitions can be used instead of the combination of flags above. The following configurations are defined:

Configuration	Description
TDRV002_INTF_OFF	Interface disabled
TDRV002_INTF_RS232	RS232
TDRV002_INTF_RS422	RS422 (Multidrop / Full duplex)
TDRV002_INTF_RS485FDM	RS485 (Full duplex master)
TDRV002_INTF_RS485FDS	RS485 (Full duplex slave)
TDRV002_INTF_RS485HD	RS485 (Half duplex)

Example

```
#include “tdrv002api.h”

TDRV002_HANDLE     handle;
Error              errVal;

/*
** 1st: configure RS422
*/
errVal = tdrv002SetTrans(handle, TDRV002_INTF_RS422);
if (errVal != Success)
{
     /* handle error */
}

…

/*
** 2nd: configure RS422 (using flags)
*/
errVal = tdrv002SetTrans(handle, (TDRV002_TRANS_RS485_RS232_SEL |
                                  TDRV002_TRANS_RTERM_SEL) );
if (errVal != Success)
{
     /* handle error */
}
```

Returns

The function returns Success if the interface has been configured or Failure if the function failed.

3.11tdrv002ConfigureTimeout

Name

tdrv002ConfigureTimeout() – Configure timeouts

Synopsis

```
Error tdrv002ConfigureTimeout
(
    TDRV002_HANDLE      handle,
    Boolean             immReturn,
    Boolean             neverTimeout,
    int                 newTimeout
)
```

Description

This function defines the timeout behavior of tdrv002Read (see chapter 3.4 tdrv002Read).

Parameters

handle

This value specifies the device handle which identifies the device. The device handle has been previously returned by tdrv002Open (see chapter 3.1 tdrv002Open).

immReturn

If this value is true, the read operation will return immediately, even if no data is available. The following arguments will be ignored.

neverTimeout

If this value is true, the read operation will wait until at least one character is received. The specified timeout time will be ignored.
(This value will be ignored if immReturn is true.)

newTimeout

This value specifies the time the read function is willing to wait for a character receive before it returns. The timeout is specified in milliseconds.
(This value will be ignored if immReturn or neverTimeout is true.)

Example

```
#include “tdrv002api.h”

TDRV002_HANDLE     handle;
Error              errVal;

/*
** Set timeout to 10 seconds
*/
errVal = tdrv002ConfigureTimeout(handle, false, false, 10000)
if (errVal != Success)
{
     /* handle error */
}

…

/*
** Set timeout for immediate read
*/
errVal = tdrv002ConfigureTimeout(handle, true, false, 0)
if (errVal != Success)
{
     /* handle error */
}
```

Returns

The function returns Success, or Failure if an invalid device is specified.

3.12tdrv002ErrorStatus

Name

tdrv002ErrorStatus() – Get receive error counts and status

Synopsis

```
Error tdrv002ErrorStatus
(
    TDRV002_HANDLE      handle,
    UINT4               *overrunOccurred,
    UINT4               *parityErrorOccurred,
    UINT4               *framingErrorOccurred,
    Boolean             *breakOccurred
)
```

Description

This function returns receive error counts and the break state. After calling this function, the counters and the state will be reset.

Parameters

handle

This value specifies the device handle which identifies the device. The device handle has been previously returned by tdrv002Open (see chapter 3.1 tdrv002Open).

overrunOccurred

The pointer references a buffer where the number of occurred overrun errors will be returned.

parityErrorOccurred

The pointer references a buffer where the number of occurred parity errors will be returned.

framingErrorOccurred

The pointer references a buffer where the number of occurred framing errors will be returned.

breakOccurred

The pointer references a boolean value which announces if a break has been received.

Example

```
#include “tdrv002api.h”

TDRV002_HANDLE     handle;
Error              errVal;
UINT4              overrun;
UINT4              parity;
UINT4              framing;
Boolean            break;


/*
** get and reset error counts
*/
errVal = tdrv002ErrorStatus(handle,
                            &overrun,
                            &parity,
                            &framing,
                            &break);
if (errVal != Success)
{
     /* handle error */
}
else
{
     printf(“Overrun Errors:%d\n”, overrun);
     …
}
```

Returns

The function returns Success, or Failure if the function failed.

3.13tdrv002RxStatus

Name

tdrv002RxStatus() – Returns the number of Data in the receive FIFO

Synopsis

```
Error tdrv002RxStatus
(
      TDRV002_HANDLE      handle,
      UINT4               *charsInFifo
)
```

Description

This function returns number of characters in the receive FIFO.

Parameters

handle

This value specifies the device handle which identifies the device. The device handle has been previously returned by tdrv002Open (see chapter 3.1 tdrv002Open).

charsInFifo

The pointer references a buffer where the number of characters in the FIFO will be returned.

Example

```
#include “tdrv002api.h”

TDRV002_HANDLE     handle;
Error              errVal;
UINT4              availChars;

…
```

```
…

/*
** get number of characters in Rx FIFO
*/
errVal = tdrv002RxStatus(handle, &availChars);
if (errVal != Success)
{
     /* handle error */
}
else
{
     printf(“Avaiable Characters:%d\n”, availChars);
     …
}
```

Returns

The function returns Success, or Failure if the function failed.

4 Appendix

4.1 Software FIFOs

There is a transmit FIFO, where data is stored before it is written to the UART channel. This allows writing data to the device also if the previous characters have not been transferred. The application can continue working and the data will be transmitted asynchronously.

There is a receive FIFO. All incoming characters will be stored in this FIFO and the read function will transfer data from this FIFO to the application. This FIFO prevents data loss. Received data will be stored in the FIFO and can be read by the application later.

4.1.1 Changing Receive and Transmit FIFO Size

The size of the FIFOs is specified in tdrv002fifo.h. To change the number of characters that can be stored in the FIFOs, change the value of the *TDRV002FIFOBUFSIZE* definition. The default FIFO size is 2048.

After changing the FIFO size, the driver must be rebuilt.

4.2 Baud Rate Tolerance

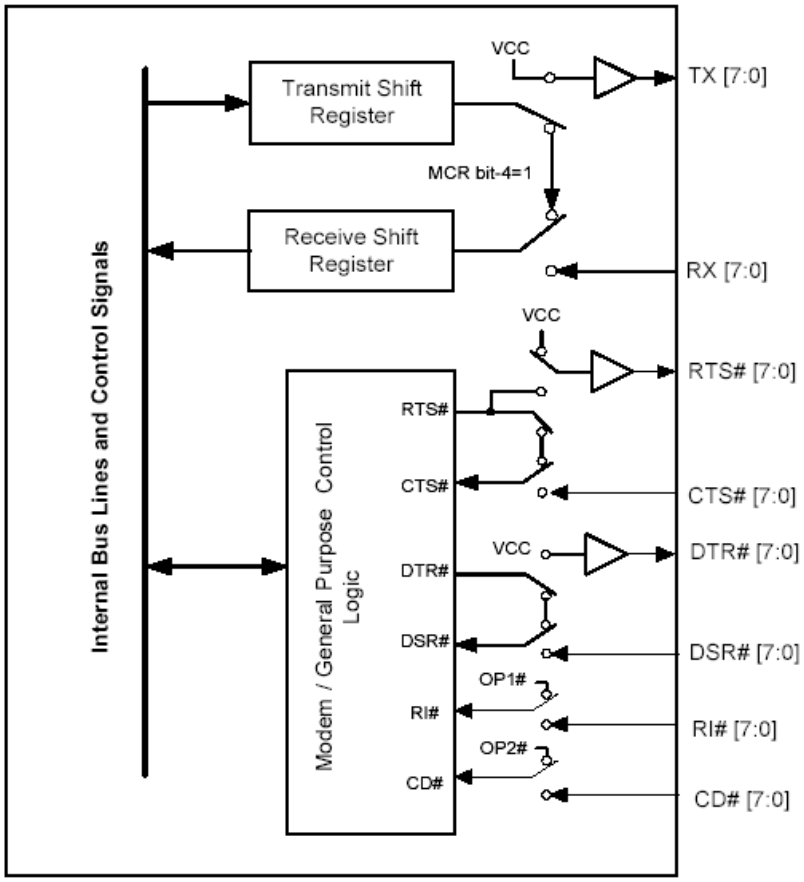
If a specified baud rate is not configurable exactly, the driver calculates the nearest configurable baud rate and checks if the deviation is tolerable. The default setting is 2.5%, which will be tolerable for most UART communications. If a different tolerance is needed or allowed, the value of *MAX_BAUDERROR* can be modified in tdrv002def.h. The value is specified in $^{1}/_{10}$% of the desired value. Deviations below 1% will always be accepted.

After changing the value for baud rate tolerance, the driver must be rebuilt.

4.3 Internal Loopback

The internal loopback mode connects output lines with input lines of the corresponding channel. This allows testing the software and general board access without any external wiring.

If internal loopback is enabled, all I/O lines can be used regardless if they are supported by board I/O or not.



4.4 Example Application

The example application shall give an overview about the use of the TDRV002 devices and how to use the TDRV002 API.

The example application is designed as an interactive console application, so make sure to properly redirect the standard input and standard output for the example application's address space. If using a Dynamic Download Build e.g. in a telnet shell, use the following command:

```
# run -filtered <example_filename> -args <example_address_space>
```