PIXY MVB API

User Manual V1.0

PIXY AG
5300 Turgi
Switzerland

Document number	**11792.000**
Revision	121
Date	5th Mai 2010
Status	Released

Revision History

Version	Revision	Item of change	Author:	Date
0.0	119	Initial version	Haag	26.04.2010
1.0	121	Rework after review	Haag	05.05.2010

Table of contents

List of figures

1 Preface

To shorten Development time and to ease the MVB access, PIXY provides a sophisticated process bus MVB library which is accessed by the SW developer via the PIXY MVB API.

The MVB API has a standard C-Interface. All the field bus specific configuration details is covered by the library and hidden to the user.

The MVB API supports both, MVB ESD+ and MVB EMD. Which version has to be used is defined by the HW.

The MVB API supports process data. It does not handle message data.

The MVB API comes as a package which contains the C-Header file, the API library file either for Linux 2.6.17 or for Windows XPe/WES2009 and the user manual (this document).

2 Abbreviations and Definitions

Abbreviations

API	Application Programming Interface
ESD+	Electrical Short Distance
EMD	Electrical Middle Distance
KB	Kilo Byte.
MVB	Multifunction Vehicle Bus
OGF	Optical Glass Fibre
PIT	Port Index Table
SW	Software
TM	Traffic Store of the MVB

3 General Description

3.1 PIXY MVB API Library States

Basically the library 'lives' in two states the main application can work with. A configuration state and an operational state. It’s possible to switch back and force between the two states in order to modify, extend or reduce the number of logical ports the application SW needs to work on.

When switching back to state “Stop”, the MVB device is closed, the MVB controller is stopped, all memory allocated by the API is de-allocated.

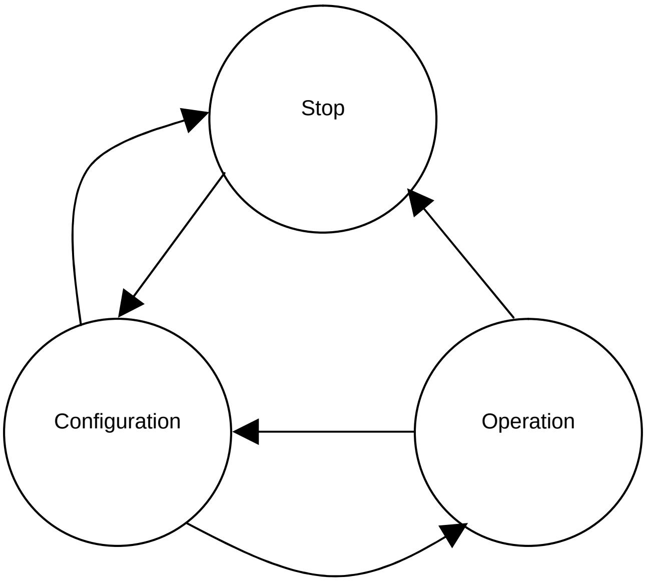

Figure 1 PIXY MVB Library State Diagram

The library maintains an internal data structure for the port configuration. Once the application is switching to the operational state, the traffic store is 'built' using to the port information found in this configuration data structure.

This configuration data structure is maintained and 'lives' as long as the main application does not 'close' the driver.

3.2 MVB API usage

Before the MVB bus can be accessed by any application, it has to be initialized and configured. The PIXY MVB API provides sophisticated functions to do these tasks. Once these two steps are done, the user can operate the MVB bus. Figure 1 shows the different steps and function calls needed to initialize, configure and run the MVB.

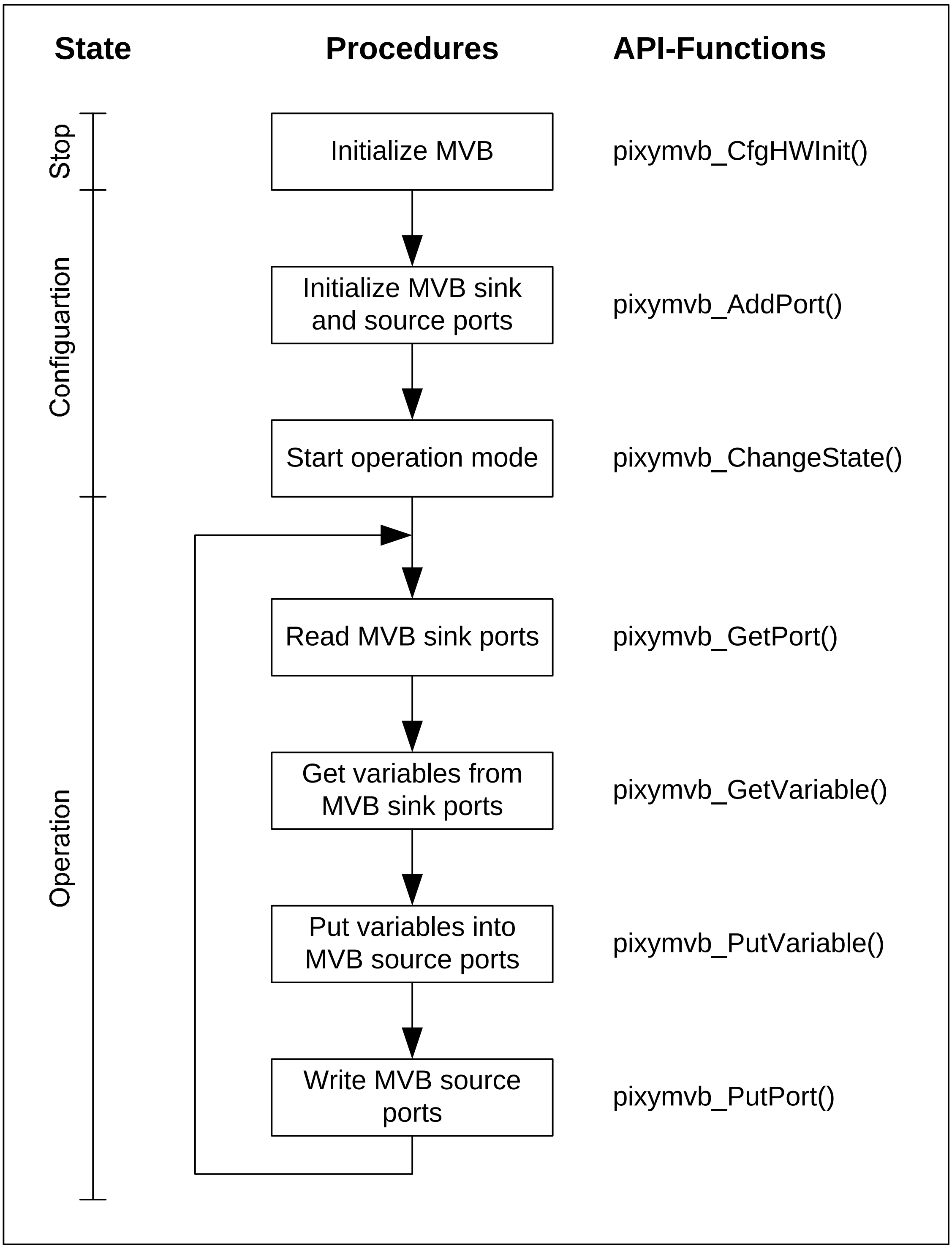

Figure 2 MVB API Application model

4 API Methods

4.1 Stop State Methods

4.1.1 Method pixymvb_CfgHWInit

Declaration:	**`SIGNED16 pixymvb_CfgHWInit( UNSIGNED8 tMModel,`**
	`UNSIGNED8 phyMode,`
	`UNSIGNED16 deviceID,`
	`UNSIGNED8 snkTmeSupvIntv );`	

State:	Stop	
Description:	Initialize the mandatory ISA Bus IO Registers and map the physical memory from the MVB traffic store into a user space. This call opens the MVB device. The device descriptor is being owned by the API. A macro is being used to perform this operation. Portability. Put the MVBIP in the configuration mode.	
Exceptions:	Returns an error if the register access fails or opening of the driver “pixymvbip” fails.	
Input:	tMModel	PIXYMVB_MIN_TMMODEL or PIXYMVB_MAX_TMMODEL PIXYMVB_MIN_TMMODEL equals to 64KB of MVB Traffic Store, the default. PIXYMVB_MAX_TMMODEL is equal to 256KB of MVB Traffic Store.
	phyMode	PIXYMVB_PHY_OFG is an OFG Interface Opto Port PIXYMVB_PHY_ESD is an ESD(+) Interface PIXYMVB_PHY_EMD is an EMD Interface
	deviceID	Each device needs a unique device ID in order to be recognized as a node participating on the MVB network. PIXYMVB_MIN_DEVID <= DeviceID <= PIXYMVB_MAX_DEVID
	snkTmeSupvIntv	Specify if sink time supervision is enabled and if so what update interval shall be chosen. Supported Supervision Interval: 0 = Inactive, 1 = 1 ms, 2 = 2 ms, 3 = 4 ms, 4 = 8 ms, 5 = 16 ms, 6 = 32 ms, 7 = 64 ms, 8 = 128 ms, 9 = 256 ms
Output:	Return Value	PIXYMVB_OK, PIXYMVB_FAILED, PIXYMVB_PARAMERR, PIXYMVB_STATEERR

4.2 Configuration State Methods

4.2.1 Method pixymvb_CfgHWInit

Declaration:	**`SIGNED16 pixymvb_AddPort( UNSIGNED16 logAddress,`**
	`UNSIGNED8 direction,`
	`UNSIGNED8 wordSize,`
	`PIXYMVB_Data *initVal );`	

State:	Configuration	
Description:	Add one port with the logical network address “LogAddress” to the list of ports the traffic store will be configured for. It will not operate on the contents of the traffic store itself. It will modify an internal configuration structure only. **If the initVal argument is a NULL pointer, all data are initialized with zero.**	
Exceptions:	Returns an error if parameters are out of range, or if no more ports are available for configuration. Do not allow that an already defined port is redefined (overwritten). To change a port it must be deleted first.	
Input:	logAddress	1 <= logAddress <= PIXYMVB_MAX_LOGADDRESS
	direction	PIXYMVB_SNKPORT Port is a sink receiving data from the MVB network PIXYMVB_SRCPORT Port is source sending data over the MVB network
	wordSize	1 <= WordSize <= PIXYMVB_MAX_WORDSIZE Size of the port in number of words must be 2^0, 2^1, 2^2, 2^3 or 2^4!
	initVal	A reference to the location being used for data exchange. Must be of type PIXYMVB_Data!
Output:	Return Value	PIXYMVB_OK, PIXYMVB_FAILED, PIXYMVB_PARAMERR, PIXYMVB_STATEERR

4.2.2 Method pixymvb_DelPort

Declaration:	`SIGNED16 pixymvb_DelPort( UNSIGNED16 logAddress );`	
State:	Configuration	
Description:	Delete a port with the logical network address “LogAddress”	
Exceptions:	Returns an error if parameters are out of range. An error is returned, if the port to delete does not exist.	
Input:	logAddress	1 <= LogAddress <= PIXYMVB_MAX_LOGADDRESS
Output:	Return Value	PIXYMVB_OK, PIXYMVB_FAILED, PIXYMVB_PARAMERR, PIXYMVB_STATEERR

4.3 Operation State Methods

4.3.1 Method pixymvb_GetPort

Declaration:	`SIGNED16 pixymvb_GetPort( UNSIGNED16 logAddress,`
	`PIXYMVB_Data *portData,`
	`UNSIGNED16 *snkTmeSupv );`	

State:	Operation	
Description:	Get the data contents of the sink port identified with “logAddress”. The value of the sink time supervision is updated as well.	
Exceptions:	Returns an error if parameters are out of range, or if the addressed port does not allow for unambiguous data retrieval from the traffic memory. For example data retrieval for a port that has not been configured can not be read.	
Input:	logAddress	1 <= LogAddress <= PIXYMVB_MAX_LOGADDRESS
	portData	A reference to the location the data will be stored.
	snkTimeSupv	A reference to the location the sink time supervision value will be stored.
Output:	Return Value	PIXYMVB_OK, PIXYMVB_FAILED, PIXYMVB_PARAMERR, PIXYMVB_STATEERR

4.3.2 Method pixymvb_PutPort

Declaration:	`SIGNED16 pixymvb_PutPort( UNSIGNED16 logAddress,`
	`PIXYMVB_Data *portData );`	

State:	Operation	
Description:	Update the source port with the data contents stored in portData.	
Exceptions:	Returns an error if parameters are out of range, or if the addressed port does not allow for an unambiguous data write operation to the traffic memory. For example data write operation for a port that has not been configured to be writeable can not be executed.	
Input:	logAddress	1 <= LogAddress <= PIXYMVB_MAX_LOGADDRESS
	portData	A reference to the location the data will be read from.
Output:	Return Value	PIXYMVB_OK, PIXYMVB_FAILED, PIXYMVB_PARAMERR, PIXYMVB_STATEERR

4.3.3 Method pixymvb_GetVariable

Declaration:	`SIGNED16 pixymvb_GetVariable(PIXYMVB_Data *portData,`
	`UNSIGNED8 size,`
	`UNSIGNED8 offset`
	`UNSIGNED16 *variable );`	

State:	Operation	
Description:	Get the variable out of the data type portData. Data are swapped if required (little endian).	
Exceptions:	Returns an error if parameters are out of range. For example is the size and offset is not within the data element of the port being read earlier.	
Input:	portData	A reference to the location the data is stored.
	size	The size of the variable in number of bits. Size is restricted to values : 1, 2, 4, 8 and 16!
	offset	An offset in number of bits the variable is being embedded in the data. For variables with size 1 to 8 bit: Offset is restricted to multiples of size! For variables with size equal 16 bit, the offset is restricted to multiples of 8!
	variable	A reference of the variable that shall be updated.
Output:	Return Value	PIXYMVB_OK, PIXYMVB_FAILED, PIXYMVB_PARAMERR, PIXYMVB_STATEERR

4.3.4 Method pixymvb_PutVariable

Declaration:	**`SIGNED16 pixymvb_PutVariable(PIXYMVB_Data *portData,`**
	`UNSIGNED8 size,`
	`UNSIGNED8 offset`
	`UNSIGNED16 *variable );`	

State:	Operation	
Description:	Put the variable in the data type portData. Data are swapped if required (little endian). No data is written to the traffic memory.	
Exceptions:	Returns an error if parameters are out of range. For example if the size and offset is not within the structure element data of the port being written. Size or range mismatch.	
Input:	portData	A reference to the location the data is stored.
	size	The size of the variable in number of bits.
	offset	An offset in number of bits the variable is being embedded in the data.
	variable	A reference of the variable that will be used to update the data contents of portData.
Output:	Return Value	PIXYMVB_OK, PIXYMVB_FAILED, PIXYMVB_PARAMERR, PIXYMVB_STATEERR

4.4 Miscellaneous Methods

4.4.1 Method pixymvb_ChangeState

Declaration:	`SIGNED16 pixymvb_ChangeState( UNSIGNED16 libState );`	
State:	Configuration, Operation	
Description:	Modify the operational state within the library in order to enable / disable different methods the application can use. **State Transition Configuration → Operation** If the library state diagram is moving from "Configuration" towards "Operation" the configuration structure is being parsed and a dedicated traffic memory configuration is being made. Data of the ports being created are initialized with their init value defined by addPort during configuration. At the end of this operation the MVB network communication is enabled. **State Transition Operation → Configuration** If the library state diagram is moving from "Operation" towards "Configuration" the device will stop participating in the process data communication over the MVB network. During configuration no source data from this device are being sent over the network, and no sink data are being received from the network. However the configuration structure remains intact, and minor changes can be made, for example add one additional logical address to the number of process data ports being used by the application. Entering the state "Operation" will rebuild the custom traffic memory configuration again. At the end of this build process the network operation is enabled and data will be transferred over the MVB again. **State Transition Operation → Stop** **State Transition Configuration → Stop** When switching to the state "Stop", the MVB device is closed, the MVB controller is stopped and all memory allocated by this library is de-allocated.	
Exceptions:	Returns an error if parameters are out of range, or if the state change is not supported.	
Input:	libState	"Configuration" or "Operation" or "Stop"
Output:	Return Value	PIXYMVB_OK, PIXYMVB_PARAMERR, PIXYMVB_STATEERR

4.4.2 Method pixymvb_GetState

Declaration:	**`SIGNED16 pixymvb_GetState( UNSIGNED16 *libState );`**	
State:	Stop, Configuration, Operation	
Description:	Get the current state of the library.	
Exceptions:	--	
Input:	LibState	A reference of the library state variable to update. The following values are supported: • PIXYMVB_STOP_STATE • PIXYMVB_CONFIG_STATE • PIXYMVB_OPERATION_STATE
Output:	Return Value	PIXYMVB_OK

4.4.3 Method pixymvb_GetLibVersion

Declaration:	**`SIGNED16 pixymvb_GetLibVersion( UNSIGNED32 *major,`**
	`UNSIGNED32 *minor );`	

State:	Stop, Configuration, Operation	
Description:	Get the version information of the library “PixyMVBLib”	
Exceptions:	--	
Input:	Major	A reference of major version index to be updated
	Minor	A reference of minor version index to be updated
Output:	Return Value	PIXYMVB_OK, PIXYMVB_PARAMERR

5 Exposed Data Structures & Design Constants

5.1 Typedef PIXYMVB_Data

The following typedef defines a data type that is being used to access the traffic store port data in operational mode. Ports being configured for shorter data length may ignore the remaining memory of the array type “PIXYMVB_Data”.

```
typedef UNSIGNED16 PIXYMVB_ Data[16];
```

5.2 Pre-processor Directives/Constants

Preprocessor Directives	Description
PIXYMVB_MIN_DEVID	Each node or device participating in the MVB network communication must be known and visible by its device number/ID. The lowest possible official device/ID is 1.
PIXYMVB_MAX_DEVID	See above. The highest value for the device ID is 4095.
PIXYMVB_OK, PIXYMVB_FAI LED, PIXYMVB_PARAMERR, PIXYMVB_STATEERR	Official method/function return values. Enumeration. PIXYMVB_OK → Ok! PIXYMVB_STATEERR → Not supported in current state PIXYM VB_FAI LED → Unknown Error occurred PIXYMVB_PARAMERR → Parameter check found an error condition.
PIXYMVB_MIN_LOGADDRESS PIXYMVB_MAX_LOGADDRESS	The highest possible network address for a process data frame the address range is from 1... .4095. 0 is reserved as internal “trash address” and shall not be used!
PIXYMVB_MAX_WORDSIZE	The maximum slave frame size of a port is 2^4 words - payload data. Slave frames for cyclic process data communication do have a “payload” length of 2^0, 2^1, 2^2, 2^3 or 2^4 words. Any other setting will result in a parameter error.

Preprocessor Directives	Description
PIXYMVB_STOP_STATE PIXYMVB_CON FIG_STATE PIXYMVB_OPERATION_STATE	PixyMVBLib library states. Enumeration.
PIXYMVB_TMTEST_OK PIXYMVB_TMTEST_REG ERR PIXYMVB_TMTEST_MEMERR	Result codes for the traffic memory test. Enumeration.
PIXYMVB_MIN_TMMODEL	Defines the minimal supported traffic memory layout in this library. Equal 2. Default layout supporting up to 1023 docks.
PIXYMVB_MAX_TMMODEL	Defines the maximal supported traffic memory layout in this library. Equal 3. This layout is supporting up to 4095 docks.
PIXYMVB_MAX_TMSIZE	Max. Traffic Memory Size in Nr of words. It depends on the traffic memory layout being supported by the HW interfaces. For layout TMModel = PIXYMVB_MIN_TMMODEL the traffic memory is 32 Kwords (64KB) For layout TMModel = PIXYMVB_MAX_TMMODEL the traffic memory is 128 Kwords (256KB)
PIXYMVB_SNKPORT	Port is a sink receiving data from the MVB network
PIXYMVB_SRCPORT	Port is source sending data over the MVB network
PIXYMVB_PHY_OFG PIXYMVB_PHY_ESD PIXYMVB_PHY_EMD	Defines the mode the PHY control register (decoder register) will be configured to. PIXYMVB_PHY_OFG is an OFG Interface Opto Port PIXYMVB_PHY_ESD is an ESD(+) Interface PIXYMVB_PHY_EMD is an EMD Interface

6 Example

```
/*
PIXY MVB-API demo program, V1.0, 6.5.2010
=========================

The screen on your linux terminal will change to something
like this :

:-:-:-:-:-:-:-:-:-:-:-:-.-.-._._._._._._._._._._._._._._._._._._._:_:_:-:-:-:-:-:-:-:-:-:-:-:-:-
:-:-:-:-:-:-.-.-.-._._._._._._._._._._._._._._._._._._:_:_:_:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-:-

This may look ugly unless you understand what is going on :

On the source port 0x100 two different values are being written in
On the source port 0x101 two (OTHER) different values are being written in too

The legend is the following:

Source Port 0x100 : "." is equal to receiving 0xAAAA on the sink port 0x208
""                : ":" is equal to receiving 0x5555 on the sink port 0x208

Source Port 0x101 : "-" is equal to receiving 0x9999 on the sink port 0x209
""                : "_" is equal to receiving 0xBBBB on the sink port 0x208

It is important to note that we never see something like "?". This indicates
a value unknown for the Testprogram.

*/


/*
*******************************************************************************
* Function     : main                                                         *
*******************************************************************************
*/

int main(int argc, char* argv[]) {

    UNSIGNED16 MemTestRes, RetVal;

    UNSIGNED16 snkTmeSupv = 0;
    PIXYMVB_Data portData = { 0xFFFF, 0xEEEE, 0xDDDD, 0xCCCC,
                              0xBBBB, 0xAAAA, 0x9999, 0x8888,
                              0x7777, 0x6666, 0x5555, 0x4444,
                              0x3333, 0x2222, 0x1111, 0x0000 };

    PIXYMVB_Data SnkPort208, SnkPort209 = { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 };

    printf("\n\n\n");
    printf("   +++++++++++++++++++++++++++++++++++++++++++++++++\n");
    printf("   + Welcome to the PixyMvbLib confidence test     +\n");

    RetVal = pixymvb_CfgHWInit( PIXYMVB_MIN_TMMODEL, PIXYMVB_PHY_EMD, 1, 5 );

    if ( PIXYMVB_OK == RetVal ) {
        /*                                                             */
        printf("   + INFO : Traffic Store Configured LAYOUT 64KB   +\n");
        printf("   + INFO : Mode EMD, Own Addr is 1, SnkSupv 5 ms  +\n");
    } else {
        printf("   + INFO : Failed to Access Traffic Store !!!     +\n");
        printf("   + INFO : Test Terminated ...                    +\n");
        return(0);
    }

    /* *************************************************************** */
    /* Configure Ports for minimal Data Reflector Application          */

/* Src Port FC 1 0xAAAA */
    pixymvb_AddPort(0x100, PIXYMVB_SRCPORT, 1, (PIXYMVB_Data *) &(portData[5]));
/* Src Port FC 1 0x9999 */
    pixymvb_AddPort(0x101, PIXYMVB_SRCPORT, 1, (PIXYMVB_Data *) &(portData[6]));
    pixymvb_AddPort(0x208, PIXYMVB_SNKPORT, 1, NULL);
    pixymvb_AddPort(0x209, PIXYMVB_SNKPORT, 1, NULL);

    RetVal = pixymvb_ChangeState(PIXYMVB_OPERATION_STATE); /* OP Mode ! */

    printf("   + INFO : CTRL-C to Stop Test Loop ...           +\n");

    for (;;) {

        pixymvb_GetPort(0x208, &SnkPort208, &snkTmeSupv);
        pixymvb_GetPort(0x209, &SnkPort209, &snkTmeSupv);

        if ( SnkPort208[0] != 0 ) {
            if ( portData[5] == SnkPort208[0] ) {
                printf(".");
                pixymvb_PutPort(0x100, (PIXYMVB_Data *) &(portData[10]));
            } else if ( portData[10] == SnkPort208[0] ) {
                printf(":");
                pixymvb_PutPort(0x100, (PIXYMVB_Data *) &(portData[5]));
            } else {
                printf ("\n?");
            }

        } else {
            printf("\n*");
            sleep (1);
        }

        if ( SnkPort209[0] != 0 ) {
            if ( portData[6] == SnkPort209[0] ) {
                printf("-");
                pixymvb_PutPort(0x101, (PIXYMVB_Data *) &(portData[4]));
            } else if ( portData[4] == SnkPort209[0] ) {
                printf("_");
                pixymvb_PutPort(0x101, (PIXYMVB_Data *) &(portData[6]));
            } else {
                printf("\n?");
            }
        } else {
            printf("\n&");
            sleep (1);
        }
    } /* for (;;) ... */
    return(0);
}

void ex_program(int sig) {

    printf("\n\n\n   + INFO : CTRL-C Catched !!! Leaving Test Loop   +\n");
    printf(      "   +++++++++++++++++++++++++++++++++++++++++++++++++\n");
 (void) signal(SIGINT, SIG_DFL);
}
```