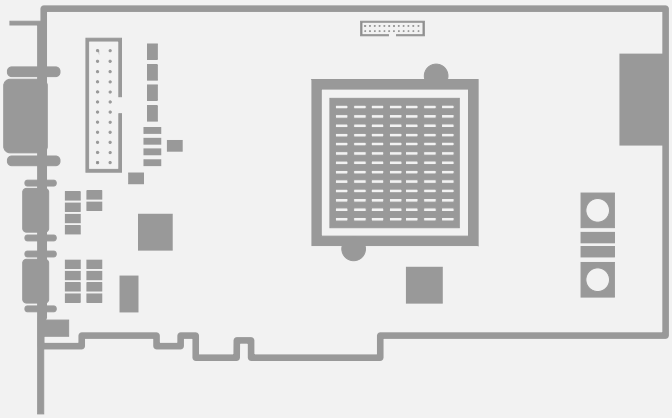
euresys
Empowering Computer Vision

REFERENCE

MultiCam

MultiCam API References



 - Doc. D403EN-MultiCam API References-6.19.0.4054 built on 2024-04-16

This documentation is provided with **MultiCam 6.19.0** (doc build **4054**).
www.euresys.com

Contents

1. About This Document

1.1. Document Revision History

Date	Version	Description
2019-07-24	6.17	MultiCam 6.17
2018-08-10	6.15.1	First edition in MadCap Flare

1.2. Document Changes

MultiCam 6.17

New topics:

- "Surface::Convert Method" on page 38
- "Pixel Format Conversion" on page 68

PART I
MULTICAM C++ REFERENCE

1. Using the MultiCam Libraries with C++

Supported Platforms

Refer to the Release Specification section of the MultiCam Release Notes.

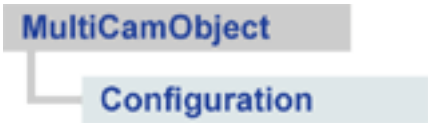
Requested Files

The C++ classes and functions can be used if including the requested header files (.h) and linking to the import library file (.lib) associated to a dynamic-link library (.dll).

- C++ headers (.h files) can be found in the **Include** folder, under the installation root.
- Import libraries (.lib files) can be found in the folder corresponding to the supported compilers.
- Dynamic link libraries counterparts (.dll files) can be found in the system folder.

2. Classes

2.1. Configuration Class



The Configuration object gives access to all MultiCam parameters dedicated to the control of system-wide features.

The system is a set of Euresys boards installed inside a computer. The Configuration object also addresses any hardware or software element of the computer requesting some degree of control for the MultiCam system operation.

The Configuration object exists in one instance per application. The user is not allowed to create a Configuration object. It is natively made available to the application through the Config global variable.

Requirements

Header: MultiCamCpp.h

Namespace: Euresys::MultiCam

Platforms

See Also

MultiCamObject Class

2.2. Board Class

The Board object gives access to all MultiCam parameters dedicated to the control of board features. It also addresses the access of I/O lines from an application program, implementing the general-purpose I/O functionality.

The Board object exists in one instance for each Euresys board installed inside a computer. The user is not allowed to create Board objects. They are natively made available to the application through the Boards global object.

Requirements

Header: MultiCamCpp.h

Namespace: Euresys::MultiCam

Platforms

See Also

MultiCamObject Class

2.3. BoardList Class

The BoardList object gives access to the Board objects representing Euresys frame grabbers in the host computer.

There is only one BoardList instance per application: the global Boards variable. The user is not allowed to create a BoardList object.

Properties and Methods

GetBoardByBoardIdentifier	Finds a reference to a Board object using its board identifier.
GetBoardByBoardName	Finds a reference to a Board object using its board name.
GetBoardByDriverIndex	Finds a reference to a Board object using its index.
GetBoardByPciPosition	Finds a reference to a Board object using its position on the PCI bus.
GetCount	Returns the number of elements in the board list.
operator[]	Finds a reference to a Board object using its index.

Requirements

Header: MultiCamCpp.h
Namespace: Euresys::MultiCam
Platforms

See Also

MultiCamObject Class

BoardList::GetBoardByBoardIdentifier Method

Finds a reference to a Board object using its board identifier.

```
[C++]
Board* BoardList::GetBoardByBoardIdentifier(
  const char* boardIdentifier
);
```

Parameters

boardIdentifier

User allocated character string holding the board identifier.

Remarks

The board 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.

"GRABLINK_FULL_000123" is an example.

See Also

BoardList Class | BoardList::GetBoardByBoardName | BoardList::GetBoardByDriverIndex | BoardList::GetBoardByPciPosition

BoardList::GetBoardByBoardName Method

Finds a reference to a Board object using its board name.

```
[C++]
Board* BoardList::GetBoardByBoardName(
  const char* boardName
);
```

Parameters

boardName

User allocated character string holding the board name.

Remarks

The designation is based on the name given to a board. The name is a string of maximum 16 ASCII characters.

To give a name to a board, use the NameBoard board parameter.

See Also

BoardList Class | BoardList::GetBoardByBoardIdentifier | BoardList::GetBoardByDriverIndex | BoardList::GetBoardByPciPosition

BoardList::GetBoardByDriverIndex Method

Finds a reference to a Board object using its index.

```
[C++]
Board* BoardList::GetBoardByDriverIndex(
  int driverIndex
);
```

Parameters

driverIndex

Index of the board in the board list.

Remarks

The designation is based on the board location in the Boards global object. The set of MultiCam compliant boards are assigned a set of consecutive integers, starting at zero. The indexing order is system-dependent.

See Also

BoardList Class | BoardList::GetBoardByBoardIdentifier | BoardList::GetBoardByBoardName | BoardList::GetBoardByPciPosition

BoardList::GetBoardByPciPosition Method

Finds a reference to a Board object using its position on the PCI bus.

```
[C++]
Board* BoardList::GetBoardByPciPosition(
  int pciPosition
);
```

Parameters

pciPosition

Number representing the PCI position.

Remarks

The designation is based on a number associated to a 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.

See Also

BoardList Class | BoardList::GetBoardByBoardIdentifier | BoardList::GetBoardByBoardName | BoardList::GetBoardByDriverIndex

BoardList::GetCount Method

Returns the number of elements in the board list.

```
[C++]
int BoardList::GetCount();
```

See Also

BoardList Class

BoardList::operator[ ] Operator

Finds a reference to a Board object using its index.

```
[C++]
Board* BoardList::operator[] (
  int driverIndex
);
```

Parameters

driverIndex

Index of the board in the board list.

Remarks

The designation is based on the board location in the board list. The set of MultiCam compliant boards are assigned a set of consecutive integers, starting at zero. The indexing order is system-dependent.

See Also

BoardList Class | BoardList::GetBoardByBoardIdentifier | BoardList::GetBoardByBoardName | BoardList::GetBoardByDriverIndex | BoardList::GetBoardByPciPosition

2.4. Channel Class

The Channel object represents the path from the camera to the surface, that is the image acquired in the computer memory. It performs the image acquisition.

Usually, the application creates one channel per camera. After creation, the channel is configured by setting relevant parameters with the GetParam and SetParam methods.

Once configured, the channel is activated with the SetActive method. It is stopped with the SetIdle method.

Optionally and before calling SetActive, the channel may be prepared with the Prepare method. This ensures that all time-consuming configuration operations are done and that the channel will immediately perform image acquisitions upon activation.

When active, the channel performs image acquisition. It reports its activity by executing callback functions previously registered with RegisterCallback. As an alternative, the WaitForSignal and GetSignalInfo methods may be used to synchronize with the channel activity.

The callback functions and the GetSignalInfo method report, when applicable, the Surface objects representing the images acquired in computer memory.

Surfaces correspond to EImage... objects. The synchronization between Surface and EImage... is ensured through the MultiCam functions.

Properties and Methods

Channel	Constructs a Channel object.
GetSignalInfo	Retrieves the information associated with a MultiCam signal.
Prepare	Reduces the Channel activation time by previously setting the creation and configuration parameters.
RegisterCallback	Registers the callback function for a given signal.
SetActive	Activates the channel.
SetIdle	Ends the channel acquisition sequence.

UnRegisterCallback	Unregisters the callback function for a given signal.
WaitForSignal	Waits for a MultiCam signal and provides the corresponding SignalInfo object.

Requirements

Header: MultiCamCpp.h
Namespace: Euresys::MultiCam
Platforms

See Also

MultiCamObject Class

Channel::Channel Constructor

Constructs a Channel object. The channel is created on a given Board. The caller specifies the connector where the camera is connected.

```
[C++]
Channel::Channel(Board*, int);
Channel::Channel(Board*, const char*);
```

See Also

Channel Class | Channel::WaitForSignal | SignalInfo Class

Channel::Channel Constructor (Board*, int)

Constructs a Channel object. The channel is created on a given Board. The caller specifies the connector where the camera is connected.

```
[C++]
Channel::Channel(
  Board* board,
  int connector
);
```

Parameters

board

Reference to the Board object.

connector

Identifier of the connector.

See Also

Channel Class | Channel::Channel Constructor

Channel::Channel Constructor (Board*, const char*)

Constructs a Channel object. The channel is created on a given Board. The caller specifies the connector where the camera is connected.

```
[C++]
Channel::Channel(
  Board* board,
  const char* connector
);
```

Parameters

board

Reference to the Board object.

connector

Name of the connector.

See Also

Channel Class | Channel::Channel Constructor

Channel::GetSignalInfo Method

Retrieves the information associated with a MultiCam signal. This function is used in combination with WaitForSignal. Once a signal is issued, calling GetSignalInfo fills a SignalInfo object with relevant information.

```
[C++]
void Channel::GetSignalInfo(
  MCSIGNAL signal,
  SignalInfo& info
);
```

Parameters

signal

Identifier of the signal.

info

User allocated SignalInfo object.

See Also

Channel Class | Channel::WaitForSignal | SignalInfo Class

Channel::RegisterCallback Method

Registers the callback function for a MultiCam signal.

```
[C++]
void Channel::RegisterCallback(
  T *owner,
  void (T::*callbackMethod)(Channel&, SignalInfo&),
  MCSIGNAL signal
);
```

Parameters

owner

Reference to the object that contains the callback method.

(T::*callbackMethod)(Channel& ch, SignalInfo& info)

Address of the callback method.
ch: the Channel object that caused the signal.
info: SignalInfo object holding signal information.

signal

Identifier of the signal.

Description

The callback method from the object owner will be called by MultiCam each time a given signal is issued.

The callback method is executed in a dedicated thread.

Example

```
class Foo {
// ...
Channel *pChannel;
public:
void OnSurfaceProcessing(Channel &channel, SignalInfo &info);// ...
};
void Foo::OnSurfaceProcessing(Channel &channel, SignalInfo &info) {
cout << "Signal " << info.Signal << endl;
// ...UpdateImageConfig(*info.Surf, someEImage);
// ...
}
BOOL Foo::Init() {
// ...
pChannel->RegisterCallback(this, &OnSurfaceProcessing, MC_SIG_SURFACE_PROCESSING);
// ...
}
```

See Also

Channel Class | Channel::UnRegisterCallback

Channel::Prepare Method

Channel creation and configuration time-consuming tasks are best performed after all relevant parameters have been set and before channel activation.

[C++]
```
void Channel.Prepare();
```

Remarks

Channel creation and configuration involve some time-consuming tasks like firmware upload, memory allocation... These time-consuming tasks are best performed after all relevant parameters have been set but before channel activation.

The Prepare method is optionally called after channel configuration and before calling SetActive.

When doing so, all time-consuming tasks are performed during the Prepare call. This ensures that the channel is immediately capable of doing image acquisition upon activation.

If Prepare was not called, or if parameters were changed after calling Prepare, the time-consuming tasks will be performed when calling SetActive.

See Also

Channel Class | Channel::SetActive

Channel::SetActive Method

Activates the channel. When active, the channel performs image acquisition according to its configuration. It reacts to acquisition triggers when configured to do so.

[C++]
```
void Channel::SetActive();
```

See Also

Channel Class | Channel::SetIdle

Channel::SetIdle Method

Ends the channel acquisition sequence. When idle, the channel does not acquire images nor responds to acquisition triggers.

[C++]
```
void Channel::SetIdle();
```

See Also

Channel Class | Channel::SetActive

Channel::UnregisterCallback Method

Unregisters the callback function for a given signal.

```
[C++]
void Channel::UnregisterCallback(
  MCSIGNAL signal
);
```

Parameters

signal

Identifier of the signal.

See Also

Channel Class | Channel::RegisterCallback

Channel::WaitForSignal Method

Waits for a MultiCam signal and provides, when the signal is issued, the corresponding SignalInfo object.

```
[C++]
void Channel::WaitForSignal(
  MCSIGNAL signal,
  unsigned int timeout,
  SignalInfo& info
);
```

Parameters

signal

Identifier of the signal.

timeout

Time-out value, expressed in milliseconds (ms).

info

SignalInfo object corresponding to the signal.

See Also

Channel Class | Channel::GetSignalInfo | SignalInfo Class

2.5. Exception Class

MultiCam reports errors through the exception system. The Exception object stores information about abnormal conditions.

Properties and Methods

What	Returns the description of the exception.

Requirements

Header: MultiCamCpp.h
Namespace: Euresys::MultiCam
Platforms

See Also

MultiCamObject Class

Exception::What Method

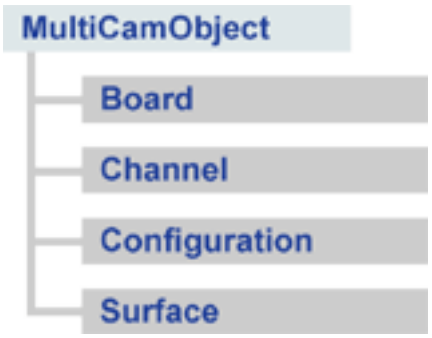
Returns the description of the exception.

```
[C++]
const char* Exception::What();
```

See Also

Exception Class

2.6. MultiCamObject Class



Properties and Methods

GetParam	Returns the value of a MultiCam parameter.
SetParam	Sets the value of a MultiCam parameter.

Requirements

Header: MultiCamCpp.h

Namespace: Euresys::MultiCam

Platforms

See Also

Configuration Class | Board Class | Channel Class | Surface Class

MultiCamObject::GetParam Method

Returns the value of a MultiCam parameter.

```
[C++]
void MultiCamObject::GetParam(MCPARAMID, int&);
void MultiCamObject::GetParam(MCPARAMID, unsigned int&);
void MultiCamObject::GetParam(MCPARAMID, double&);
void MultiCamObject::GetParam(MCPARAMID, Surface*&);
void MultiCamObject::GetParam(MCPARAMID, char*, int);
void MultiCamObject::GetParam(MCPARAMID, long long int&);
void MultiCamObject::GetParam(MCPARAMID, void*&);
void MultiCamObject::GetParam(const char*, int&);
void MultiCamObject::GetParam(const char*, unsigned int&);
void MultiCamObject::GetParam(const char*, double&);
void MultiCamObject::GetParam(const char*, Surface*&);
void MultiCamObject::GetParam(const char*, char*, int);
void MultiCamObject::GetParam(const char*, long long int&);
void MultiCamObject::GetParam(const char*, void*&);
```

Remarks

There are two ways to designate the parameter: by identifier and by name.

Independently of the parameter type, its value may be returned as int, unsigned int, double or character string. In addition, some parameters have the Surface type.

See Also

MultiCamObject Class | MultiCamObject::SetParam

MultiCamObject::GetParam Method (MCPARAMID, int&)

Retrieves the value of a MultiCamObject parameter. Independently of the parameter type, its value is returned as an int.

```
[C++]
void MultiCamObject::GetParam(
  MCPARAMID param,
  int& value
);
```

Parameters

param

MultiCam parameter identifier.

value

Parameter value.

See Also

MultiCamObject Class | MultiCamObject::GetParam | MultiCamObject::SetParam

MultiCamObject::GetParam Method (MCPARAMID, unsigned int&)

Retrieves the value of a MultiCamObject parameter. Independently of the parameter type, its value is returned as an unsigned int.

```
[C++]
void MultiCamObject::GetParam(
  MCPARAMID param,
  unsigned int& value
);
```

Parameters

param

MultiCam parameter identifier.

value

Parameter value.

See Also

MultiCamObject Class | MultiCamObject::GetParam | MultiCamObject::SetParam

MultiCamObject::GetParam Method (MCPARAMID, double&)

Retrieves the value of a MultiCamObject parameter. Independently of the parameter type, its value is returned as a double.

```
[C++]
void MultiCamObject::GetParam(
  MCPARAMID param,
  double& value
);
```

Parameters

param

MultiCam parameter identifier.

value

Parameter value.

See Also

MultiCamObject Class | MultiCamObject::GetParam | MultiCamObject::SetParam

MultiCamObject::GetParam Method (MCPARAMID, Surface*&)

Retrieves the value of a MultiCamObject parameter. Independently of the parameter type, its value is returned as a Surface.

```
[C++]
void MultiCamObject::GetParam(
  MCPARAMID param,
  Surface*& value
);
```

Parameters

param

MultiCam parameter identifier.

value

Parameter value.

See Also

MultiCamObject Class | MultiCamObject::GetParam | MultiCamObject::SetParam

MultiCamObject::GetParam Method (MCPARAMID, char*, int)

Returns the value of a MultiCam parameter. Independently of the parameter type, its value is returned as a character string.

```
[C++]
void MultiCamObject::GetParam(
  MCPARAMID param,
  char* value,
  int maxLength
);
```

Parameters

param

MultiCam parameter identifier.

value

Caller-allocated character string. After the call, it holds the parameter value.

maxLength

Size of the caller-allocated character string.

See Also

MultiCamObject Class | MultiCamObject::GetParam | MultiCamObject::SetParam

MultiCamObject::GetParam Method (MCPARAMID, long long int&)

Retrieves the value of a MultiCamObject parameter. Independently of the parameter type, its value is returned as a long long int.

```
[C++]
void MultiCamObject::GetParam(
  MCPARAMID param,
  long long int& value
);
```

Parameters

param

MultiCam parameter identifier.

value

Parameter value.

See Also

MultiCamObject Class | MultiCamObject::GetParam | MultiCamObject::SetParam

MultiCamObject::GetParam Method (MCPARAMID, void*&)

Retrieves the value of a MultiCamObject parameter. Independently of the parameter type, its value is returned as a void.

```
[C++]
void MultiCamObject::GetParam(
  MCPARAMID param,
  void*& value
);
```

Parameters

param

MultiCam parameter identifier.

value

Parameter value.

See Also

MultiCamObject Class | MultiCamObject::GetParam | MultiCamObject::SetParam

MultiCamObject::GetParam Method (const char*, int&)

Retrieves the value of a MultiCamObject parameter. Independently of the parameter type, its value is returned as an int.

```
[C++]
void MultiCamObject::GetParam(
  const char* param,
  int& value
);
```

Parameters

param

Caller-allocated character string holding the MultiCam parameter name.

value

Parameter value.

See Also

MultiCamObject Class | MultiCamObject::GetParam | MultiCamObject::SetParam

MultiCamObject::GetParam Method (const char*, unsigned int&)

Retrieves the value of a MultiCamObject parameter. Independently of the parameter type, its value is returned as an unsigned int.

```
[C++]
void MultiCamObject::GetParam(
  const char* param,
  unsigned int& value
);
```

Parameters

param

Caller-allocated character string holding the MultiCam parameter name.

value

Parameter value.

See Also

MultiCamObject Class | MultiCamObject::GetParam | MultiCamObject::SetParam

MultiCamObject::GetParam Method (const char*, double&)

Retrieves the value of a MultiCamObject parameter. Independently of the parameter type, its value is returned as a double.

```
[C++]
void MultiCamObject::GetParam(
  const char* param,
  double& value
);
```

Parameters

param

Caller-allocated character string holding the MultiCam parameter name.

value

Parameter value.

See Also

MultiCamObject Class | MultiCamObject::GetParam | MultiCamObject::SetParam

MultiCamObject::GetParam Method (const char*, Surface*&)

Retrieves the value of a MultiCamObject parameter. Independently of the parameter type, its value is returned as a Surface.

```
[C++]
void MultiCamObject::GetParam(
  const char* param,
  Surface*& value
);
```

Parameters

param

Caller-allocated character string holding the MultiCam parameter name.

value

Parameter value.

See Also

MultiCamObject Class | MultiCamObject::GetParam | MultiCamObject::SetParam

MultiCamObject::GetParam Method (const char*, char*, int)

Returns the value of a MultiCam parameter. Independently of the parameter type, its value is returned as a character string.

```
[C++]
void MultiCamObject::GetParam(
  const char* param,
  char* value,
  int maxLength
);
```

Parameters

param

Caller-allocated character string holding the MultiCam parameter name.

value

Caller-allocated character string. After the call, it holds the parameter value.

maxLength

Size of the caller-allocated character string.

See Also

MultiCamObject Class | MultiCamObject::GetParam | MultiCamObject::SetParam

MultiCamObject::GetParam Method (const char*, long long int&)

Retrieves the value of a MultiCamObject parameter. Independently of the parameter type, its value is returned as a long long int.

```
[C++]
void MultiCamObject::GetParam(
  const char* param,
  long long int& value
);
```

Parameters

param

MultiCam parameter identifier.

value

Parameter value.

See Also

MultiCamObject Class | MultiCamObject::GetParam | MultiCamObject::SetParam

MultiCamObject::GetParam Method (const char*, void*&)

Retrieves the value of a MultiCamObject parameter. Independently of the parameter type, its value is returned as a void.

```
[C++]
void MultiCamObject::GetParam(
  const char* param,
  void*& value
);
```

Parameters

param

MultiCam parameter identifier.

value

Parameter value.

See Also

MultiCamObject Class | MultiCamObject::GetParam | MultiCamObject::SetParam

MultiCamObject::SetParam Method

Sets the value of a MultiCam parameter.

```
[C++]
void MultiCamObject::SetParam(MCPARAMID, int);
void MultiCamObject::SetParam(MCPARAMID, unsigned int);
void MultiCamObject::SetParam(MCPARAMID, double);
void MultiCamObject::SetParam(MCPARAMID, Surface&);
void MultiCamObject::SetParam(MCPARAMID, const char*);
void MultiCamObject::SetParam(MCPARAMID, long long int);
void MultiCamObject::SetParam(MCPARAMID, void*);
void MultiCamObject::SetParam(const char*, int);
void MultiCamObject::SetParam(const char*, unsigned int);
void MultiCamObject::SetParam(const char*, double);
void MultiCamObject::SetParam(const char*, Surface&);
void MultiCamObject::SetParam(const char*, const char*);
void MultiCamObject::SetParam(const char*, long long int);
void MultiCamObject::SetParam(const char*, void*);
```

Remarks

There are two ways to designate the parameter: by identifier and by name.

Independently of the parameter type, the value may be passed as int, unsigned int, double or character string. In addition, some parameters have the Surface type.

See Also

MultiCamObject Class | MultiCamObject::GetParam

MultiCamObject::SetParam Method (MCPARAMID, int)

Sets the value of a MultiCam parameter. Independently of the parameter type, its value is passed as an int.

```
[C++]
void MultiCamObject::SetParam(
  MCPARAMID param,
  int value
);
```

Parameters

param

MultiCam parameter identifier.

value

Parameter value.

See Also

MultiCamObject Class | MultiCamObject::SetParam | MultiCamObject::GetParam

MultiCamObject::SetParam Method (MCPARAMID, unsigned int)

Sets the value of a MultiCam parameter. Independently of the parameter type, its value is passed as an unsigned int.

```
[C++]
void MultiCamObject::SetParam(
  MCPARAMID param,
  unsigned int value
);
```

Parameters

param

MultiCam parameter identifier.

value

Parameter value.

See Also

MultiCamObject Class | MultiCamObject::SetParam | MultiCamObject::GetParam

MultiCamObject::SetParam Method (MCPARAMID, double)

Sets the value of a MultiCam parameter. Independently of the parameter type, its value is passed as a double.

```
[C++]
void MultiCamObject::SetParam(
  MCPARAMID param,
  double value
);
```

Parameters

param

MultiCam parameter identifier.

value

Parameter value.

See Also

MultiCamObject Class | MultiCamObject::SetParam | MultiCamObject::GetParam

MultiCamObject::SetParam Method (MCPARAMID, Surface&)

Sets the value of a MultiCamObject parameter. Independently of the parameter type, its value is passed as a Surface.

```
[C++]
void MultiCamObject::SetParam(
  MCPARAMID param,
  Surface& value
);
```

Parameters

param

MultiCam parameter identifier.

value

Parameter value.

See Also

MultiCamObject Class | MultiCamObject::SetParam | MultiCamObject::GetParam

MultiCamObject::SetParam Method (MCPARAMID, const char*)

Sets the value of a MultiCam parameter. Independently of the parameter type, its value is passed as a character string.

```
[C++]
void MultiCamObject::SetParam(
  MCPARAMID param,
  const char* value
);
```

Parameters

param

MultiCam parameter identifier.

value

Parameter value.

See Also

MultiCamObject Class | MultiCamObject::SetParam | MultiCamObject::GetParam

MultiCamObject::SetParam Method (MCPARAMID, long long int)

Sets the value of a MultiCam parameter. Independently of the parameter type, its value is passed as a long long int.

```
[C++]
void MultiCamObject::SetParam(
  MCPARAMID param,
  long long int value
);
```

Parameters

param

MultiCam parameter identifier.

value

Parameter value.

See Also

MultiCamObject Class | MultiCamObject::SetParam | MultiCamObject::GetParam

MultiCamObject::SetParam Method (MCPARAMID, void*)

Sets the value of a MultiCam parameter.

```
[C++]
void MultiCamObject::SetParam(
  MCPARAMID param,
  void* value
);
```

Parameters

param

MultiCam parameter identifier.

value

Parameter value.

See Also

MultiCamObject Class | MultiCamObject::SetParam | MultiCamObject::GetParam

MultiCamObject::SetParam Method (const char*, int)

Sets the value of a MultiCam parameter. Independently of the parameter type, its value is passed as an int.

```
[C++]
void MultiCamObject::SetParam(
  const char* param,
  int value
);
```

Parameters

param

Caller-allocated character string holding the MultiCam parameter name.

value

Parameter value.

See Also

MultiCamObject Class | MultiCamObject::SetParam | MultiCamObject::GetParam

MultiCamObject::SetParam Method (const char*, unsigned int)

Sets the value of a MultiCam parameter. Independently of the parameter type, its value is passed as an unsigned int.

```
[C++]
void MultiCamObject::SetParam(
  const char* param,
  unsigned int value
);
```

Parameters

param

Caller-allocated character string holding the MultiCam parameter name.

value

Parameter value.

See Also

MultiCamObject Class | MultiCamObject::SetParam | MultiCamObject::GetParam

MultiCamObject::SetParam Method (const char*, double)

Sets the value of a MultiCam parameter. Independently of the parameter type, its value is passed as a double.

```
[C++]
void MultiCamObject::SetParam(
  const char* param,
  double value
);
```

Parameters

param

Caller-allocated character string holding the MultiCam parameter name.

value

Parameter value.

See Also

MultiCamObject Class | MultiCamObject::SetParam | MultiCamObject::GetParam

MultiCamObject::SetParam Method (const char*, Surface&)

Sets the value of a MultiCamObject parameter. Independently of the parameter type, its value is passed as a Surface.

```
[C++]
void MultiCamObject::SetParam(
  const char* param,
  Surface& value
);
```

Parameters

param

Caller-allocated character string holding the MultiCam parameter name.

value

Parameter value.

See Also

MultiCamObject Class | MultiCamObject::SetParam | MultiCamObject::GetParam

MultiCamObject::SetParam Method (const char*, const char*)

Sets the value of a MultiCam parameter. Independently of the parameter type, its value is passed as a character string.

```
[C++]
void MultiCamObject::SetParam(
  const char* param,
  const char* value
);
```

Parameters

param

Caller-allocated character string holding the MultiCam parameter name.

value

Parameter value.

See Also

MultiCamObject Class | MultiCamObject::SetParam | MultiCamObject::GetParam

MultiCamObject::SetParam Method (const char*, long long int)

Sets the value of a MultiCam parameter. Independently of the parameter type, its value is passed as a long long int.

```
[C++]
void MultiCamObject::SetParam(
  const char* param,
  long long int value
);
```

Parameters

param

MultiCam parameter identifier.

value

Parameter value.

See Also

MultiCamObject Class | MultiCamObject::SetParam | MultiCamObject::GetParam

MultiCamObject::SetParam Method (const char*, void*)

Sets the value of a MultiCam parameter.

```
[C++]
void MultiCamObject::SetParam(
  const char* param,
  void* value
);
```

Parameters

param

MultiCam parameter identifier.

value

Parameter value.

See Also

MultiCamObject Class | MultiCamObject::SetParam | MultiCamObject::GetParam

2.7. SignalInfo Class

The SignalInfo object conveys information about a MultiCam signal.

Properties and Methods

Signal	Represents the MCSIGNAL identifier of a MultiCam signal.
Surf	Refers to the SURFACE object related to the MultiCam signal.

Requirements

Header: MultiCamCpp.h
Namespace: Euresys::MultiCam
Platforms

See Also

MultiCamObject Class

SignalInfo::Signal Property

Represents the MCSIGNAL identifier of a MultiCam signal.

```
[C++]
MCSIGNAL SignalInfo::Signal;
```

Remarks

Read-only.

See Also

SignalInfo Class

SignalInfo::Surf Property

Refers to the Surface object related to the MultiCam signal.

```
[C++]
Surface* SignalInfo::Surf;
```

Remarks

Read-only.

See Also

SignalInfo Class

2.8. Surface Class

The Surface object represents the images acquired by the channel in the computer memory.

There is a direct relation between Surface objects and EImage... objects. *This relation involves no image buffer copy, the image buffers being shared between Surface and EImage... objects.*

A Channel acquires images in a group of Surface objects. MultiCam provides a protection mechanism to ensure that no new acquisition occurs in a surface during image processing. This mechanism is automatic: during the 'surface processing' callback, the surface is protected and no new acquisition will occur in this surface for the duration of the callback method.

In addition, the Reserve and Free methods allow controlling manually the surface protection.

Properties and Methods

Surface	Constructs a Surface object.
Reserve	Excludes the Surface object from the acquisition process.
Free	Reverts the effect of Reserve.
Convert	Handles software conversion between various pixel formats.

Requirements

Header: MultiCamCpp.h

Namespace: Euresys::MultiCam

Platforms

See Also

MultiCamObject Class

Surface::Surface Constructor

Constructs an empty Surface object.

[C++]
```
void Surface::Surface();
```

Remarks

Usually, the surface creation is not mandatory. The channel automatically creates Surface objects for image acquisition.

Surface construction is useful when the application requires control on the memory allocation process. In this case, the application configures the relevant Surface parameters through GetParam/SetParam. Surfaces are passed to the Channel through the Cluster parameter.

The surface creation step is not mandatory because the channel automatically creates Surface objects for image acquisition.

Surface construction is useful when the application requires control on the memory allocation process. In this case, the application configures the relevant Surface parameters. Surfaces are passed to the Channel through the Cluster parameter.

See Also

Surface Class

Surface::Reserve Method

Excludes the Surface object from the acquisition process. When this method is called, new image acquisition are not allowed to target this surface.

[C++]
```
void Surface::Reserve();
```

See Also

Surface Class | Surface::Free

Surface::Free Method

Reverts the effect of Reserve. When this method is called, the channel is allowed to acquire new images in the surface.

[C++]
```
void Surface::Free();
```

See Also

Surface Class | Surface::Reserve

Surface::Convert Method

Handles software conversion between various pixel formats. A user-created Surface properly configured and passed as an argument will contain the converted image buffer after calling this method.

Remarks

To configure the target Surface, the user must allocate a buffer and set required values to the various Surface parameters: SurfaceAddr, SurfaceSize, SurfacePitch, SurfaceSizeX, SurfaceSizeY, SurfaceColorFormat and SurfaceColorComponentsOrder.

```
[C++]
void Surface::Convert(Surface& convertedSurface);;
```

See Also

Surface Class

3. Functions

3.1. Initialize

Establishes the communication of the application process with the MultiCam driver.

[C++]
void Initialize();

Remarks

Before using any MultiCam function, the communication between the application process and the MultiCam driver *must* be established. This is done by calling the Initialize function.

If an application calls Initialize several times, it must call Terminate the same number of times, to adequately close the communication with the MultiCam driver.

Do not call Initialize:

- inside the constructor of a global or static object;
- inside the DLL entry point (DllMain).

Requirements

Header: MultiCamCpp.h

Namespace: Euresys::MultiCam

Platforms

See Also

Terminate

3.2. Terminate

Terminates the communication of the application process with the MultiCam driver.

[C++]
void Terminate();

Remarks

Before terminating the application, the user *must* terminate the communication of the application process with the MultiCam driver. This is done by calling the Terminate function.

If an application calls Initialize several times, it must call Terminate the same number of times, to adequately close the communication with the MultiCam driver.

Do not call Terminate:

- inside the destructor of a global or static object;
- inside the DLL entry point (DllMain).

Requirements

Header: MultiCamCpp.h

Namespace: Euresys::MultiCam

Platforms

See Also

Initialize

4. Appendix

4.1. C++ Predefined Types

The following data types are used in the MultiCam C++ Reference. They are predefined using typedef.

C++ predefined types

Predefined type	Description
MCPARAMID	Unsigned 32-bit integer. **typedef UINT32 MCPARAMID;**
MCSIGNAL	Signed 32-bit integer. **typedef int MCSTATUS;**

PART II
MULTICAM C REFERENCE

1. Introduction

Using the MultiCam Libraries with C

Supported Platforms

Refer to the Release Specification section of the MultiCam Release Notes.

Requested Files

The C++ classes and functions can be used if including the requested header files (.h) and linking to the import library file (.lib).

- C header: MultiCam.h file can be found in the Include folder, under the installation root.
- Import library: MultiCam.lib file can be found in the folder corresponding to the supported compilers.

Creating and Deleting a MultiCam Object

A MultiCam object, that is an instance of a MultiCam class, is designated by a handle of the MultiCam-defined type MCHANDLE. This type is defined in the MultiCam.h header. Most MultiCam functions use such a handle as an argument.

Class instances can be created in two different ways, "by handle" or "by name", with the MultiCam API functions McCreate or McCreateNm.

When an object is no more needed by the application, it is advisable to release the resources assigned to it. This is done with the MultiCam API function McDelete.

See also Classes in the MultiCam User Guide.

Accessing a Parameter

The native way for the application to interact with the MultiCam system is the API in C language. All parameters are consistently accessed by a unique set of get/set functions. According to the type of the parameter, and to the by-identifier or by-name access way, several versions of these get and set functions are available.

Parameters access functions

Parameter type	Access	Recommended function	
		By-identifier parameter access	By-name parameter access
Integer	Set	McSetParamInt	McSetParamNmInt
	Get	McGetParamInt	McGetParamNmInt

Parameter type	Access		Recommended function	
			By-identifier parameter access	By-name parameter access
64-bit integer	Set		McSetParamInt64	McSetParamNmInt64
	Get		McGetParamInt64	McGetParamNmInt64
Floating point	Set		McSetParamFloat	McSetParamNmFloat
	Get		McGetParamFloat	McGetParamNmFloat
String	Set		McSetParamStr	McSetParamNmStr
	Get		McGetParamStr	McGetParamNmStr
Instance	Set		McSetParamInst	McSetParamNmInst
	Get		McGetParamInst	McGetParamNmInst
Pointer	Set		McSetParamPtr	McSetParamNmPtr
	Get		McGetParamPtr	McGetParamNmPtr
Enumerated	Set	By-identifier enumerated access	McSetParamInt	McSetParamNmInt
		By-name enumerated access	McSetParamStr	McSetParamNmStr
	Get	By-identifier enumerated access	McGetParamInt	McGetParamNmInt
		By-name enumerated access	McGetParamStr	McGetParamNmStr

Accessing a Collection Element

Some MultiCam parameters exist as a collection. This applies to any parameter type.

Accessing a Collection Element **by Identifier**

In this case, the parameter reference is established with an integer identifier defined in the MultiCam McParams.h header file.

The first element of the collection is referred to with this identifier just like a non-collection parameter. The following elements are referred to with consecutive integer values starting from the identifier value.

Accessing a Collection Element **by Name**

In this case, the name allowing access to the collection element is the concatenation of the following items:

- Parameter name
- Colon (:)
- Zero-based index of the collection element.

Example: accessing the elements of the collection parameter Cluster

Collection element	By-identifier access	By-name access
First element	MC_Cluster	Cluster
First element, alternative	MC_Cluster + 0	Cluster:0
Second element	MC_Cluster + 1	Cluster:1
Third element	MC_Cluster + 2	Cluster:2

See also By-Name vs. By-Identifier Access.

2. Functions

A MultiCam function is a software item complying to the standard C language syntax.

The set of all MultiCam functions constitutes the MultiCam API (Application Programming Interface). The API in C language provides the native way for the application to interact with the MultiCam system.

The API is designed for stability. The MultiCam principles and structure are such that the MultiCam functions **do not change** in case of new functional features, new frame grabber or new camera.

The functions are organized in five categories:

- Driver connection functions: enable or disable the driver communication.
- Instance management functions: create or remove object instances.
- Parameters management functions: set and get MultiCam parameters value.
- Signaling functions: manage the event flow associated to the acquisition.
- Pixel Format Conversion functions: handles software conversion between various pixel formats.

2.1. Driver Connection

McCloseDriver

Terminates the communication of the application process with the MultiCam driver.

```
[C]
MCSTATUS McCloseDriver();
```

Remarks

- If an application successfully calls McOpenDriver several times, it must call McCloseDriver the same number of times, to adequately close the communication with the MultiCam driver.
- Do not call McCloseDriver:
  - inside the destructor of a global or static object;
  - inside the DLL entry point (DllMain).

See Also

Driver Connection | McOpenDriver

McOpenDriver

Establishes the communication of the application process with the MultiCam driver.

```
[C]
MCSTATUS McOpenDriver(
  PCCHAR MultiCamName
);
```

Parameters

MultiCamName

This argument is reserved for future functionality. Specify a NULL pointer.

Remarks

- If an application successfully calls McOpenDriver several times, it must call McCloseDriver the same number of times, to adequately close the communication with the MultiCam driver.
- Starting with MultiCam 6, on Windows operating systems, MultiCam relies on a service named "MultiCam Service". This service is automatically started when the computer boots.
- Software should only access MultiCam when this service is started. McOpenDriver will return MC_SERVICE_ERROR if the MultiCam service is not started when called.
- To check when the service is started, the application has to call the McOpenDriver function in a loop until the function returns MC_OK.
- Do not call McOpenDriver:
  - inside the constructor of a global or static object;
  - inside the DLL entry point (DllMain).

See Also

Driver Connection | McCloseDriver

2.2. Instance Management

McCreate

Creates an instance for a MultiCam object according to a model referred to by its handle.

```
[C]
MCSTATUS McCreate(
  MCHANDLE Model,
  PMCHANDLE Instance
);
```

Parameters

Model

When creating a Channel, use an handle designating the connector structure.
When creating a Surface, use MC_DEFAULT_SURFACE_HANDLE.

Instance

Pointer to the newly created instance.

See Also

Instance Management | McCreateNm | McDelete

McCreateNm

Creates an instance for a MultiCam object according to a model referred to by its name.

```
[C]
MCSTATUS McCreateNm(
  PCHAR ModelName,
  PMCHANDLE Instance
);
```

Parameters

ModelName

When creating a Channel, use a string representing the connector structure.
When creating a Surface, use McCreate.

Instance

Pointer to the newly created instance.

See Also

Instance Management | McCreate | McDelete

McDelete

Deletes the instance of a MultiCam object.

```
[C]
MCSTATUS McDelete(
  MCHANDLE Instance
);
```

Parameters

Instance

Handle of the instance to delete.

See Also

Instance Management | McCreate | McCreateNm

2.3. Parameters Management

McGetParamFloat

Returns the current value of a MultiCam parameter as an floating-point variable. The parameter is referred to by-identifier, and is preferably of the floating-point type.

```
[C]
MCSTATUS McGetParamFloat(
  MCHANDLE Instance,
  MCPARAMID Param,
  PFLOAT64 ValueFloat
);
```

Parameters

Instance

Handle of the instance of the parameter to read.

Param

Identifier of the parameter to read.

ValueInt

Pointer to the floating-point variable that receives the parameter value.

Remarks

If the MultiCam parameter is not of the floating-point type, a type conversion is performed.

See Also

Parameters Management

McGetParamNmFloat

Returns the current value of a MultiCam parameter as an floating-point variable. The parameter is referred to by-name, and is preferably of the floating-point type.

```
[C]
MCSTATUS McGetParamNmFloat(
  MCHANDLE Instance,
  PCHAR ParamName,
  PFLOAT64 ValueFloat
);
```

Parameters

Instance

Handle of the instance of the parameter to read.

ParamName

Pointer to a string containing the name of the parameter to read.

ValueFloat

Pointer to the floating-point variable that receives the parameter value.

Remarks

If the MultiCam parameter is not of the floating-point type, a type conversion is performed.

See Also

Parameters Management

McGetParamInst

Returns the current value of a MultiCam parameter as an instance variable. The parameter is referred to by-identifier, and is of the instance type.

```
[C]
MCSTATUS McGetParamInst(
  MCHANDLE Instance,
  MCPARAMID Param,
  PMCHANDLE ValueInst
);
```

Parameters

Instance

Handle of the instance of the parameter to read.

Param

Identifier of the parameter to read.

ValueInst

Pointer to the instance variable that receives the parameter value.

Example

The following code gets the channel-class select-level instance parameter Cluster.

```
//Declaring a MCHANDLE variable
MCHANDLE MySurface;
```

```
//Getting the parameter using the by-ident method
Status = McGetParamInst(ObjectHandle, MC_Cluster, &MySurface);
```

See Also

By-Name vs. By-Ident Access
McGetParamNmInst

McGetParamNmInst

Returns the current value of a MultiCam parameter as an instance variable. The parameter is referred to by-name, and is of the instance type.

```
[C]
MCSTATUS McGetParamNmInst(
  MCHANDLE Instance,
  PCHAR ParamName,
  PMCHANDLE ValueInst
);
```

Parameters

Instance

Handle of the instance of the parameter to read.

ParamName

Pointer to a string containing the name of the parameter to read.

ValueInst

Pointer to the instance variable that receives the parameter value.

Example

The following code gets the channel-class select-level instance parameter Cluster.

```
//Declaring a MCHANDLE variable
MCHANDLE MySurface;
//Getting the parameter using the by-name method
Status = McGetParamNmInst(ObjectHandle, "Cluster", &MySurface);
```

See Also

By-Name vs. By-Ident Access
McGetParamInst

McGetParamInt

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

```
[C]
MCSTATUS McGetParamInt(
  MCHANDLE Instance,
  MCPARAMID Param,
  PINT32 ValueInt
);
```

Parameters

Instance

Handle of the instance of the parameter to read.

Param

Identifier of the parameter to read.

ValueInt

Pointer to the integer variable that receives the parameter value.

Remarks

If the MultiCam parameter is not of the integer type, a type conversion is performed.

Example

The following code gets the channel-class expert-level integer parameter Elapsed_Fr.

```
//Declaring an integer variable
INT32 MyCount;
//Getting the parameter using the by-ident method
Status = McGetParamInt(ObjectHandle, MC_Elapsed_Fr, &MyCount);
```

See Also

By-Name vs. By-Ident Access
McGetParamNmInt

McGetParamNmInt

Returns 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.

```
[C]
MCSTATUS McGetParamNmInt(
  MCHANDLE Instance,
  PCHAR ParamName,
  PINT32 ValueInt
);
```

Parameters

Instance

Handle of the instance of the parameter to read.

ParamName

Pointer to a string containing the name of the parameter to read.

ValueInt

Pointer to the integer variable that receives the parameter value.

Remarks

If the MultiCam parameter is not of the integer type, a type conversion is performed.

Example

```
The following code gets the channel-class expert-level integer parameter Elapsed_Fr.
//Declaring an integer variable
INT32 MyCount;
//Getting the parameter using the by-name method
Status = McGetParamNmInt(ObjectHandle, "Elapsed_Fr", &MyCount);
```

See Also

By-Name vs. By-Ident Access
McGetParamInt

McGetParamInt64

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

```
[C]
MCSTATUS McGetParamInt64(
  MCHANDLE Instance,
  MCPARAMID Param,
  PINT64ValueInt64
);
```

Parameters

Instance

Handle of the instance of the parameter to read.

Param

Identifier of the parameter to read.

ValueInt64

Pointer to the integer variable that receives the parameter value.

Remarks

If the MultiCam parameter is not of the integer type, a type conversion is performed.

See Also

Parameters Management

McGetParamNmInt64

Returns 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.

```
[C]
MCSTATUS McGetParamNmInt64(
  MCHANDLE Instance,
  PCHAR ParamName,
  PINT64ValueInt64
);
```

Parameters

Instance

Handle of the instance of the parameter to read.

ParamName

Pointer to a string containing the name of the parameter to read.

ValueInt64

Pointer to the integer variable that receives the parameter value.

Remarks

If the MultiCam parameter is not of the integer type, a type conversion is performed.

See Also

Parameters Management

McGetParamPtr

Returns the current value of a MultiCam parameter as a string variable. The parameter is referred to by-identifier.

```
[C]
MCSTATUS McGetParamPtr(
  MCHANDLE Instance,
  MCPARAMID Param,
  PVOID* ValuePtr
);
```

Parameters

Instance

Handle of the instance of the parameter to read.

Param

Identifier of the parameter to read.

ValuePtr

Pointer to the parameter value.

See Also

Parameters Management

McGetParamNmPtr

Returns the current value of a MultiCam parameter as a string variable. The parameter is referred to by-name.

```
[C]
MCSTATUS McGetParamNmPtr(
  MCHANDLE Instance,
  PCHAR ParamName,
  PVOID* ValuePtr
);
```

Parameters

Instance

Handle of the instance of the parameter to read.

ParamName

Pointer to a string containing the name of the parameter to read.

ValuePtr

Pointer to the parameter value.

See Also

Parameters Management

McGetParamStr

Returns the current value of a MultiCam parameter as a string variable. The parameter is referred to by-identifier, and is preferably of the string or enumerated type.

```
[C]
MCSTATUS McGetParamStr(
  MCHANDLE Instance,
  MCPARAMID Param,
  PCHAR ValueStr,
  UINT32 MaxLength
);
```

Parameters

Instance

Handle of the instance of the parameter to read.

Param

Identifier of the parameter to read.

ValueStr

Pointer to the string variable that receives the parameter value.

MaxLength

Maximum number of characters in the read string.

Example

The following code gets the channel-class select-level string parameter BoardIdentifier. The last argument is the size of the buffer where the returned string is to be hosted. It should be large enough to accommodate any possible returned string parameter.

```
//Declaring a string variable
CHAR MyBoard[32];
//Getting the parameter using the by-name method
Status = McGetParamNmStr(ObjectHandle, "BoardIdentifier", &MyBoard, 32);
```

Remarks

If the MultiCam parameter is not of the string type, a type conversion is performed.

See Also

By-Name vs. By-Ident Access
McGetParamNmStr

McGetParamNmStr

Returns 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.

```
[C]
MCSTATUS McGetParamNmStr(
  MCHANDLE Instance,
  PCHAR ParamName,
  PCHAR ValueStr,
  UINT32 MaxLength
);
```

Parameters

Instance

Handle of the instance of the parameter to read.

ParamName

Pointer to a string containing the name of the parameter to read.

ValueStr

Pointer to the string variable that receives the parameter value.

MaxLength

Maximum number of characters in the read string.

Example

The following code gets the channel-class select-level string parameter BoardIdentifier. The last argument is the size of the buffer where the returned string is to be hosted. It should be large enough to accommodate any possible returned string parameter.

```
//Declaring a string variable
CHAR MyBoard[32];
//Getting the parameter using the by-name method
Status = McGetParamNmStr(ObjectHandle, "BoardIdentifier", &MyBoard, 32);
```

Remarks

If the MultiCam parameter is not of the string type, a type conversion is performed.

See Also

By-Name vs. By-Ident Access
McGetParamStr

McSetParamFloat

Assigns a floating-point variable to a MultiCam parameter. The parameter is referred to by-identifier, and is preferably of the floating-point type.

```
[C]
MCSTATUS McSetParamFloat(
  MCHANDLE Instance,
  MCPARAMID Param,
  FLOAT64 ValueFloat
);
```

Parameters

Instance

Handle of the instance of the parameter to configure.

Param

Identifier of the parameter to configure.

ValueFloat

Floating-point value assigned to the parameter.

Remarks

If the MultiCam parameter is not of the floating-point type, a type conversion is performed.

See Also

Parameters Management

McSetParamNmFloat

Assigns a floating-point variable to a MultiCam parameter. The parameter is referred to by-name, and is preferably of the floating-point type.

```
[C]
MCSTATUS McSetParamNmFloat(
  MCHANDLE Instance,
  PCHAR ParamName,
  FLOAT64 ValueFloat
);
```

Parameters

Instance

Handle of the instance of the parameter to configure.

ParamName

Pointer to a string containing the name of the parameter to configure.

ValueFloat

Floating-point value assigned to the parameter.

Remarks

If the MultiCam parameter is not of the floating-point type, a type conversion is performed.

See Also

Parameters Management

McSetParamInst

Assigns an instance variable to a MultiCam parameter. The parameter is referred to by-identifier, and is of the instance type.

```
[C]
MCSTATUS McSetParamInst(
  MCHANDLE Instance,
  MCPARAMID Param,
  MCHANDLE ValueInst
);
```

Parameters

Instance

Handle of the instance of the parameter to configure.

Param

Identifier of the parameter to configure.

ValueInst

Handle of the instance assigned to the parameter.

Example

The following code sets the channel-class select-level instance parameter Cluster to the value SurfaceHandle, a handle to a surface that has to be created beforehand.

```
//Setting the parameter using the by-ident method
Status = McSetParamInst(ChannelHandle, MC_Cluster, SurfaceHandle);
```

See Also

By-Name vs. By-Ident Access

McSetParamNmInst

McSetParamNmInst

Assigns an instance variable to a MultiCam parameter. The parameter is referred to by-name, and is of the instance type.

```
[C]
MCSTATUS McSetParamNmInst(
  MCHANDLE Instance,
  PCHAR ParamName,
  MCHANDLE ValueInst
);
```

Parameters

Instance

Handle of the instance of the parameter to configure.

ParamName

Pointer to a string containing the name of the parameter to configure.

ValueInst

Handle of the instance assigned to the parameter.

Example

The following code sets the channel-class select-level instance parameter Cluster to the value SurfaceHandle, a handle to a surface that has to be created beforehand.

```
//Setting the parameter using the by-name method
Status = McSetParamNmInst(ChannelHandle, "Cluster", SurfaceHandle);
```

See Also

By-Name vs. By-Ident Access
McSetParamInst

McSetParamInt

Assigns an integer variable to a MultiCam parameter. The parameter is referred to by-identifier, and is preferably of the integer or enumerated type.

```
[C]
MCSTATUS McSetParamInt(
  MCHANDLE Instance,
  MCPARAMID Param,
  INT32 ValueInt
);
```

Parameters

Instance

Handle of the instance of the parameter to configure.

Param

Identifier of the parameter to configure.

ValueInt

Integer value assigned to the parameter.

Remarks

If the MultiCam parameter is not of the integer type, a type conversion is performed.

Example

The following code sets the channel-class adjust-level integer parameter SeqLength_Fr to the value 100.

```
//Setting the parameter using the by-ident method
Status = McSetParamInt(ObjectHandle, MC_SeqLength_Fr, 100);
```

See Also

By-Name vs. By-Ident Access
McSetParamNmInt

McSetParamNmInt

Assigns an integer variable to a MultiCam parameter. The parameter is referred to by-name, and is preferably of the integer or enumerated type.

```
[C]
MCSTATUS McSetParamNmInt(
  MCHANDLE Instance,
  PCHAR ParamName,
  INT32 ValueInt
);
```

Parameters

Instance

Handle of the instance of the parameter to configure.

ParamName

Pointer to a string containing the name of the parameter to configure.

ValueInt

Integer value assigned to the parameter.

Remarks

If the MultiCam parameter is not of the integer type, a type conversion is performed.

Example

The following code sets the channel-class adjust-level integer parameter SeqLength_Fr to the value 100.

```
//Setting the parameter using the by-name method
Status = McSetParamNmInt(ObjectHandle, "SeqLength_Fr", 100);
```

See Also

By-Name vs. By-Ident Access
McSetParamInt

McSetParamInt64

Assigns an integer variable to a MultiCam parameter. The parameter is referred to by-identifier, and is preferably of the integer or enumerated type.

```
[C]
MCSTATUS McSetParamInt64(
  MCHANDLE Instance,
  MCPARAMID Param,
  INT64 ValueInt64
);
```

Parameters

Instance

Handle of the instance of the parameter to configure.

Param

Identifier of the parameter to configure.

ValueInt64

Integer value assigned to the parameter.

Remarks

If the MultiCam parameter is not of the integer type, a type conversion is performed.

See Also

Parameters Management

McSetParamNmInt64

Assigns an integer variable to a MultiCam parameter. The parameter is referred to by-name, and is preferably of the integer or enumerated type.

```
[C]
MCSTATUS McSetParamNmInt64(
  MCHANDLE Instance,
  PCHAR ParamName,
  INT64 ValueInt64
);
```

Parameters

Instance

Handle of the instance of the parameter to configure.

ParamName

Pointer to a string containing the name of the parameter to configure.

ValueInt64

Integer value assigned to the parameter.

Remarks

If the MultiCam parameter is not of the integer type, a type conversion is performed.

See Also

Parameters Management

McSetParamPtr

Assigns an string variable to a MultiCam parameter. The parameter is referred to by-identifier.

```
[C]
MCSTATUS McSetParamPtr(
  MCHANDLE Instance,
  MCPARAMID Param,
  PVOID ValuePtr
);
```

Parameters

Instance

Handle of the instance of the parameter to configure.

Param

Identifier of the parameter to configure.

ValuePtr

Pointer to the parameter.

Remarks

If the MultiCam parameter is not of the string type, a type conversion is performed.

See Also

Parameters Management

McSetParamNmPtr

Assigns a string variable to a MultiCam parameter. The parameter is referred to by-name.

```
[C]
MCSTATUS McSetParamNmPtr(
  MCHANDLE Instance,
  PCHAR ParamName,
PVOID ValuePtr
);
```

Parameters

Instance

Handle of the instance of the parameter to configure.

ParamName

Pointer to a string containing the name of the parameter to configure.

ValuePtr

Pointer to the parameter.

Remarks

If the MultiCam parameter is not of the string type, a type conversion is performed.

See Also

Parameters Management

McSetParamStr

Assigns an string variable to a MultiCam parameter. The parameter is referred to by-identifier, and is preferably of the string or enumerated type.

```
[C]
MCSTATUS McSetParamStr(
  MCHANDLE Instance,
  MCPARAMID Param,
  PCHAR ValueStr
);
```

Parameters

Instance

Handle of the instance of the parameter to configure.

Param

Identifier of the parameter to configure.

ValueStr

Pointer to the string assigned to the parameter.

Remarks

If the MultiCam parameter is not of the string type, a type conversion is performed.

Example

The following code sets the channel-class select-level string parameter CamFile to the value CAM2000-I200SA.

```
//Setting the parameter using the by-ident method
Status = McSetParamStr(ObjectHandle, MC_CamFile, "CAM2000-I200SA");
```

See Also

By-Name vs. By-Ident Access
McSetParamNmStr

McSetParamNmStr

Assigns a string variable to a MultiCam parameter. The parameter is referred to by-name, and is preferably of the string or enumerated type.

```
[C]
MCSTATUS McSetParamNmStr(
  MCHANDLE Instance,
  PCHAR ParamName,
  PCHAR ValueStr
);
```

Parameters

Instance

Handle of the instance of the parameter to configure.

ParamName

Pointer to a string containing the name of the parameter to configure.

ValueStr

Pointer to the string assigned to the parameter.

Remarks

If the MultiCam parameter is not of the string type, a type conversion is performed.

Example

The following code sets the channel-class select-level string parameter CamFile to the value CAM2000-I200SA.

```
//Setting the parameter using the by-name method
Status = McSetParamNmStr(ObjectHandle, "CamFile", "CAM2000-I200SA");
```

See Also

By-Name vs. By-Ident Access
McSetParamStr

2.4. Signaling

McGetSignalInfo

Returns the information structure associated with the last occurrence of a MultiCam signal.

```
[C]
MCSTATUS McGetSignalInfo(
  MCHANDLE Instance,
  MCSIGNAL Signal,
  PMCSIGNALINFO SignalInformation
);
```

Parameters

Instance

Handle of the channel instance that generates the signal.

Signal

Identifier of the signal.

SignalInformation

Signal information structure pointer. The structure is updated with the signal information when the function completes successfully.

See Also

Signaling

McRegisterCallback

Registers the callback function to a channel instance.

```
[C]
MCSTATUS McRegisterCallback(
  MCHANDLE Instance,
  PMCCALLBACK CallbackFunction,
  PVOID Context
);
```

Parameters

Instance

Handle of the channel instance of the callback function to register.

CallbackFunction

Pointer to the user-supplied callback function.
To unregister a function, use NULL.

Context

Argument to pass to the callback function.

See Also

Signaling

McWaitSignal

Allows a thread to wait for a specific MultiCam signal occurrence.

```
[C]
MCSTATUS McWaitSignal(
  MCHANDLE Instance,
  MCSIGNAL Signal,
  UINT32 TimeOut,
  PMCSIGNALINFO SignalInformation
);
```

Parameters

Instance

Handle of the channel instance that generates the signal.

Signal

Identifier of the signal to be waited for.

Timeout

Timeout duration expressed in milliseconds.
To disable the timeout, use INFINITE.

SignalInformation

Signal information structure pointer. The structure is updated with the signal information when the function completes successfully.

See Also

Signaling

2.5. Pixel Format Conversion

McConvertSurface

Handles software conversion between various pixel formats:

- Bayer pixel formats to RGB pixel formats;
- packed pixel formats (e.g. Y10P) to unpacked pixel formats;
- RGB to BGR or BGR to RGB pixel formats;
- planar to non-planar or non-planar to planar RGB pixel formats;
- image depth conversion.

```
[C]
MCSTATUS McConvertSurface(
  MCHANDLE InputInstance,
  MCHANDLE OutputInstance
);
```

Parameters

InputInstance

Handle of the input surface instance containing the image buffer to convert.

OutputInstance

Handle of the output surface instance which will hold the converted image buffer.

See Also

Pixel Format Conversion

3. Enumerations

3.1. MCPARAMID

Refer to Parameters Reference for the parameters identifiers.

3.2. MCSTATUS

Signed 32-bit integer.

```
typedef int MCSTATUS;
```

To get the error codes returned by MultiCam Functions, see MultiCam Error Codes.

4. Exceptions Management

4.1. API Errors

MultiCam errors are managed either with a MultiCam error code or a Windows exception, and with or without an error dialog box.

The ErrorHandling parameter sets the error management behavior and has 4 possible values:

NONE, the function returns a MultiCam error code.

- All API functions have the type MCSTATUS and return an integer value.
- If no error occurs, the function returns the value MC_OK or 0.
- If an error occurs, the function returns a negative value as defined in MCSTATUS.

MSG, the function displays a dialog box then returns a MultiCam error code. *(Windows OS only)*

- All API functions have the type MCSTATUS and return an integer value.
- If no error occurs, the function returns the value MC_OK or 0.
- If an error occurs, a dialog box is displayed with 3 buttons:
  - OK,the function returns a negative value as defined in MCSTATUS and passes this error value to the application.
  - ABORT terminates the application. All MultiCam resources are cleaned up but the application resources.
  - IGNORE forces the function to return MC_OK to the application, allowing the program to ignore the error.

EXCEPTION, the function issues a Windows exception. *(Windows OS only)*

- If an error occurs, the function generates a WIN32 structured exception with an error code as defined in MCSTATUS.
- The exception handling is compiler dependent. Refer to the Windows SDK and your compiler documentation for more information.

MSGEXCEPTION, the function displays a dialog box then issues a Windows exception. *(Windows OS only)*

- If an error occurs, a dialog box is displayed with 3 buttons:
  - OK, the function generates a WIN32 structured exception with an error code as defined in MCSTATUS.
  - ABORT terminates the application. All MultiCam resources are cleaned up but the application resources.

- IGNORE forces the function to return MC_OK to the application without generating an exception, allowing the program to ignore the error.

- The exception handling is compiler dependent. Refer to the Windows SDK and your compiler documentation for more information.

4.2. Events Signaling

MultiCam driver generates signals representing events issued by a channel. These signals enable interaction between the image acquisition process and the application.

Refer to the signaling documentation in the MultiCam User Guide.

5. Appendix

5.1. C Predefined Types

The following data types are used in the MultiCam C API. They are predefined using typedef.

C predefined types

Predefined type	Description
PVOID	Pointer to a void. **typedef void *PVOID;**
INT32	Signed 32-bit integer. **typedef signed int INT32;**
PINT32	Pointer to a signed 32-bit integer. **typedef signed int *PINT32;**
UINT32	Unsigned 32-bit integer. **typedef unsigned int UINT32;**
INT64	Signed 64-bit integer. **typedef signed int INT64;**
PINT64	Pointer to a signed 64-bit integer. **typedef signed int *PINT64;**
FLOAT64	Double precision 64-bit floating point. **typedef double FLOAT64;**
PFLOAT64	Pointer to a double precision 64-bit floating point. **typedef double *PFLOAT64;**
PCHAR	Pointer to a character. **typedef char *PCHAR;**
PCCHAR	Pointer to a constant character. **typedef const char *PCCHAR;**
MCHANDLE	Unsigned 32-bit integer. **typedef UINT32 MCHANDLE;**
PMCHANDLE	Pointer to an unsigned 32-bit integer. **typedef UINT32 *PMCHANDLE;**
MCSTATUS	Signed 32-bit integer. **typedef int MCSTATUS;**
MCPARAMID	Unsigned 32-bit integer. **typedef UINT32 MCPARAMID;**

Predefined type	Description
MCSIGNAL	Signed 32-bit integer. **typedef int MCSTATUS;**
PMCSIGNALINFO	Pointer to a structure containing a MultiCam signal information. **struct** **{** **PVOID Context;** **MCHANDLE Instance;** **MCSIGNAL Signal;** **UINT32 SignalInfo;** **UINT32 SignalContext;** **} *PMCSIGNALINFO;**
PMCCALLBACK	**typedef struct _MC_CALLBACK_INFO** **{** **PVOID Context;** **MCHANDLE Instance;** **MCSIGNAL Signal;** **UINT32 SignalInfo;** **UINT32 SignalContext;** **} MCSIGNALINFO, *PMCSIGNALINFO, *PMCCALLBACKINFO, MCCALLBACKINFO;** **typedef void (MCAPI *PMCCALLBACK)(PMCSIGNALINFO CbInfo);** **typedef void (MCAPI *PMCCALLBACKEX)(PVOID Context);**