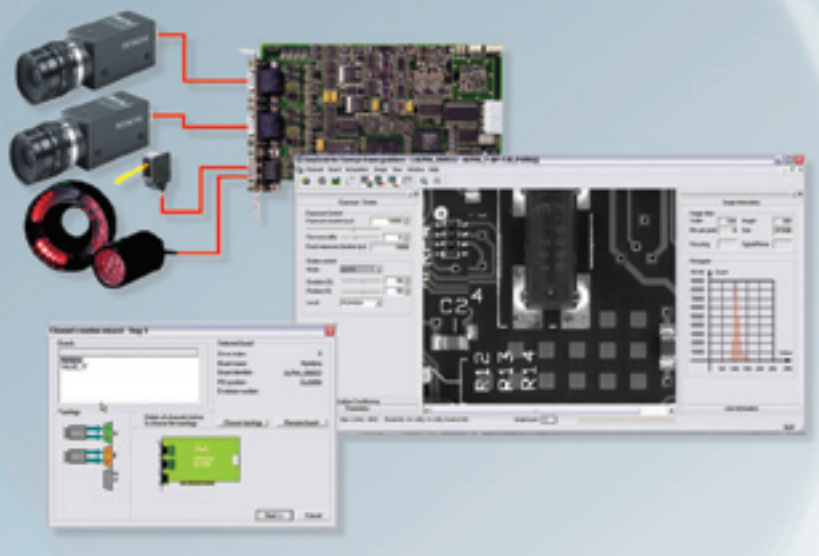
# Application Note

## Migration from MultiCam for PICOLO to MultiCam



**www.euresys.com**
info@euresys.com



EURESYS™
Excellence in vision

# Migration from MultiCam for PICOLO to MultiCam

## Contents

**Purpose of this Application Note** 3
MultiCam for Picolo 3
MultiCam 3
EasyMultiCam 2 3
Error Reporting Differences 3

**Initializing and Terminating the Driver** 4
Initialization 4
Termination 4

**Setting and Getting the Parameters in the Driver** 4
Setting an Integer Type Parameter 4
Getting an Integer Type Parameter 5
Setting an String Type Parameter 6
Getting an String Type Parameter 6

**Board Management** 7
Board Information 7
Checking the Presence of MIO Modules 8
Configuring the I/O Lines 9
Changing the State of I/O Lines 10
Checking and Setting OEM Key 10

**Channel Management** 12
Creating a Channel 12
Deleting a Channel 16
Channel Parameters 16

**Surface Management** 24
Surface Creation 24
Adding Surfaces to the Cluster 24
Surface Deletion 24
Surface Parameter 25

**Signal Management** 26
Signal Handling Differences 26
Registering the Callback Signaling 26
Callback Information 27
Waiting Signals 28

**MultiCam Operation** 29
Starting the Acquisition 29
Stopping the Acquisition 29
Setting the Number of Acquisitions 30
Controlling the Acquisition Rate 30

# Purpose of this Application Note

Traditionally the driver that supports the Picolo series acquisition boards (Picolo, Picolo Pro 2, Picolo Pro 3, Picolo Tetra and Picolo Tetra 2) has been the MultiCam for Picolo driver.

Since 2004, all the acquisition boards from the Picolo series are also supported by the MultiCam driver.

Since 2005, all Euresys frame grabbers and image conditioners can alternatively be controlled through the EasyMultiCam2 Application Programming Interface (API).

This document is intended for Picolo series acquisition board users who wish to port their applications from the MultiCam for Picolo to MultiCam or want to begin a new application using the MultiCam driver.

This document is not meant to be a complete reference for the new user to the Picolo series boards. If you are a first-time Picolo series board user, please refer to "*Euresys Hardware Documentation*" for the understanding of the MultiCam driver concept and MultiCam acquisition principles.

## MultiCam for PICOLO

MultiCam for Picolo is the software layer enabling the operation of Euresys acquisition boards from the Picolo series. The last version of the MultiCam for Picolo is 3.8.3. For ease of reading purposes, all code and parameters relating to the MultiCam for Picolo (MfP) driver are written in green.

## MultiCam

Unified MultiCam or simply MultiCam is the software layer enabling the operation of any Euresys frame grabber or image conditioner. It is a set of API compliant with the standard C language. MultiCam provides an efficient and easy-to-use abstraction layer of the hardware functionality. As such, MultiCam can be considered as the high-level driver of the boards. This driver is unique to all supported Euresys boards and will continue to serve for next generation hardware products complying with the MultiCam specification. For ease of reading purposes, all code and parameters relating to the Unified MultiCam API (UM) driver are written in blue.

## EasyMultiCam 2

EasyMultiCam2 C++ classes are user-friendly tools to gain access to the MultiCam driver functionality. It is a set of object oriented classes complying with the C++ standard that serve as an alternative interface to the native access mode in C language MultiCam API. The concept and operation of EasyMultiCam2 complies with the MultiCam specification. The usage of MultiCam parameters in EasyMultiCam2 is exactly the same as in the MultiCam driver. For ease of reading purposes, all code relating to the EasyMultiCam2 (EMC2) classes is written in pink.

## Error Reporting Differences

MultiCam for Picolo and MultiCam driver's error reporting is error code based.

EasyMultiCam2 error reporting is exception based.

# Initializing and Terminating the Driver

## Initialization

Before using any MultiCam function, the communication between the application process and the MultiCam driver should be established, hence initializing the driver.

In MfP, this is achieved by calling:

ECSTATUS MultiCamSystemInitialize ();

In UM, this is achieved by calling:

MCSTATUS McOpenDriver (NULL);

In EMC2, the initialization of the driver is done automatically. Users do not have to write any code for this purpose.

## Termination

Before terminating the application, the user should terminate the communication of the application process with the MultiCam driver, hence terminating the driver.

In MfP, this is achieved by calling:

ECSTATUS MultiCamSystemTerminate ();

In UM, this is achieved by calling:

MCSTATUS McCloseDriver ();

All channels and surfaces must be deleted before terminating the communication of the driver in UM. After the driver is closed, none of the UM functions will return successfully.

In EMC2, the termination of the driver is done automatically. Users do not have to write any code for this purpose.

# Setting and Getting the Parameters in the Driver

There are two types of parameters in the MfP, one of integer-type and the other of string-type.

Accessing these parameters is easy and requires only one function call.

## Setting an Integer Type Parameter

In MfP, setting an integer MfP parameter is achieved by calling:

MultiCamSystemSetBoardParam (ECHANDLE hBoard, ECPARAMID ParamID, UINT32 Value);

Where hBoard is the identifier of the board.

MultiCamChannelSetParameterInt (ECHANDLE hChannel, ECPARAMID ParamID, INT32 Value);

Where hChannel is the handle to a channel.

In UM, setting an integer MultiCam parameter is achieved by calling one of these methods:

McSetParamInt (MCHANDLE hInstance, MCPARAMID ParamID, INT32 Value);

This function assigns an integer variable to a MultiCam parameter. The parameter is referred to by-ident, and is preferably of the integer or enumerated type.

McSetParamNmInt (MCHANDLE hInstance, PCHAR ParamName, INT32 Value);

This function is an alternative method to assign an integer variable to a MultiCam parameter. The parameter is referred to by-name, and is preferably of the integer or enumerated type.

In both methods above, hInstance is the handle designating the instance owning the parameter to be set.

In EMC2, setting an integer MultiCam parameter belonging to an object is achieved by calling the object member function:

SetParam (MCPARAMID ParamID, INT32 Value);

Or

SetParam (const PCHAR ParamName, INT32 Value);

## Getting an Integer Type Parameter

In MfP, getting an integer MfP parameter is achieved by calling one of these methods:

MultiCamSystemGetBoardParam (ECHANDLE hBoard, ECPARAMID ParamID, PUINT32 pValue );

Where hBoard is the identifier of the board.

MultiCamChannelGetParameterInt (ECHANDLE hBoard, ECPARAMID ParamID, PINT32 pValue);

Where hChannel is the handle to a channel.

MultiCamSurfaceGetParameterInt (ECHANDLE hSurface, ECPARAMID ParamID, PINT32 pValue);

MultiCamSurfaceGetParameterInt64 (ECHANDLE hSurface, ECPARAMID ParamID, PINT64 pValue);

Where hSurface is the handle to a surface.

In UM, getting an integer MultiCam parameter is achieved by calling one of these methods:

McGetParamInt (MCHANDLE hInstance, MCPARAMID ParamID, PINT32 pValue);

This function obtains the current value of a MultiCam parameter as an integer variable. The parameter is referred to by-ident, and is preferably of the integer or enumerated type.

McGetParamNmInt (MCHANDLE hInstance, PCHAR ParamName, PINT32 pValue);

This function is an alternative method to obtain the current value of a MultiCam parameter as an integer variable. The parameter is referred to by-name, and is preferably of the integer or enumerated type.

In both methods above, hInstance is the handle designating the instance owning the parameter to be gotten.

In EMC2, getting an integer MultiCam parameter belonging to an object is achieved by calling the object member function:

GetParam (MCPARAMID ParamID, INT32 &Value);

Or

GetParam (const PCHAR ParamName, INT32 &Value);

## Setting a String Type Parameter

In MfP, setting a string MfP parameter is achieved by calling:

MultiCamChannelSetParameterString (ECHANDLE hChannel, ECPARAMID ParamID, PCCHAR String);

Where hChannel is the handle to a channel.

In UM, setting a string MultiCam parameter is achieved by calling one of these methods:

McSetParamStr (MCHANDLE hInstance, MCPARAMID ParamID, PCHAR String);

This function assigns a string variable to a MultiCam parameter. The parameter is referred to by-ident, and is preferably of the string or enumerated type.

McSetParamNmStr (MCHANDLE hInstance, PCHAR ParamName, PCHAR String);

This function is an alternative method to assign a string variable to a MultiCam parameter. The parameter is referred to by-name, and is preferably of the string or enumerated type.

In both methods above, hInstance is the handle designating the instance owning the parameter to be set.

In EMC2, setting a string MultiCam parameter belonging to an object is achieved by calling the object member function:

SetParam (MCPARAMID ParamID, const PCHAR String);

Or

SetParam (const PCHAR  ParamName, const PCHAR String);

## Getting a String Type Parameter

In MfP, getting a string MfP parameter is achieved by calling:

MultiCamChannelGetParameterString (ECHANDLE hChannel, ECPARAMID ParamID, PCHAR pString,
UINT32 MaxLength);

Where hChannel is the handle to a channel.

In UM, getting a string MultiCam parameter is achieved by calling one of these methods:

McGetParamStr (MCHANDLE hInstance, MCPARAMID ParamID, PCHAR pString, UINT32 MaxLength);

This function obtains the current value of a MultiCam parameter as a string variable. The parameter is referred to by-ident, and is preferably of the string or enumerated type.

McGetParamNmStr (MCHANDLE hInstance, PCHAR ParamName, PCHAR pString, UINT32 MaxLength);

This function is an alternative method to obtain the current value of a MultiCam parameter as a string variable. The parameter is referred to by-name, and is preferably of the string or enumerated type.

In both methods above, hInstance is the handle designating the instance owning the parameter to be gotten.

In EMC2, getting a string MultiCam parameter belonging to an object is achieved by calling the object member function:

GetParam (MCPARAMID ParamID, PCHAR String, INT32 MaxLength);

Or

GetParam (const PCHAR ParamName, PCHAR String, INT32 MaxLength);

# Board Management

The MfP provide a series of functions that can be use to retrieve or set board specific parameters. In UM these functions are consolidated in a Board object class.

## Board Information

The driver provides a way to retrieve the information relative to all the compatible boards detected by the driver. It provides system information about the boards (firmware revision, serial number, name ...) and useful information about the sources available on these boards.

In MfP, this is done by calling:

MultiCamSystemGetBoardInfo (PECBOARDINFO BoardList, UINT32 ListCount, PUINT32 ReturnedCount);

or

MultiCamSystemGetBoardInfoEx (PECBOARDINFOEX BoardList, UINT32ListCount, PUINT32 ReturnedCount);

Where BoardList is the address of a table of ECBOARDINFO structures for which the user provides storage. After the call, this structure will be filled by MultiCam with information about the MultiCam-compatible boards detected in the system. The parameter ListCount is the number of ECBOARDINFO available in the user allocated table. The parameter ReturnedCount is the number of different boards detected by driver.

In UM, this is done by calling:

```
// Defining the database structure type
Typedef struct
{
        Char BoardName[17];
        INT32 SerialNumber;
        INT32 BoardType;
        INT32 PciPosition;
        INT32 ConnectorCount;
} MULTICAM_BOARDINFO ;

// Variable declaration
MULTICAM_BOARDINFO BoardInfo[10];
INT32 BoardCount, i;
//Getting number of boards
McGetParamInt(MC_CONFIGURATION, MC_BoardCount, &BoardCount);

//Scan across all MultiCam boards detected
for (i=0; i<BoardCount; i++)
{
        //Get the board name (String MultiCam parameter)
        McGetParamStr (MC_BOARD+i, MC_BoardName, BoardInfo[i].BoardName, 17);
        //Get the board serial number (Integer MultiCam parameter)
        McGetParamInt (MC_BOARD+i, MC_SerialNumber, &BoardInfo[i].SerialNumber);
        //Get the board type (Enumerated MultiCam parameter)
        McGetParamInt (MC_BOARD+i, MC_BoardType, &BoardInfo[i].BoardType);
        //Get the board PCI position
        McGetParamInt (MC_BOARD+i, MC_PciPosition, &BoardInfo[i].PciPosition);
        //Get the connector count of the board
        McGetParamInt (MC_BOARD+i, MC_ConnectorCount, &BoardInfo[i].ConnectorCount);
}
```

In EMC2, this is done by calling:

```
INT32 BoardCount, i;
char Name[17];
//Getting number of boards, Config is the configuration object
//It is not needed to declare this object in the application
Config.GetParam(MC_BoardCount, &BoardCount);
//Scan across all MultiCam boards detected
for (i=0; i<BoardCount; i++)
{
        //Get the board name (String MultiCam parameter)
        Boards[i].GetParam( MC_BoardName, Name);
…
}
```

## Checking the Presence of MIO Modules

MIO stands for Module Input Output. This I/O module holds four inputs and four outputs. This module can be connected to the Picolo Pro3, Picolo Tetra and Picolo Tetra-X boards. The driver provides a way to check if modules are present or not.

In MfP, this is done by calling:

```
MultiCamSystemGetBoardParam (hBoard, EC_PARAM_ModuleCheck + ModuleID, *result);
```

Where ModuleID represents the module for checking. It can be one of the following values:

MODULE_CHECK_MIO_0: to check the presence of a MIO # 0
MODULE_CHECK_MIO_1: to check the presence of a MIO # 1
MODULE_CHECK_MIO_2: to check the presence of a MIO # 2
MODULE_CHECK_MIO_3: to check the presence of a MIO # 3
MODULE_CHECK_MIO_4: to check the presence of a MIO # 4

The returned value is an integer: 0 if checked module is absent; other value if checked module is present.

In UM, this is done by calling:

```
McGetParamInt (MC_BOARD + BoardID, MC_InputFunction + LineIndex, *Result);
```

Or

```
McGetParamInt (MC_BOARD + BoardID, MC_OutputFunction + LineIndex, *Result);
```

Where BoardID is the driver index given by the driver and LineIndex is the name identifier or index number associated with a given IO line on the module.

For more information on the list of applicable I/O lines index, refer to topic “MultiCam User’s Guide \ Board Object User’s Notes \How_to_work_with_input_output_lines”.

The returned value is an integer: MC_InputFunction_UNKNOWN or MC_OutputFuntion_UNKNOWN if the checked module is present; MC_InputFunction_NONE or MC_OutputFunction_NONE if the checked module is absent.

In EMC2, this is done by calling:

```
Boards[BoardID].GetParam(MC_InputFunction + LineIndex, Result);
```

Or

```
Boards[BoardID].GetParam(MC_OutputFunction + LineIndex, Result);
```

Where BoardID is the driver index given by the driver and LineIndex is the name identifier or index number associated with a given IO line on the module.

The returned value is an integer: MC_InputFunction_UNKNOWN or MC_OutputFuntion_UNKNOWN if the checked module is present; MC_InputFunction_NONE or MC_OutputFunction_NONE if the checked module is absent.

## Configuring the I/O Lines

The driver provides a way to configure the general purpose I/O lines functionality.

### Configuring the I/O Lines Used as Inputs

In MfP, this is done by calling:

MultiCamSystemlSetBoardParam (hBoard, EC_PARAM_InputConfig + LineIndex, INPUT_CONFIG_SOFT);

Where LineIndex is the identifier name or index number associated with a given IO line on the board or module.

In UM, this is done by calling:

McSetParamInt (MC_BOARD + BoardID, MC_InputConfig + LineIndex, InputConfig);

Where BoardID is the driver index of the board and LineIndex is the identifier name or index number associated with a given IO line on the board or module.

InputConfig can be one of these values: MC_InputConfig_SOFT or MC_InputConfig_FREE.

In EMC2, this is done by calling:

Boards[BoardID].SetParam(MC_InputConfig + LineIndex, InputConfig);

Where BoardID is the driver index of the board and LineIndex is the name identifier or index number associated with a given IO line on the board or module.

InputConfig can be one of these values: MC_InputConfig_SOFT or MC_InputConfig_FREE.

### Configuring the I/O Lines Used as Outputs

In MfP, this is done by calling:

MultiCamSystemSetBoardParam (hBoard, EC_PARAM_OutputConfig + LineIndex, OutputConfig);

Where LineIndex is the name identifier or index number associated with a given IO line on the board or module.

OutputConfig can be one of these values: OUTPUT_CONFIG_SOFT, OUTPUT_CONFIG_WATCHDOG or OUTPUT_CONFIG_ALARM.

In UM, this is done by calling:

McSetParamInt (MC_BOARD + BoardID, MC_OutputConfig + LineIndex, OutputConfig);

Where BoardID is the driver index of the board and LineIndex is the identifier name or index number associated with a given IO line on the board or module.

OutputConfig can be one of these values: MC_OutputConfig_SOFT or MC_OutputConfig_FREE.

In EMC2, this is done by calling:

Boards[BoardID].SetParam(MC_OuputConfig + LineIndex, OuputConfig);

Where BoardID is the driver index of the board and LineIndex is the identifier name or index number associated with a given IO line on the board or module.

OuputConfig can be one of these values: MC_OuputConfig_SOFT or MC_OuputConfig_FREE.

## Changing the State of I/O Lines

The driver provides a way to get the logic state of I/O lines used as inputs, and to get or set the logic state of I/O lines used as outputs.

### Reporting the Logic State of an Input I/O Line

In MfP, this is done by calling:

MultiCamSystemGetBoardParam (hBoard, EC_PARAM_InputState, *State);

In UM, this is done by calling:

McGetParamInt (hBoard, MC_InputState, *State);

In EMC2, this is done by calling:

Boards[BoardID].GetParam(MC_InputState, State);

### Reporting the Logic State of an Output I/O Line

In MfP, this is done by calling:

MultiCamSystemGetBoardParam (hBoard, EC_PARAM_OuputState, *State);

In UM, this is done by calling:

McGetParamInt (hBoard, MC_OuputState, *State);

In EMC2, this is done by calling:

Boards[BoardID].GetParam(MC_OuputState, State);

### Issuing the Logic State of an Output I/O Line

In MfP, this is done by calling:

MultiCamSystemSetBoardParam (hBoard, EC_PARAM_OuputState, State);

In UM, this is done by calling:

McSetParamInt (hBoard, MC_OuputState, State);

In EMC2, this is done by calling:

Boards[BoardID].SetParam(MC_OuputState, State);

## Checking and Setting OEM Key

A security feature is incorporated in all MultiCam compliant boards. The general idea is that the OEM application programmer is able to engrave in the board a secret proprietary key. The security key is stored in the non-volatile memory of the board and cannot be read back. However, it is possible to verify that a given board currently holds a security key equal to a given one. Using this simple mechanism it is easy to lock an application to a board or to a set of boards.

### Setting OEM Key

In MfP, to write the OEM key:

MultiCamSystemSetOemKey (BoardID, key);

Where BoardID is the driver index of the board and key is an 8-byte string.

In UM, to write the OEM key:
McSetParamStr (MC_BOARD + BoardID, MC_OemSafetyLock, key);
Where BoardID is the driver index of the board and key is an 8-byte string.

In EMC2, to write the OEM key:
Boards[BoardID].SetParam(MC_OemSafetyLock, key);
Where BoardID is the driver index of the board and key is an 8-byte string.

### Checking OEM Key

In MfP, to check the OEM key:
MultiCamSystemCheckOemKey (BoardID, key);
Where BoardID is the driver index of the board and key is an 8-byte string.
If the user's key matches the OEM key, the return value is EC_OK.

In UM, to check the OEM key:

```
// Variables declaration
char Match[6];
// Entering the reference key number
McSetParamStr (MC_BOARD + BoardID, MC_OemSafetyKey, key);

// Checking for correspondence
McGetParamStr (MC_BOARD + BoardID, MC_OemSafetyLock, &Match, 6);
if (Match==”FALSE”)
{
// Rejection code…
//...
}
```

Where BoardID is the driver index of the board and key is an 8-byte string.

In EMC2, to check the OEM key:

```
// Variables declaration
char Match[6];
// Entering the reference key number
Boards[BoardID].SetParam(MC_OemSafetyKey, key);

// Checking for correspondence
Boards[BoardID].GetParam(MC_OemSafetyLock, Match, 6);
if (Match==”FALSE”)
{
// Rejection code…
//...
}
```

Where BoardID is the driver index of the board and key is an 8-byte string.

# Channel Management

A channel is the link between a video source and its memory buffers in the pc (surfaces grouped in a cluster). It holds all parameters dedicated to the image acquisition of this camera.

## Creating a Channel

To create a channel in the MfP or UM, the Board and Connector information is necessary. To specify the linkage of a board to a channel, four different ways are provided to address a particular board.

### Case 1: Index in a List of Compliant Boards

This method designates a particular board where the channel should be created based on the board location in the list returned by the driver. The list is a set of consecutive integer numbers starting at index zero given to each board that complies with the driver. The indexing order is system dependent. However, for a given host computer and identical insertion of the boards in the PCI slots, the numbering order will be consistently repeated.

In MfP, this method is achieved by calling:

```
MultiCamChannelCreate (NULL, “#XXX”, ConnectorID);
```

Where XXX is the BoardID number in the list given by MfP.

In UM, this method is achieved by calling:

```
McCreate (MC_CHANNEL, &hChannel);
McSetParamInt (hChannel, MC_DriverIndex, XXX);
McSetParamInt (hChannel, MC_Connector, Connector);
```

Where XXX is the DriverIndex number[1] in the list given by UM.

In EMC2, this method is achieved by calling the constructor of the channel object:

```
Channel (Boards[XXX], Connector);
```

See connector conversion matrix (page 15) for details on compatibility of ConnectorID in MfP to Connector in UM.

***Example***

This example shows the way to create a channel on the 1st connector of the acquisition board found with driver index 0 in the system.

In MfP:

```
ECHANDLE hChannel = MultiCamChannelCreate (NULL, “#0”, PICOLO_VID1);
```

In MU:

```
MCHANDLE hChannel;
McCreate (MC_CHANNEL, &hChannel);
McSetParamInt (hChannel, MC_DriverIndex, 0);
McSetParamInt (hChannel, MC_Connector, MC_Connector_VID1);
```

In EMC2:

```
Channel *channel = new Channel (Boards[0], MC_Connector_VID1);
```

[1] The index number may not match from MfP to UM. The list in MfP only consists of acquisition boards from the Picolo series that are supported by the driver. The list in UM consists of all the Euresys frame grabbers that complies with the driver. Compatible characteristics, but different locations. All surfaces belonging to a cluster should be able to accept images coming from the same source through a given channel.

### Case 2: Board Type and Serial Number

This method designates a particular board where the channel should be created using its type and its serial number. The identifier is an ASCII character string resulting from the concatenation of the board type and the serial number with an intervening underscore. The serial number is a 6-digit string made of characters 0 to 9.

In MfP, this method is achieved by calling:

```
MultiCamChannelCreate (NULL, "@XXX", ConnectorID)
```

In UM, this method is achieved by calling:

```
McCreate (MC_CHANNEL, &hChannel);
McSetParamStr (hChannel, MC_BoardIdentifier, "XXX");
McSetParamInt (hChannel, MC_Connector, Connector);
```

In EMC2, this method is achieved by calling the constructor of the channel object:

```
Channel (Boards.GetBoardByBoardIdentifier("XXX"), Connector);
```

See connector conversion matrix (page 15) for details on compatibility of ConnectorID in MfP to Connector in UM.

***Example***

This example shows the way to create a channel on the 2nd connector of the Picolo Pro2 acquisition board with serial number 007293.

In MfP:

```
ECHANDLE hChannel = MultiCamChannelCreate (NULL, "@PICOLO_PRO2_007293", PICOLO_VID2);
```

In MU:

```
MCHANDLE hChannel;
McCreate (MC_CHANNEL, &hChannel);
McSetParamStr (hChannel, MC_BoardIdentifier, "PICOLO_PRO2_007293");
McSetParamInt (hChannel, MC_Connector, MC_Connector_VID2);
```

In EMC2:

```
Board *b1 = Boards.GetBoardByBoardIdentifier ("PICOLO_PRO2_007293");
Channel *channel = new Channel (b1, MC_Connector_VID2);
```

### Case 3: Internal Name

This method designates a particular board where the channel should be created based on the internal name given to the board. The name is a string of maximum 16 ASCII characters. It can be altered by the user if desired, and can be made unique for each board in a system. The name is written in a non-volatile memory residing in the board.

In MfP, this method is achieved by calling:

```
MultiCamChannelCreate (NULL, “&XXX”, ConnectorID)
```

In UM, this method is achieved by calling:

```
McCreate (MC_CHANNEL, &hChannel);
McSetParamStr (hChannel, MC_BoardName, “XXX”);
McSetParamInt (hChannel, MC_Connector, Connector);
```

In EMC2, this method is achieved by calling the constructor of the channel object:

```
Channel (Boards.GetBoardByBoardName(“XXX”), Connector);
```

See connector conversion matrix (page 15) for details on compatibility of ConnectorID in MfP to Connector in UM.

***Example***

This example shows the way to create a channel on the 1st connector of the acquisition board with name called This example shows the way to create a channel on the 1st connector of the acquisition board with name called MyPicoloPro.

In MfP:

```
ECHANDLE hChannel = MultiCamChannelCreate (NULL, “&MyPicoloPro”, PICOLO_VID1);
```

In MU:

```
MCHANDLE hChannel;
McCreate (MC_CHANNEL, &hChannel);
McSetParamStr (hChannel, MC_BoardName, “MyPicoloPro”);
McSetParamInt (hChannel, MC_Connector, MC_Connector_VID1);
```

In EMC2:

```
Board *b1 = Boards.GetBoardByBoardName (“MyPicoloPro”);
Channel *channel = new Channel (b1, MC_Connector_VID1);
```

### Case 4: PCI Position

This method designates a particular board where the channel should be created based on the number associated to the PCI slot. This number is assigned by the operating system in a non-predictable way but remains consistent for a given configuration in a given system.

In MfP, this method is achieved by calling:

```
MultiCamChannelCreate (NULL, “%XXX”, ConnectorID)
```

In UM, this method is achieved by calling:

```
McCreate (MC_CHANNEL, &hChannel);
McSetParamInt (hChannel, MC_PciPosition, XXX);
McSetParamInt (hChannel, MC_Connector, Connector);
```

In EMC2, this method is achieved by calling the constructor of the channel object:

```
Channel (Boards.GetBoardByPciPosition(XXX), Connector);
```

See connector conversion matrix (below) for details on compatibility of ConnectorID in MfP to Connector in UM.

***Example***

This example shows the way to create a channel on the 1st connector of the acquisition board that is on the PCI position 0x10038 in the system.

In MfP:

```
ECHANDLE hChannel = MultiCamChannelCreate (NULL, “%0x10038”, PICOLO_VID1);
```

In MU:

```
MCHANDLE hChannel;
McCreate (MC_CHANNEL, &hChannel);
McSetParamInt (hChannel, MC_PciPosition, 0x10038);
McSetParamInt (hChannel, MC_Connector, MC_Connector_VID1);
```

In EMC2:

```
Board *b1 = Boards.GetBoardByPciPosition (0x10038);
Channel *channel = new Channel (b1, MC_Connector_VID1);
```

**Connector conversion matrix**

| Board | ConnectorID (MfP) | Connector (UM) | Remarks |
|---|---|---|---|
| Picolo | PICOLO_VID(X) | MC_Connector_VID(X) | (X) = 1 to 3 |
| Picolo | PICOLO_YC | MC_Connector_YC | |
| Picolo Pro 2, Junior | PICOLO_VID(X) | MC_Connector_VID(X) | (X) = 1 to 4 |
| Picolo Pro 3, Tetra, Tetra-X | PICOLO_VID(X) | MC_Connector_VID(X) | (X) = 1 to 16 |

## Deleting a Channel

In MfP, this is achieved by calling:
MultiCamChannelDelete (hChannel);
Where hChannel is a handle to the MfP channel.

In UM, this is achieved by calling:
McDelete (hChannel);
Where hChannel is a handle to the MultiCam channel.

In EMC2, the deletion of the channel is done by deleting the channel object in the destructor function.
delete channel;
Where channel is an instance of the MultiCam channel class.

## Channel Parameters

The driver provides a list of parameters relating to the channel functionality and control.
**In the following section, application of the MultiCam parameters is exactly the same in UM and EMC2.**

### Selecting Video Standard

After creating the channel, user have to specify the type of video signal standard.

In MfP, the type of video signal standard is specified by the Standard parameter.
MultiCamChannelSetParameterInt (hChannel, EC_PARAM_Standard, Standard);
Where hChannel is a handle to the MfP channel.

In UM, the type of video signal standard is specified by the Camera parameter.
McSetParamInt(hChannel, MC_Camera, Camera);
Where hChannel is a handle to the MultiCam channel.

Standard conversion matrix:

| Standard (MfP) | Camera (UM) |
|---|---|
| EC_STANDARD_CCIR | MC_Camera_CAMERA_CCIR |
| EC_STANDARD_EIA | MC_Camera_CAMERA_EIA |
| EC_STANDARD_PAL | MC_Camera_CAMERA_PAL |
| EC_STANDARD_NTSC | MC_Camera_CAMERA_NTSC |
| EC_STANDARD_SECAM | Not supported |

### Selecting Color Format

After creating the channel, the user can specify the channel respective acquisition data color format.

In MfP, the acquisition data color format is specified by the AcqColFmt parameter.
MultiCamChannelSetParameterInt (hChannel, EC_PARAM_AcqColFmt, AcqColFmt);
Where hChannel is a handle to the MfP channel.

In UM, the acquisition data color format is specified by ColorFormat.
McSetParamInt (hChannel, MC_ColorFormat, ColorFormat);
Where hChannel is a handle to the MultiCam channel.

Color format conversion matrix:

| AcqColFmt (MfP) | ColorFormat (UM) |
|---|---|
| MC_COLORFORMAT_Y8 | MC_ColorFormat_Y8 |
| MC_COLORFORMAT_RGB32 | MC_ColorFormat_ARGB32 |
| MC_COLORFORMAT_RGB24 | MC_ColorFormat_RGB24 |
| MC_COLORFORMAT_RGB16 | MC_ColorFormat_RGB16 |
| MC_COLORFORMAT_RGB15 | MC_ColorFormat_RGB15 |
| MC_COLORFORMAT_YUV422 | MC_ColorFormat_YUV422 |
| MC_COLORFORMAT_YUV411 | MC_ColorFormat_YUV411 |
| MC_COLORFORMAT_PLANAR_YUV422 | MC_ColorFormat_YUV422PL |
| MC_COLORFORMAT_PLANAR_YUV411 | MC_ColorFormat_YUV411PL |
| MC_COLORFORMAT_PLANAR_YUV420 | MC_ColorFormat_I420 |
| MC_COLORFORMAT_PLANAR_YUV410 | Not supported yet |
| MC_COLORFORMAT_PLANAR_YVU420 | Not supported yet |
| MC_COLORFORMAT_PLANAR_YVU410 | Not supported yet |

### Getting Information from Planar Format

For Planar format, each image data stored in MfP is organized in one MultiCam surface containing three planes in the same memory buffer. In UM, each image data is organized in one MultiCam surface containing three planes, each plane is stored in a separate image buffer.

In UM, to retrieve the number of planes in a surface:
McGetParamInt (hChannel, MC_PlaneCount, &Planes);
The result is 3 planes for planar format, otherwise 1.
To get the individual address of each plane:
McGetParamInt (hChannel, MC_SurfaceAddr, &ImageBuffer0);
McGetParamInt (hChannel, MC_SurfaceAddr+1, &ImageBuffer1);
McGetParamInt (hChannel, MC_SurfaceAddr+2, &ImageBuffer2);

### Selecting Field Mode

The Picolo series acquisition boards are able to select if acquire from one or both fields and in which order.

In MfP, this is achieved by calling:

MultiCamChannelSetParameterInt (hChannel, EC_PARAM_InitGrabSync, GrabSync);

MultiCamChannelSetParameterInt (hChannel, EC_PARAM_RepeatGrabSync, GrabSync);

MultiCamChannelSetParameterInt (hChannel, EC_PARAM_FieldMode, FieldMode);

In UM, this is done by calling:

McSetParamInt(hChannel, MC_GrabField, GrabField);

McSetParamInt(hChannel, MC_NextGrabField, GrabField);

Field mode conversion matrix:

| MfP | | MU |
|---|---|---|
| **Field Mode** | **GrabSync** | **Grabfield** |
| 1 | GRABSYNC_NEXT | MC_GrabField_FLD |
| 1 | GRABSYNC_NEXT_UP | MC_GrabField_UP |
| 1 | GRABSYNC_NEXT_DOWN | MC_GrabField_DOWN |
| 2 | GRABSYNC_NEXT | MC_GrabField_FRAME |
| 2 | GRABSYNC_NEXT_UP | MC_GrabField_UPDW |
| 2 | GRABSYNC_NEXT_DOWN | MC_GrabField_DWUP |

### Setting Gain and Offset

The Picolo series acquisition boards can apply a gain to provide contrast adjustment and offset to provide brightness adjustment.

In MfP, gain adjustment is achieved by calling:

MultiCamChannelSetParameterInt (hChannel, EC_PARAM_VideoGain, Value);

Where Value is a 32-bit fixed point positive value (16:16). Bits [31-16] are the signed integer part; bits [15-0] are the unsigned fractional part.

In UM, gain adjustment is achieved by calling:

McSetParamInt (hChannel, MC_GainCtl, MC_GainCtl_LIN);

McSetParamInt (hChannel, MC_Gain, Value);

Where Value is expressed in 1/1000 and applies when GainCtl is set to LIN.

In MfP, offset adjustment is achieved by calling:

MultiCamChannelSetParameterInt (hChannel, EC_PARAM_VideoOffset, Value);

Where Value is a 32-bit fixed point integer value (16:16). Bits [31-16] are the signed integer part; bits [15-0] are the unsigned fractional part.

In UM, offset adjustment is achieved by calling:

McSetParamInt (hChannel, MC_Offset, Value);

Where Value is expressed in 1/1000.

In MfP, chrominance U gain is achieved by calling:

MultiCamChannelSetParameterInt (hChannel, EC_PARAM_VideoUGain, Value );

Where Value is a 32-bit fixed point positive value (16:16). Bits [31-16] are the signed integer part; bits [15-0] are the unsigned fractional part.

In UM, chrominance U gain is achieved by calling:

McSetParamInt (hChannel, MC_ColorGainTrimU, Value);

Where Value is expressed in 1/1000.

In MfP, chrominance V gain is achieved by calling:

MultiCamChannelSetParameterInt (hChannel, EC_PARAM_VideoVGain, Value);

Where Value is a 32-bit fixed point positive value (16:16). Bits [31-16] are the signed integer part; bits [15-0] are the unsigned fractional part.

In UM, this is achieved by calling:

McSetParamInt (hChannel, MC_ColorGainTrimV, Value);

Where Value is expressed in 1/1000.

In MfP, NTSC Hue adjustment is achieved by calling:

MultiCamChannelSetParameterInt (hChannel, EC_PARAM_VideoNtscHue, Value);

Where Value is a 32-bit fixed point value (16:16). Bits [31-16] are the signed integer part; bits [15-0] are the unsigned fractional part.

In UM, this is achieved by calling:

McSetParamInt (hChannel, MC_ColorGainTrimHue, Value);

Where Value is expressed in degrees.

Other gain and offset control parameters provided in UM include: MC_GainTrim1; MC_OffsetTrim1; MC_ColorGain

### Setting Acquisition Filters

The Picolo series acquisition boards can apply filtering to provide high quality image data.

In MfP, the selection of the color sub-carrier removal filter is done by calling:

MultiCamChannelSetParameterInt (hChannel, EC_PARAM_ColorTrap, ColorTrap);

Where ColorTrap could be one of these values: EC_COLORTRAP_NONE, EC_COLORTRAP_PAL, EC_COLORTRAP_NTSC.

In UM, the selection of the color sub-carrier removal filter is done by calling:

McSetParamInt (hChannel, MC_ColorTrap, ColorTrap);

Where ColorTrap could be one of these values: MC_ColorTrap_ENABLE, MC_ColorTrap_DISABLE.

In MfP, the selection of coring option is achieved by calling:

MultiCamChannelSetParameterInt (hChannel, EC_PARAM_Coring, Coring);

Where Coring could be one of these values: EC_CORING_NONE, EC_CORING_8, EC_CORING_16, EC_CORING_32.

In UM, the selection of coring option is achieved by calling:

McSetParamInt (hChannel, MC_DataCoring, DataCoring);

Where DataCoring could be one of these values: MC_DataCoring_NONE, MC_DataCoring_CORING8, MC_DataCoring_CORING16, MC_DataCoring_CORING32.

In MfP, the selection of removing chrominance from a low color image is achieved by calling:

MultiCamChannelSetParameterInt(hChannel, EC_PARAM_CKILL, Value);

Where Value could be one of these settings: 0, 1.

In UM, the selection of removing chrominance from a low color image is achieved by calling:

McSetParamInt (hChannel, MC_ColorKiller, ColorKiller);

Where ColorKiller could be one of these values: MC_ColorKiller_ENABLE, MC_ColorKiller_DISABLE.

In MfP, the selection of gamma removal is achieved by calling:

MultiCamChannelSetParameterInt(hChannel, EC_PARAM_Gamma, Value);

Where Value could be one of these settings: 0, 1.

In UM, the selection of gamma removal is achieved by calling:

McSetParamInt (hChannel, MC_GammaRemoval, GammaRemoval);

Where GammaRemoval could be one of these values: MC_GammaRemoval_ENABLE, MC_GammaRemoval_DISABLE.

### Triggered Acquisition

The Picolo series acquisition boards support acquisitions initiated by external events (trigger).

In MfP, the behavior of the system triggering is specified by TriggerMode parameter. The channel's trigger line is selected with the character string parameter DigitalIoConfig. The attribute of the trigger line is specified by InitTrigger and RepeatTrigger parameters.

MultiCamChannelSetParameterInt (hChannel, EC_PARAM_TriggerMode, TRIGGERMODE_SYSTEM);
MultiCamChannelSetParameterInt (hChannel, EC_PARAM_InitTrigger, InitTrigger);
MultiCamChannelSetParameterInt (hChannel, EC_PARAM_RepeatTrigger, RepeatTrigger);
MultiCamChannelSetParameterString (hChannel, EC_PARAM_DigitalIoConfig, TriggerPattern);

The TriggerPattern is the character string parameter used to configure the digital I/O lines.

The possible combination of InitTrigger and RepeatTrigger parameters in MfP is listed as below.

| InitTrigger | RepeatTrigger | | | | | |
|---|---|---|---|---|---|---|
| | TRIGGER _PAUSE | TRIGGER _EXT_ HIGH | TRIGGER _EXT_LOW | TRIGGER _EXT_ GOING_HI | TRIGGER _EXT_ GOING_LOW | TRIGGER _EXT_ GOING_ANY |
| TRIGGER _PAUSE | ✓ | | | | | |
| TRIGGER _EXT_ HIGH | ✓ | ✓ | | | | |
| TRIGGER _EXT_LOW | ✓ | | ✓ | | | |
| TRIGGER _EXT_ GOING_HI | ✓ | | | ✓ | | |
| TRIGGER _EXT_ GOING_LOW | ✓ | | | | ✓ | |
| TRIGGER _EXT_ GOING_ANY | ✓ | | | | | ✓ |

In UM, the behavior of the system triggering is specified by the TrigMode and NextTrigMode parameters. The channel's trigger line is specified by the TrigLineIndex parameter. The attribute of the trigger line is specified by the TrigEdge parameter.

McSetParamInt(hChannel, MC_TrigMode, TrigMode);
McSetParamInt(hChannel, MC_NextTrigMode, NextTrigMode);
McSetParamInt (hChannel, MC_TrigLineIndex, TrigLineIndex);
McSetParamInt (hChannel, MC_TrigEdge, TrigEdge);

The TrigMode parameter could be one of these values: MC_TrigMode_COMBINED, MC_TrigMode_HARD, MC_TrigMode_SOFT, MC_TrigMode_IMMEDIATE.

The NextTrigMode parameter could be one of these values: MC_NextTrigMode_REPEAT, MC_NextTrigMode_PERIODIC, MC_NextTrigMode_SAME.

The TrigLineIndex is the hardware line sensed by the channel and aimed at generating the trigger event. For more information on the list of applicable I/O lines index, refer to topic "*MultiCam User's Guide \ Board Object User's Notes \ How_to_work_with_input_output_lines*".

The TrigEdge parameter could be one of these values: MC_TrigEdge_GOLOW, MC_TrigEdge_GOHIGH.

### Channel State

The channel state parameter controls the activity of the channel. Changing the state value will either start or stop the channel activity.

In MfP:

MultiCamChannelSetParameterInt (hChannel, EC_PARAM_ChannelState, ChannelState);

Where ChannelState could be one of these values: CHANNEL_STATE_ACTIVE, CHANNEL_STATE_IDLE.

Using MU:

McSetParamInt(hChannel, MC_ChannelState, ChannelState);

Where ChannelState could be one of these values: MC_ChannelState_ACTIVE, MC_ChannelState_IDLE.

Notice that the UM provides additional states for the channel that are not supported by MfP.

### Image Scaling

The Picolo series acquisition boards can provide image scaling factor to the acquired video stream on-the-fly. This provides a scaled down version of video image to the user.

In MfP, applying a horizontal scaling factor is done by calling:

MultiCamChannelSetParameterInt (hChannel, EC_PARAM_ImageScaleX, ScaleX);

In MfP, applying a vertical scaling factor is done by calling:

MultiCamChannelSetParameterInt (hChannel, EC_PARAM_ImageScaleY, ScaleY);

Where ScaleX and ScaleY are 32-bit fixed point values (16:16). Bits [31-16] are the signed integer part; bits [15-0] are the unsigned fractional part.

In UM, this is done by calling:

McSetParamInt (hChannel, MC_Scaling, MC_Scalling_ENABLE);

McSetParamInt (hChannel, MC_ImageSizeX, SizeX);

McSetParamInt (hChannel, MC_ImageSizeY, SizeY);

Where SizeX and SizeY are expressed as a number of rows and columns of the acquired image.

The conversion formula between Scale (X\Y) and Size(X/Y) is

$$\text{Size(X/Y)} = \frac{\text{Acquire(X/Y)*65536}}{\text{Scale(X/Y)}}$$

Where AcquireX = 768 and AcquireY = 576 for CCIR / PAL and

AcquireX = 640 and AcquireY = 480 for EIA / NTSC

Notice PicTmgXXX geometric parameters available in MfP are not supported in UM.

### Image Cropping

The Picolo series acquisition boards can provide image cropping factor to the acquired video stream on-the-fly. This offers several smart ways to adjust the grabbing window inside the Camera Active Window.

In MfP, to apply a horizontal offset (must be positive), call:
MultiCamChannelSetParameterInt (hChannel, EC_PARAM_ImageOffsetX, OffsetX);
To apply a vertical offset (must be positive), call:
MultiCamChannelSetParameterInt (hChannel, EC_PARAM_ImageOffsetY, OffsetY);
In MfP, applying image horizontal size is done by calling:
MultiCamChannelSetParameterInt (hChannel, EC_PARAM_ImageSizeX, SizeX);
In MfP, applying image vertical size is done by calling:
MultiCamChannelSetParameterInt (hChannel, EC_PARAM_ImageSizeY, SizeY);

In UM, this is done by calling:
McSetParamInt (hChannel, MC_GrabWindow, MC_GrabWindow_MAN);
// applying image horizontal offset (may be negative)
McSetParamInt (hChannel, MC_OffsetX_Px, OffsetX);
// applying image vertical offset (may be negative)
McSetParamInt (hChannel, MC_OffsetY_Ln, OffsetY);
// applying image horizontal size
McSetParamInt (hChannel, MC_WindowX_Px, SizeX);
// applying image vertical size
McSetParamInt (hChannel, MC_WindowY_Ln, SizeY);
Where SizeX and SizeY are expressed as a number of rows and columns of the acquired image.

### Flipping the Image

The Picolo series acquisition boards are capable of vertical mirror transformation. The picture can be flipped in the Y direction.

In MfP, this is done by calling:
MultiCamChannelSetParameterInt (hChannel, EC_PARAM_ImageFlipY, Flip);
Where Flip could be one of the following values: 0 if no flipping; 1 if flipping.

In UM, this is done by calling:
McSetParamInt (hChannel, MC_ImageFlipY, Flip);
Where Flip could be one of these values: MC_ImageFlipY_OFF if no flipping; MC_ImageFlipY_ON if flipping.

# Surface Management

The surface is a container where a 2D image can be stored. In the acquisition process, the surface is the destination buffer in the host computer memory that holds the grabbed images from the camera. A cluster is a set of surfaces having compatible characteristics, but different locations. All surfaces belonging to a cluster should be able to accept images coming from the same source through a given channel.

## Surface Creation

In MfP,

MultiCamSurfaceCreate (&SurfaceInfo);

Where SurfaceInfo holds the information about the surface that has to be created.

Since MultiCam 4.5, UM automatically creates the surfaces and automatically allocates the memory buffers, if not done by the application. If the number of surfaces to be created in a channel is not specified, UM automatically creates 3 surfaces for each channel.

SurfaceCount is a new parameter to indicate the number of surfaces in a channel. The user may change the SurfaceCount to another value before channel activation.

McSetParamInt (hChannel, MC_SurfaceCount, Count)

Where Count is the number of surfaces to be allocated for this channel. The maximum number of surfaces assigned to a channel is 4096 and the maximum number of surfaces instantiated within an application is also 4096.

In EMC2, the surface allocation is done automatically by the MultiCam driver.

## Adding Surfaces to the Cluster

In MfP,

MultiCamChannelAddSurface (hChannel, hSurface0);

Where hSurface0 is the surface to be added to the cluster belonging to the channel.

In UM, this process is not required anymore since MultiCam 4.5

However, it is still possible to add the surfaces to the cluster manually.

In EMC2, this process is not required anymore since MultiCam 4.5

However, it is still possible to add the surfaces to the cluster manually.

## Surface Deletion

In MfP,

MultiCamSurfaceDelete(hSurface0);

Where hSurface0 is the surface to be deleted.

In UM, this process is done automatically by the driver before terminating the driver communication. This applies to MultiCam 4.5 and later.

## Surface Parameter

The MfP driver provides time stamp parameters relating to the surface.

In the following section, application of the MultiCam parameters is exactly the same in UM and EMC2.

### Time Stamp

The time stamp is the moment when the surface was acquired. It is expressed as time elapsed since midnight (00:00:00), January 1, 1970, coordinated universal time (UTC), according to the system clock.

In MfP, to get the time stamp expressed in seconds:

MultiCamSurfaceGetParameterInt (hSurface, EC_PARAM_TimeStamp_s, *TimeStamp);

Where the result is a 32-bit signed integer value.

In MfP, to get the time stamp expressed in milliseconds:

MultiCamSurfaceGetParameterInt (hSurface, EC_PARAM_TimeStamp_ms, *TimeStamp);

Where the result is a 64-bit signed integer value.

In UM, to get the time stamp expressed in seconds:

McGetParamInt (hSurface, MC_TimeAnsi, *TimeStamp);

Where the result is a 32-bit signed integer value.

In UM, to get the time stamp expressed in milliseconds:

McGetParamInt (hSurface, MC_TimeStamp_us+MC_LOW_PART, *LowTimeStamp);

McGetParamInt (hSurface, MC_TimeStamp_us+MC_HIGH_PART, *HighTimeStamp);

Where the result is a collection of two 32-bit integers: one for the low part (MC_LOW_PART) and one for the high part (MC_HIGH_PART).

# Signal Management

A MultiCam signal is an entity representing a particular event issued from a channel and able to interact with the application. MultiCam provides two mechanisms to synchronize the application operation with the system.

| Mechanism | Description |
|---|---|
| Callback | Mechanism involving a user written function automatically called when a pre-defined signal occurs. |
| Waiting | Dedicated mechanism allowing for a thread to wait for the occurrence of a pre-defined signal. |

## Signal Handling Differences

The main difference between the way MfP and UM handles the signals is that in MfP all channels share the same message handler, while in UM each channel has its own handler that handles all the messages related to it.

In MfP, there is one message handler for each given type of signal to handle all channels in the same system.

In UM, each channel has its own message handler that handles all messages for this channel.

In EMC2, each channel has its own message handler that handles all messages for this channel.

## Registering the Callback Signaling

Before using the callback signal, it has to be registered first.

In MfP, the signal is registered for callback using:

MultiCamSignalRegister (eventID, CallBackFunction, context);

Where eventID can be one of the following: EC_SIGNAL_SURFACE_STATE, EC_SIGNAL_SURFACE_AVAILABLE, EC_SIGNAL_ASM_ERROR or EC_SIGNAL_IO_ACTIVITY

To unregister the callback signal, use MultiCamSignalUnregister function.

By default, all signals are enabled. To change the enable/disable state of each callback signal, use MultiCamSignalMask function.

In UM, the signal is registered for callback using:

McRegisterCallback (hChannel, CallBackFunction, Context);

McSetParamInt (hChannel, MC_SignalEnable + SignalID, MC_SignalEnable_ON);

There is no function to unregister the callback signal.

By default, all signals are disabled. To change the enable/disable state of each callback signal, set MC_SignalEnable_ON or MC_SignalEnable_OFF.

EMC2 eases the use of the MultiCam callbacks. Each channel manages its own callbacks by means of registering a given MultiCam signal to a specific callback function.

The signal is registered for callback using:

RegisterCallback (*obj, (T::*f) (Channel &chan, SignalInfo &info), MCSIGNAL SignalID);

Where obj is the object of the class T and f is the member function of class T that will utilize the callback signal.

```
// Example
class MySource
{
        void OnSurfaceFilled(Channel &c, SignalInfo &i);
};
…
```

```
MySource      Source;
channel->RegisterCallback(&Source, MySource::OnSurfaceFilled, MC_SIG_SURFACE_FILLED);
channel->SetParam(MC_SignalEnable+MC_SIG_SURFACE_FILLED, MC_SignalEnable_ON);
```

There is no function to unregister the callback signal.

By default, all signals are disabled. To change the enable/disable state of each callback signal, use MC_SignalEnable_ON or MC_SignalEnable_OFF

Signal conversion matrix:

| MfP | MU | Signal Information |
|---|---|---|
| EC_SIGNAL_SURFACE_STATE | MC_SIG_SURFACE_FILLED | This signal is issued when a surface of the destination cluster enters the FILLED state. |
| EC_SIGNAL_SURFACE_AVAILABLE | MC_SIG_SURFACE_PROCESSING | This signal is issued when a surface of the destination cluster enters the PROCESSING state. |
| EC_SIGNAL_ASM_ERROR | MC_SIG_ACQUISITION_FAILURE | This signal is issued when the channel does not receive video signal. |
| EC_SIGNAL_IO_ACTIVITY | (Not available) | This signal is sent by the MultiCam device driver when some configuration of I/O lines activity occurs |

## Callback Information

The callback function receives a set of driver parameters when a predefined event happened.

In MfP the parameters are gathered in a structure called ECEVENTINFO which contains the following members:

EventID identifies the event which triggered the callback.

ChannelID identifies the channel for which the event happened.

SurfaceID identifies the surface for which the event happened.

Flags identifies in which condition the event happened.

Timecode identifies the order number of the surface associated with the event.

MfP also provide a context that is user provided when the callback was registered.

In UM, the parameters are gathered in a structure called MCCALLBACKINFO which contains the following members:

Context is equivalent to Context provided in MfP.

Instance is equivalent to ChannelID provided in MfP.

Signal this is equivalent to the EventID provided in MfP.

SignalInfo this is equivalent to the SurfaceID provided in MfP.

Timecode is a Surface parameter; it can be retrieved by calling:

McGetParamInt (hSurface, MC_TimeCode, *SurfaceCode);

Notice the Flags parameter is not available in UM.

In EMC2, each channel manages its own callbacks by registering a given MultiCam signal to a specific callback function. Therefore the information about the event that triggers the callback and the channel for which it happened is immediately known.

```
// Example
class MySource
{
        // Callback function
        void OnSurfaceFilled(Channel &c, SignalInfo &i);
};
```

The callback function explicitly receives a pointer to the pertaining channel and an instance of the SignalInfo class which contains the following members:

Signal identifies the event which triggered the callback.

Surf identifies the surface for which the event happened.

The TimeCode can be retrieve by calling the member function of the surface object:

GetParam (MC_TimeCode, *SurfaceCode);

## Waiting Signals

The driver provides a way to allow the application to wait for the occurrence of a MultiCam signal in a thread. The calling thread is blocked until one of following occurs: the signal is generated or the time-out interval elapses.

In MfP, this is done by calling:

MultiCamSignalWait (SignalID);

In UM, this is done by calling:

McWaitSignal (hChannel, SignalID, TimeOut, SignalInformation);

Only a single signal can be waited for. The SignalID should be enabled with the MC_SignalEnable parameter.

In EMC2, this is done by calling the member function belonging to the channel:

WaitSignal (SignalID, Timeout, SignalInfomation);

# MultiCam Operation

## Starting the Acquisition

The acquisition sequence is activated when the channel is activated while the acquisition flag is started, whichever occurs last. As long as the conjunction “channel is active” and “acquisition is started” is realized, the acquisition sequence remains alive.

Starting the acquisition flag is a system-wide operation.

In MfP, this is done by calling:
MultiCamSystemAcquisitionStart();

In UM and in EMC2, this step is not required. It is done automatically by the driver.

Activating the channel is a matter of assigning the channel to ACTIVE state. This is a channel-specific operation.

In MfP, this is done by calling:
MultiCamChannelSetParameterInt (hChannel, EC_PARAM_ChannelState, CHANNEL_STATE_ACTIVE);

In UM, this is done by calling:
McSetParamInt (hChannel, MC_ChannelState, MC_ChannelState_ACTIVE);

## Stopping the Acquisition

The application can stop the acquisition sequence for all channels simultaneously by stopping the acquisition flag.

In MfP, this is done by calling:
MultiCamSystemAcquisitionStop();

In UM and in EMC2, this step is not required. It is done automatically by the driver.

It can also stop the acquisition sequence on a per-channel basis by setting the channel to IDLE.

In MfP, this is done by calling:
MultiCamChannelSetParameterInt (hChannel, EC_PARAM_ChannelState, CHANNEL_STATE_IDLE);

In UM, this is done by calling:
McSetParamInt (hChannel, MC_ChannelState, MC_ChannelState_IDLE);

## Setting the Number of Acquisitions

The driver provides a way to specify the number of acquisitions constituting a sequence. Setting acquisition count to -1 results in indefinitely repeated acquisition. A user break is required to terminate a sequence.

In MfP, this is done by calling:

MultiCamChannelSetParameterInt (hChannel, EC_PARAM_RepeatGrabCount, GrabCount);

In UM, this is done by calling:

McSetParamInt (hChannel, MC_SeqLength_Fr, Count);

## Controlling the Acquisition Rate

The driver provides a way to control the acquisition rate or frequency.

In MfP, this is done by calling:

MultiCamChannelSetParameterInt (hChannel, EC_PARAM_InitGrabDelay, InitDelay);

MultiCamChannelSetParameterInt (hChannel, EC_PARAM_RepeatGrabDelay, Delay);

In UM, this is done by calling:

McSetParamInt (hChannel, MC_NextTrigMode, MC_NextTrigMode_PERIODIC);

McSetParamInt (hChannel, MC_TargetFrameRate_Hz, FrameRate);

Where FrameRate specifies the acquisition repetition rate.

*America,* **Euresys Inc.**
500 Park Boulevard, suite 525, Itasca, Illinois 60143
Phone: 1-866-EURESYS

*Asia,* **Euresys Pte. Ltd.**
627A Aljunied Road, #08-09 BizTech Centre, Singapore 389837
Phone: +65 6748 0085 - Fax: +65 6841 2137

*Japan,* **Euresys s.a. Japan Representative Office**
AIOS Hiroo Building 8F, Hiroo 1-11-2, Shibuya-ku, Tokyo 150-0012
Phone: +81 3 5447-1256 - Fax: +81 3 5447-0529

*Europe,* **Euresys s.a. Corporate Headquarters**
14, Avenue du Pré-Aily, B-4031 Angleur, Belgium
Phone: +32 4 367 72 88 - Fax: +32 4 367 74 66

EURESYS™
Excellence in vision

***www.euresys.com*** ***info@euresys.com***

Your distributor