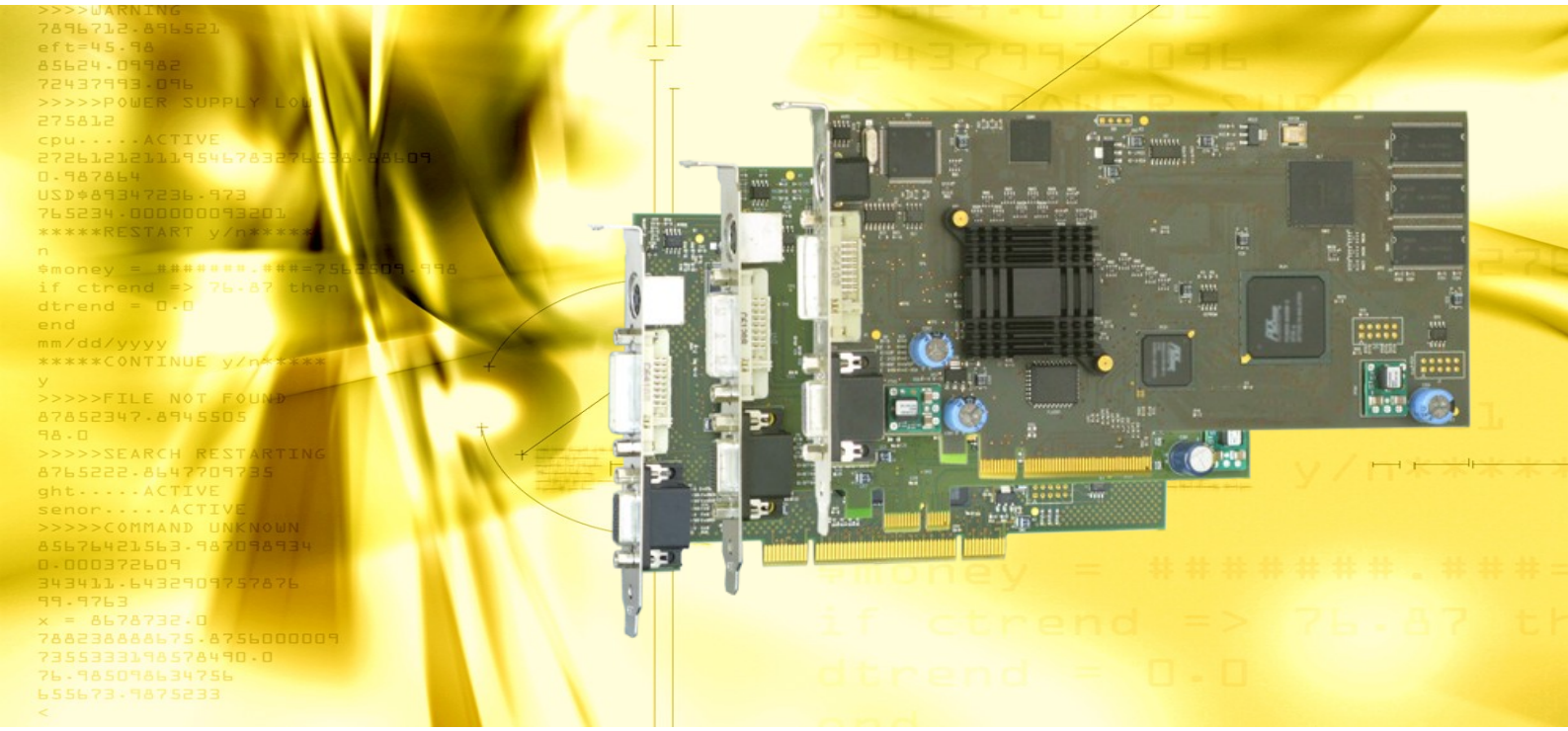
UFG-05 Family



Programmers Reference Manual

UNIGRAF

Copyright

Copyright © 1991-2009 Unigraf Oy. All rights reserved.

This document is protected with international copyright laws and must not copied without written permission. Information provided in this document is confidential and must not be shared to third parties without permission.

Notice

The information in this manual has been verified on the date of issue. The authors reserve rights to make any changes to this product and revise the information without obligation to notify any person about such revisions or changes.

Edition

Title	UFG-05 Programmers Reference Manual
Document ID	TJ2800
Issue date	November 2009

Company information

Unigraf Oy

Ruukintie 3
FI-02330 ESPOO
Finland, EU

Phone. +358 9 589 550
Fax. +358 9 802 6699

e-mail: sales@unigraf.fi
web: http://www.unigraf.fi

Trademarks

UFG is a trademark of Unigraf Oy.

All other trademarks are properties of their respective owners.

Table of Contents

1. General....5
1.1. About This Document....5
1.2. Acronyms and Abbreviations....5

2. Introduction to UFG-05 control mechanisms....6
2.1. Available Control Mechanisms....6
2.1.1. Driver Property Pages....6
2.1.2. Command line tool....7
2.1.3. Configuration DLL....7
2.1.4. Capture DLL....7
2.1.5. Extended Message Interface (EMI)....8
2.1.6. DirectShow Capture....8

3. UFG05Conf.dll....9
3.1. General Requirements....9
3.2. Accessing the UFG05Conf.dll....9
3.3. Interface Class Function Reference....10
3.3.1. UFG05_Conf....10
3.3.2. ~UFG05_Conf....11
3.3.3. Update....11
3.3.4. Commit....12
3.3.5. SetAutoUpdates....12
3.3.6. SetAutoCommits....13
3.3.7. GetValue....14
3.3.8. SetValue....15
3.3.9. SaveConfig....16
3.3.10. LoadConfig....17
3.3.11. GetErrorDescription....18
3.4. DLL API....19
3.4.1. GetDLLVersion....19
3.4.2. Open....20
3.4.3. Close....21
3.4.4. Update....22
3.4.5. Commit....23
3.4.6. AutoUpdates....24
3.4.7. AutoCommits....25
3.4.8. GetValue....26
3.4.9. SetValue....27
3.4.10. SaveConfig....28
3.4.11. LoadConfig....29
3.4.12. GetErrorDescription....30

4. Extended Message Interface....31
4.1. Requirements....31
4.2. Accessing EMI....31
4.3. EMI Function Reference....35
4.3.1. ExtendedInterfaceMsgEx....35
4.3.2. SetAutoCommit....36
4.3.3. Commit....37
4.3.4. Update....37
4.3.5. GetConfig....38
4.3.6. SetConfig....39
4.4. EMI Message Types....40

5. DirectShow capture....41

6. Error-exception class....51
6.1. Public Members....51
6.1.1. Class Constructor....51
6.1.2. Public Variables....51
6.2. Using ErrorException Class....52
6.2.1. Throwing ErrorException....52
6.2.2. Catching ErrorException....52

7. UFG-05 properties....53

1. GENERAL

1.1. About This Document

This document is a part of the UFG-05 API Software Development Kit supplied with the *UFG-05 Software Package*.

The UFG-05 SDK was developed to make the UFG-05 frame grabbers easier to use in client software. This document describes all of the Application Programming Interfaces offered by the UFG-05 SDK.

For controlling other UFG families than the UFG-05, see the particular documentation or contact Unigraf for more information.

1.2. Acronyms and Abbreviations

UFG	Unigraf Frame Grabber
DLL	Dynamic Link Library
API	Application Programming Interface
SDK	Software Development Kit
COM	Component Object Model
GUI	Graphical User Interface
MSDN	Microsoft Developers' Network
VB	Visual Basic
.NET	A framework developed by Microsoft.
RGB	Red-Green-Blue (TrueColor) channel mode for color definition.
EMI	Extended Message Interface – UFG-05 capture card's native control interface.

2. INTRODUCTION TO UFG-05 CONTROL MECHANISMS

2.1. Available Control Mechanisms

The UFG-05 frame grabber can be controlled with several different ways, starting from a simple GUI and ending to almost direct control of the card's hardware.

- The main control interface is the filter property pages. They are probably the most common way to control a frame grabber. These frame grabber properties are usually directly available from your favorite video capture application. Please find the explanation of the functions in UFG-05 User manual.
- The SDK includes a command-line utility to control basic video properties like saturation, contrast and brightness. The command line utility is also available as a DLL with it's own interface class unit for C++ programmers.
- The SDK includes also a DLL library with a simple C++ interface class unit for C++ programmers that enable simple, but efficient video capture.
- COM object, Extended Message Interface provides same configuration library as the DLL.
- The default interface for all DirectShow applications is the DirectShow source filter. It must be used by all video capture programs that can capture video using UFG-05.

2.1.1. Driver Property Pages

The driver's property pages are the most common way to access UFG-05 features. The pages can usually be accessed from any DirectShow video capture program, and their use is the recommended method in generic video capture solutions.

More advanced capture programs can make most of the necessary setup for you and it may not be necessary to access the property pages at all. When using an advanced capture software, do not check the override settings on the UFG-05 property page.

Notes

Accessing the UFG-05 control panel varies from program to program. Please, refer to the user manual of the capture software to determine how to access the frame grabber's property pages.

Some capture programs might show two property pages for UFG-05: "Filter properties" and "Pin properties". In these cases, we recommend that you leave the Pin properties untouched, and use the "Filter properties" and capture program's options to control the UFG-05 capture board.

Some video capture programs also fail to apply the settings. In these cases, it may be necessary to close the capture program and re-start it to make the changes effective.

2.1.2. Command line tool

The command line configuration tool can be used to configure video image parameters in case the used capture program does not allow access to the driver's property pages, or using the property pages is not wanted.

2.1.3. Configuration DLL

UFG05Conf.dll can be used to control video image parameters much like the command line tool. UFG05Conf.dll can easily be used from a custom capture program:

To use UFG05Conf.dll:

1. Make sure the DLL is located in the same folder as your application.
2. Include the provided interface class unit to your project, create an instance of it and all the functionality it offers are available. The interface class can be readily used with Embarcadero's C++ Builder (Or RAD Studio), and with Microsoft Visual C++. Other C++ compilers may work, but are untested.

The configuration DLL is a "helper" DLL. All the functionality it offers are available through the Extended message interface. The DLL is provided only to make the extended message interface's functionality simpler to use.

For further information about the configuration DLL, refer to section 3 “UFG05Conf.dll” in this manual.

Note: This configuration DLL provides a sub-set of features if compared to the capture DLL. If you use the capture DLL, there is no need to use this configuration DLL.

2.1.4. Capture DLL

The Capture DLL is a "helper" DLL. All the functionality it offers are available through a DirectShow video capture graph, and the extended message interface. The DLL is provided only to make the capture and control features available to a wider range of developers by simplifying video and image capturing.

UFG05Cap.dll can be used to capture video or single frames with a custom software. The DLL uses DirectShow to perform the actual video capturing, and provides a greatly simplified API to capture video frames compared to DirectShow. An interface class unit is provided for the DLL. The interface class unit can be readily used with Embarcadero's C++ Builder (Or RAD studio), and with Microsoft Visual C++. Other C++ compilers may work, but are untested.

For further information about the capture DLL, refer to section 4 in this manual.

Note: The capture DLL provides all features offered by Configuration DLL. The Capture DLL is a super-set of the configuration DLL. If you use the capture DLL, there is no need to use configuration DLL at it's side.

2.1.5. Extended Message Interface (EMI)

The extended message interface provides the same functionality as the configuration DLL, but in certain situations it might be necessary to use this interface instead of the DLL.

The extended message interface is a part of the DirectShow source filter of the UFG-05 frame grabber. To access the extended message interface, first open the UFG-05 source filter and then get the extended message interfaces pointer from the source filter.

For further information about the Extended message interface, refer to section 5 in this manual.

Note: The extended message interface is a part of the UFG-05 frame grabber driver package. No need to install or download anything to use it. Developers, however, will need the SDK in order to make software that uses the interface.

2.1.6. DirectShow Capture

All capture programs capable of using the UFG-05 frame grabber must use the DirectShow source filter to capture video. The source filter provides only the capture pin. Graphics data is in RGB32 format.

A native DirectShow capture is the most efficient method of capturing video with UFG-05 frame grabber, but it is also most the most complex one.

Note: If you write a custom DirectShow capture program, you can use the DirectShow features to control the capture resolution and frame rate. However, you need to un-check all of the override settings in the driver property pages of the user interface manually.

If you allow the user to access the source filters property page, please make sure that the filter object is destroyed and re-created after showing the page. Otherwise the settings will not be applied.

The UFG-05 DirectShow source filter is a part of the UFG-05 frame grabber driver package. There is no need to install or download anything to use it.

---

3. UFG05CONF.DLL

The UFG05Conf.dll is a library which contains the necessary functions to get and set various values that control the UFG-05 video grabber's function or indicate operational state.

For example: read the incoming video signals native resolution and refresh rate. If the video source is analog, basic image properties can be adjust, like saturation, contrast and brightness.

3.1. General Requirements

The provided examples and documentation are made for C++ compilers. The library can be used from standard C as well, but we strongly encourage use of a C++ compiler.

The UFG05Conf.dll interface class and library should be compatible with Embarcadero's C++ Builder and RAD Studio products, as well as Microsoft's Visual Studio.

3.2. Accessing the UFG05Conf.dll

In general there are three ways to use a DLL in a C++ application.

- Use the provided interface class. It should compile in many C++ compilers, but if there are problems one can always resort to other methods.
- Use the provided .LIB + .H file to attach the DLL to the project. This method is generally very simple, but it has some drawbacks. One these must be the uncertainty of which physical dll file is loaded. Program error handling messages can be cryptic to common users. On the positive side, this method allows more freedom in selection of the development tools.
- Use the DLL manually, by using LoadLibrary and GetProcAddress WinAPI functions to gain access to functions. In this case, one can use the Interface class as a reference.

Unigraf recommends option 1, it is simple to use and very safe. In this case the interface class takes care of finding the DLL, proper initializations and version handling. Outside, the interface class offers pretty much the same functions the DLL contains with some minor differences.

3.3. Interface Class Function Reference

This chapter describes the UFG05Conf.dll's interface class. To use the interface, class you need to add the files “UFG05_Conf_Interface.cpp”, “UFG05_Conf_Interface.h” and “UFG05_Types.h” to your C++ project.

Please note, that the interface class does not offer any of the configuration features in itself. Instead, it makes the configuration DLL easier to use. As a result, you need to have the “UFG05Conf.dll” in the same folder with your resulting executable file.

If you wish to use the DLL directly without the interface class, refer to section 3.4, which describes the DLL API and it's functions.

3.3.1. UFG05_Conf

Creates an instance of the UFG-05 configuration class.

Prototype

```
UFG05_Conf
(
     int SourceIndex
);
```

Synopsis

Creates an instance of the UFG05Conf.dll interface class. The constructor automatically loads the library, resolves the service functions it contains and opens the requested resource. The cards are identified with an index value.

Do not call this function directly. Instead use the *new* keyword.

Parameters

SourceIndex
An integer value identifying the card to open. First index value is zero.

Exceptions

This function may throw exceptions of type ErrorException, if it encounters errors from which it cannot recover. This function will throw exception if one of the following errors are encountered:

- “UFG05Conf.dll” is not found (Error 1)
- “UFG05Conf.dll” is of an unacceptable version. (Error 2)
- Cannot resolve required function(s). (Errors 3 and 4)
- Failed to open the requested UFG-05 resource. (Error 5)

See Also

Error-Exception Class, ~UFG05_Conf

3.3.2. ~UFG05_Conf

Destroys a previously created instance of UFG-05 configuration class

Prototype

```
~UFG05_Conf
(
);
```

Synopsis

This function closes the UFG-05 resource opened by the constructor function and releases the DLL library and other used resources.

Class destructor. Don't call this function directly. Instead use the *delete* keyword.

See Also

UFG05_Conf

3.3.3. Update

Update the local copy of UFG-05 properties

Prototype

```
int Update
(
);
```

Synopsis

Updates the local copy of the UFG-05 property storage directly from the hardware device.

Return

Zero if successful, or a negative error code.

See Also

SetAutoUpdate, Commit, SetAutoCommit

3.3.4. Commit

Commit the current property settings on the local copy to hardware

Prototype

```
int Commit
(
);
```

Synopsis

Commits the local copy of the UFG-05 property storage directly on the hardware.

Return

Zero if successful, or a negative error code.

See Also

SetAutoCommit, Update, SetAutoUpdate

3.3.5. SetAutoUpdates

Enables automatic updates of local property storage on GetValue calls

Prototype

```
int SetAutoUpdates
(
    bool Enable
);
```

Synopsis

Enables or disables automated update calls in GetValue function calls. The feature is enabled by default.

Parameters

Enable
True to enable or false to disable automatic updates.

Return

Zero if successful, or a negative error code.

See Also

Update, Commit, SetAutoCommits

3.3.6. SetAutoCommits

Enable automatic commits of local property storage on SetValue calls

Prototype

```
int SetAutoCommits
(
    bool Enable
);
```

Synopsis

Enable or disable automated commit calls in SetValue function calls. The feature is enabled by default.

Parameters

Enable
Indicates whether to enable or disable the feature

Return

Zero if successful, or a negative error code.

See Also

Commit, Update, SetAutoCommits

3.3.7. GetValue

Get a UFG-05 property value

Prototype

```
int GetValue
(
     int ValueIndex,
     int &Value
);
```

Synopsis

Get an UFG-05 property value identified by the ValueIndex. The value is read from the UFG-05 resource associated to the class instance.

Parameters

ValueIndex
: Identifies the UFG-05 Property to read. Please refer to “UFG-05 properties” for a list of valid ValueIndexes.

Value
: A pointer reference to an integer variable, which receives the value of the property.

Return

Negative error code, or zero if successful.

See Also

SetValue, GetErrorDescription

3.3.8. SetValue

Set UFG-05 property value

Prototype

```
int SetValue
(
     int ValueIndex,
     int NewValue
);
```

Synopsis

Set a new value to the property identified by ValueIndex. The value is set to the UFG-05 resource associated to the class instance.

Parameters

ValueIndex
Identifies the UFG-05 property to set. Please refer to "UFG-05 properties" for a list of valid ValueIndexes.

NewValue
Value to set to the given property.

Return

Negative error code, or zero if successful.

See Also

GetValue, GetErrorDescription

3.3.9. SaveConfig

Save UFG-05 configuration to a file.

Prototype

```
int SaveConfig
(
     char *ToFileName
);
```

Synopsis

Save the UFG-05 resource setup to a file on disk. The resulting configuration file is compatible with UFG-05Pre_Config command line tool. The function will overwrite any existing file without any warning.

Parameters

ToFileName
: Pointer to a NULL-terminated string, which contains the full path and filename of the file into which to save the configuration data.

Return

A negative error code, or zero if succesful.

See Also

LoadConfig, GetErrorDescription

3.3.10. LoadConfig

Load an UFG-05 configuration from a file

Prototype

```
int LoadConfig
(
     char *FromFileName
);
```

Synopsis

Loads an UFG-05 configuration from a file on disk. The function can load configuration files created with SaveConfig function and UFG04Pre_Config command line tool.

Parameters

FromFileName
: Pointer to a NULL terminated string, which contains the full path and filename of the file from which to load the new configuration.

Return

Negative error code, or zero if successful.

See Also

SaveConfig, GetErrorDescription

3.3.11. GetErrorDescription

Retrieves a human readable error message for given error code

Prototype

```
int GetErrorDescription
(
     int ErrorCode,
     char *Errorstring,
     int MaxLength
);
```

Synopsis

Call this function to retrieve a human readable error message that matches to the given error code. If the resulting string does not fit to the given string buffer, the function results an emptry string.

Parameters

ErrorCode
: An error code returned by a previous call to an UFG05_Conf class function.

ErrorString
: Pointer to a user allocated buffer to hold the resulting error description.

MaxLength
: Indicates the size of the allocated string buffer.

Return

Zero if successful, or positive value indicating the required size to hold error description text completely.

3.4. DLL API

This section describes the API inside the “UFG05Conf.dll” library.

Please refer to “UFG05_Conf_Interface.cpp” to see an example of how to gain access to the functions inside the DLL.

3.4.1. GetDLLVersion

Retrieves the version information of the DLL.

Prototype

```
int __stdcall GetDLLVersion
(
     int &Major,
     int &Minor,
     int &Rev
);
```

Synopsis

Retrieves the version information of the DLL library in order to verify compatibility with the host application. While it is not mandatory to call this function, checking the version information enables the host application to avoid compatibility issues before running into problems.

Major version should match exactly to expected version. The Major version usually does not change. Minor version must at least same as expected, or higher. The minor version is usually changed between releases, if the DLL's API has been extended but remains compatible with older host software. The Revision can be ingored when checking compatibility. Revision number is changed between various development stages.

Parameters

Major
: Reference pointer to an integer variable which receives the Major version number of the DLL.

Minor
: Reference pointer to an integer variable which receives the Minor version number of the DLL.

Rev
: Reference pointer to an integer variable which receives the Revision number of the DLL.

Return

Zero if successful, or a negative error code.

See Also

GetErrorDescription

3.4.2. Open

Open an UFG-05 resource.

Prototype

```
int __stdcall Open
(
    HANDLE &InterfaceHandle,
    int SourceIndex
);
```

Synopsis

Open an UFG-05 resource to access the frame grabber properties. The resulting handle must be released by calling Close before the DLL is unloaded.

Parameters

InterfaceHandle
: Reference pointer to a HANDLE variable, which will receive the opened interface handle.

SourceIndex
: Index of the resource to open.

Return

Zero if successful, or a negative error code.

See Also

Close, GetErrorDescription

3.4.3. Close

Close an UFG-05 resource previously opened with Open

Prototype

```
int __stdcall Close
(
     HANDLE &InterfaceHandle
);
```

Synopsis

Closes a previously opened UFG-05 resource. The Interface handle variable will be invalid after this function call.

Parameters

InterfaceHandle
Reference to a variable which contains the interface handle to be closed.

See Also

Open

3.4.4. Update

Update the local copy of UFG-05 properties

Prototype

```
int Update
(
    HANDLE InterfaceHandle
);
```

Synopsis

Updates the local copy of the UFG-05 property storage directly from the hardware device.

Parameters

InterfaceHandle
Handle of the UFG-05 resource in which the command is to execute.

Return

Zero if successful, or a negative error code.

See Also

SetAutoUpdate, Commit, SetAutoCommit

3.4.5. Commit

Commit the current property settings on the local copy to hardware

Prototype

```
int Commit
(
     HANDLE IntefaceHandle
);
```

Synopsis

Commits the local copy of the UFG-05 property storage directly on the hardware.

Parameters

InterfaceHandle
Handle of the UFG-05 resource in which the command is to execute.

Return

Zero if successful, or a negative error code.

See Also

SetAutoCommit, Update, SetAutoUpdate

3.4.6. AutoUpdates

Enables automatic updates of local property storage on GetValue calls

Prototype

```
int SetAutoUpdates
(
    HANDLE InterfaceHandle,
    int Setting
);
```

Synopsis

Enables or disable automated update calls in GetValue function calls. The feature is enabled by default.

Parameters

InterfaceHandle
Handle of the UFG-05 reosurce for which to set auto update feature's state.

Setting
Non-zero to enable, zero to disable feature.

Return

Zero if successful, or a negative error code.

See Also

Update, Commit, SetAutoCommits

3.4.7. AutoCommits

Enable automatic commits of local property storage on SetValue calls

Prototype

```
int SetAutoCommits
(
    HANDLE InterfaceHandle,
    int Setting
);
```

Synopsis

Enable or disable automated commit calls in SetValue function calls. The feature is enabled by default.

Parameters

InterfaceHandle
Handle of the UFG-05 resource for which to set auto commit feature's state.

Setting
Non-zero to enable, zero to disable feature.

Return

Zero if successful, or a negative error code.

See Also

Commit, Update, SetAutoCommits

3.4.8. GetValue

Read a property value from an open UFG-05 resource

Prototype

```
int __stdcall GetValue
(
    HANDLE InterfaceHandle,
    int ValueIndex,
    int &Value
);
```

Synopsis

Read the property value identified by ValueIndex from the UFG-05 resource identified by InterfaceHandle. If the function fails, the Value variable will be zeroed.

Parameters

InterfaceHandle
Handle of the UFG-05 resource from which to read the property.

ValueIndex
Identifies the value to read. Please refer to “UFG-05 properties” for a list of valid ValueIndexes.

Return

Zero if successful, or a negative error code.

See Also

SetValue, SaveConfig, LoadConfig, GetErrorDescription

3.4.9. SetValue

Set a new value to an UFG-05 property

Prototype

```
int __stdcall SetValue
(
    HANDLE InterfaceHandle,
    int ValueIndex,
    int NewValue
);
```

Synopsis

Write the property identified by ValueIndex in the UFG-05 resource identified by InterfaceHandle. Please note, that there is no guarantee that the writen value will have any immediate effect on the capture device.

Parameters

InterfaceHandle
Handle of the UFG-05 resource to which to write the property.

ValueIndex
Identifies the property to write. Please refer to “UFG-05 properties” for a list of valid ValueIndexes.

NewValue
The new value to write

Return

Zero if successful, or a negative error code.

See Also

GetValue, SaveConfig, LoadConfig, GetErrorDescription

3.4.10. SaveConfig

Save the UFG-05 resource configuration data to a file on disk.

Prototype

```
int __stdcall SaveConfig
(
    HANDLE InterfaceHandle,
    char *FileName
);
```

Synopsis

Saves the complete internal status of the UFG-05 resource identified by the InterfaceHandle to a file on disk. The resulting file will be compatible with the UFG05_CMD.exe command line tool. This function will overwrite the file if it exists. If the function fails, the state of the target file will be unknown.

Parameters

InterfaceHandle
: Handle of the UFG-05 resource whose configuration is to be saved.

FileName
: Pointer to a NULL terminated string, containing complete path and filename to which the configuration is to be saved.

Return

Zero if successful, or a negative error code.

See Also

LoadConfig, GetErrorDescription

3.4.11. LoadConfig

Load configuration of an UFG-05 resource from a file on disk

Prototype

```
int __stdcall LoadConfig
(
    HANDLE InterfaceHandle,
    char *FileName
);
```

Synopsis

Loads a previously saved UFG-05 resource configuration from a file on disk. This function can load configuration files saved by the UFG05_CMD.exe command line tool.

Parameters

InterfaceHandle
: Handle to the UFG-05 resource whose configuration is to be loaded

FileName
: Pointer to a NULL terminated string containing full path and filename of the file from which the configuration is loaded.

Return

Zero if successful, or a negative error code

See Also

SaveConfig, GetErrorDescription

3.4.12. GetErrorDescription

Retrieves a human readable error description for an error code

Prototype

```
int __stdcall GetErrorDescription
(
     int ErrorCode,
     char *ErrorString,
     int MaxLength
);
```

Synopsis

Retrieves a human readable error description for an error code. If the text will not fit into the given string buffer, the buffer is zeroed.

Parameters

ErrorCode
The error code for which to get the description

ErrorString
A User allocated string buffer into which the error description is copied.

MaxLenth
The allocated size of the ErrorString buffer

Return

Zero if successful, or a positive value indicating the required size of the string buffer.

4. EXTENDED MESSAGE INTERFACE

This section describes the Extended message Interface (EMI), which can be used to control the basic video parameters of an UFG-05 frame grabber.

The Extended Message Interface is very different if compared to the UFG05Cap.dll, or UFG05Conf.dll. Also, it works much close to hardware level. If your capture solution is using DirectShow, Unigraf recommends you use this interface to control the UFG-05 capture board's parameters directly.

4.1. Requirements

To successfully access and use the Extended Message Interface, you need the following tools and knowledge:

- UFG-05 Software Development kit
- Embarcadero Codegear C++ Builder or Microsoft Visual C++
- Basic knowledge of COM interfaces, and how to use them

4.2. Accessing EMI

Opening the EMI is not as simple as opening a single DLL. This chapter gives step-by-step instructions on how to open EMI for use. The instructions are for C++ Builder, but with minor changes other compilers should work.

Includes and Declarations

The rest of the code examples are going to use functions and variables that are defined below.

```
#include <Dshow.h>
#include "fg04iface.h"

IMoniker *pMoniker = NULL;
ICreateDevEnum *pDevEnum = NULL;
IEnumMoniker *pClassEnum = NULL;
IBaseFilter *pSrc = NULL;
IFG04KsproxySampleConfig = pConfig;
```

<*Continued*>

Initializing the COM library

If you are already using COM in your application the compiler may have already initialized COM for you. However, it is necessary to initialize COM library in all threads that use it.

```
HRESULT Result = CoInitialize(NULL);
if((Result != S_OK) && (Result != S_FALSE))
{
     // COM initialization failed - Handle error here
}
```

System video Capture Device Enumeration

First we need to create an instance of the ICreateDevEnum interface, to gain access to system device enumerator.

```
HRESULT Result = CoCreateInstance(
     CLSID_SystemDeviceEnum,
     NULL,
     CLSCTX_INPROC,
     IID_ICreateDevEnum,
     (void **) &pDevEnum);

if(FAILED(Result))
{
     // Failed to create instance - Handle error here
}
```

Next, we need to use the ICreateDevEnum interface to create an enumerator object for video capture devices. This is done with the CreateClassEnumerator function.

```
HRESULT Result = pDevEnum->CreateClassEnumerator(
     CLSID_VideoInputDeviceCategory,
     &pClassEnum,
     0);

if(FAILED(Result))
{
     // Failed to create device enumerator
     // Handle error here
}
```

If everything went as expected, we now have a device enumerator for the video input device category, and we start with the the device enumeration loop

<*Continued*>

Search for the UFG-05 Device

```
while((pClassEnum->Next(1, &pMoniker, NULL) == S_OK) &&
      (pConfig == NULL))
{
```

This loop searches the device list either until the list is completely searched, or we have found the device that we were looking for.

Next, we need to get an IPropertyBag interface in order to read (or write) properties of the device we got from the pClassEnum->Next(...) function. If the device does not implement IPropertyBag interface, move to next device.

```
IPropertyBag *Props;
HRESULT Result = pMoniker->BindToStorage(
     0,
     0,
     IID_IPropertyBag,
     (void **) &Props);

if(FAILED(Result))
{
     pMoniker->Release();
     pMoniker = NULL;
     continue;
}
```

In order to identify UFG-05, we now need to find out the device's friendly name and compare it with the friendly name of the UFG-05 frame grabber (“UFG-05 Channel”).

```
tagVARIANT FriendlyName;
FriendlyName.vt = VT_EMPTY;
char UFGname[] = “UFG-05 Channel”;

HRESULT Result = Props->Read(
     L”FriendlyName”,
     &FriendlyName,
     NULL);

Props->Release();

if((FAILED(Result)) ||
     (strcmp(FriendlyName.bstrVal, UFGName) != 0))
{
     pMoniker->Release();
     pMoniker = NULL;
     continue;
}
```

If we get to this point, it means that we have found an UFG-05 frame grabber.

<*Continued*>

Retrieve the EMI Instance

Now we need to use the IMoniker interface to retrieve the IBaseFilter interface of the UFG-05 and query the EMI from that interface.

```
HRESULT Result = pMoniker->BindToObject(
     0,
     0,
     IID_IBaseFilter,
     (void **) &pSrc);

pMoniker->Release();
pMoniker = NULL;

if(FAILED(Result)) continue;

HRESULT Result = pSrc->QueryInterface(
     __uuidof(IFG04KsproxySampleConfig),
     (void **) &pConfig);

pSrc->Release();
pSrc = NULL;

if(FAILED(Result)) pConfig = NULL;
```

Ending the Loop

At this point we just need to perform the necessary cleanup after the loop. If EMI was available, it's pointer is now set in pConfig.

```
} // End of while-loop

if(pDevEnum != NULL) pDevEnum->Release();
if(pClassEnum != NULL) pClassEnum->Release();
if(pSrc != NULL) pSrc->Release();
```

Releasing the EMI

When you are done with the EMI (pConfig), remember to release it. Calling it's Release function will cleanup all the remaining COM objects and deallocate all resources related to it.

```
pConfig->Release();
```

4.3. EMI Function Reference

Please notice, that Unigraf does not recommend using any of the functions which are not documented here. Furthermore, Unigraf cannot be held responsible if the use or missuse of any of the functions contained in the extended message interface cause loss of data or equipment failure. The “fg04iface.h” contains all definitions required to use EMI.

4.3.1. ExtendedInterfaceMsgEx

Send parameter related commands to the UFG-05 grabber board.

Prototype

```
HRESULT STDMETHODCALLTYPE ExtendedInterfasMsgEx
(
    long message,
    long lParam1,
    long *lParam2
);
```

Synopsis

Implements a message based command interface to the UFG-05 frame grabber. The function can enumerate parameter types, min and max values of parameters, parameter names and tooltips, set and get parameters and query if a parameter is supported.

Parameters

message
: Value indicating the message type. See EMI Message Types

lParam1
: Indicates the addressed parameter.

lParam2
: Pointer to a variable that contains value to set, or receives the query result.

Return

S_OK or E_FAIL – Use FAILED or SUCCEEDED macros to test.

See Also

SetAutoCommit, Commit, Update

4.3.2. SetAutoCommit

Enable or Disable the automatic commits

Prototype

```
HRESULT STDMETHODCALLTYPE SetAutoCommit
(
    long lParam1
);
```

Synopsis

Sets the state of the automatic commit feature. If set to “true”, all writen parameters are automatically commited on the UFG-05 frame grabber hardware without the need to call Commit(). If set to “false”, a number of parameter sets can be combined to be transferred to hardware with a single Commit() call.

Parameters

lParam
Set to 1 (true) or 0 (false) to control automatic commit feature.

Return

S_OK or E_FAIL – Use FAILED or SUCCEEDED macros to test.

See Also

Commit, Update

4.3.3. Commit

Commits all parameters set so far

Prototype

```
HRESULT STDMETHODCALLTYPE Commit
(
);
```

Synopsis

Commits all parameters set so far.

Return

S_OK or E_FAIL – Use FAILED or SUCCEEDED macros to test

See Also

AutoCommit, Update

4.3.4. Update

Updates the local parameter cache of the EMI

Prototype

```
HRESULT STDMETHODCALLTYPE Update
(
);
```

Synopsis

Updates the EMI internal parameter cache from the UFG-05 hardware. The cache is automatically updated when EMI is opened.

Return

S_OK or E_FAIL – Use FAILED or SUCCEEDED macros to test

See Also

SetAutoCommit, Commit

4.3.5. GetConfig

Gets the UFG-05 card configuration as GM1601_PARAMETERS structure.

Prototype

```
HRESULT STDMETHODCALLTYPE GetConfig
(
    GM1601_PARAMETERS *plRate
);
```

Synopsis

Gets the UFG-05 card configuration as GM1601_PARAMETERS structure.

Parameters

plRate

Pointer to GM1601_PARAMETERS structure, which receives the card configuration.

Return

S_OK or E_FAIL – Use FAILED or SUCCEEDED macros to test

See Also

SetConfig

4.3.6. SetConfig

Sets the UFG-05 card configuration with GM1601_PARAMETERS structure.

Prototype

```
HRESULT STDMETHODCALLTYPE SetConfig
(
    GM1601_PARAMETERS *lRate
);
```

Synopsis

Sets the UFG-05 card configuration with GM1601_PARAMETERS structure.

Parameters

lRate

Pointer to GM1601_PARAMETERS structure, which contains the card configuration to set.

Return

S_OK or E_FAIL – Use FAILED or SUCCEEDED macros to test

See Also

GetConfig

4.4. EMI Message Types

Currently supported message types are:

DXFG04_ENUM_PARAMETER_TYPE
Returns type of parameter indicated by lParam1, Possible results are:

DXFG04_PAR_TYPE_INTEGER - A single 32-bit integer.

DXFG04_PAR_TYPE_BOOLEAN - A boolean value, 0 or 1.

DXFG04_PAR_TYPE_BUTTON - A Strobe parameter.

DXFG04_PAR_TYPE_BUFFER - A byte buffer.

DXFG04_ENUM_PARAMETER_MIN
Returns minimum value for the indicated parameter.

DXFG04_ENUM_PARAMETER_MAX
Returns maximum value for the indicated parameter.

DXFG04_ENUM_PARAMETER_NAME
Returns name of the indicated parameter.

DXFG04_ENUM_PARAMETER_TOOLTIP
Returns tooltip for the indicated parameter.

DXFG04_SET_PARAMETER
Sets value of the indicated parameter. Notice that the setting may not be set into hardware until the Commit() function is called.

DXFG04_GET_PARAMETER
Gets value of the indicated parameter. Notice that in you need to call *Update()* to get new values from the grabber board. Otherwise, reading the parameter again will always result same value.

DXFG04_QUERY_PARAMETER_SUPPORTED
return true if indicated parameter is supported in hardware.

5. DIRECTSHOW CAPTURE

This section goes through a simple DirectShow based video capture and display sample step-by-step. This sample requires an open window to work properly.

Start with the necessary include files and pointer declarations.

```
#include <windows.h>
#include <dshow.h>
#include <d3d9.h>
#include <vmr9.h>
#include <streams.h>
#include "fg04iface.h"

char UFGName[] = "UFG-05 Channel"
bool Success = false;

IMoniker                    *pMoniker = NULL;
ICreateDevEnum              *pDevEnum = NULL;
IEnumMoniker                *pClassEnum = NULL;
IVMRFilterConfig            *pFilterConfig = NULL;
IBaseFilter                 *pVmr = NULL;
IEnumPins                   *pEnumPins = NULL;
IAMStreamConfig             *pStreamConfig = NULL;
IEnumMediaTypes             *pMedia = NULL;
AM_MEDIA_TYPE               *pmt = NULL, *pfnt = NULL;
IFG04KsproxySampleConfig    *pConfig = NULL;
IGraphBuilder               *pGraphBuilder = NULL;
IVMRWindowlessControl9      *pWindowlessControl = NULL;
IPin                        *pCamOutPin = NULL;
IBaseFilter                 *pSrc = NULL;
IMediaControl               *pMediaControl = NULL;
IMediaEventEx               *pMediaEvent = NULL;
```

Function beginning and COM initialization

Unlike the example in the previous sample, this example uses C++ exceptions to handle errors. In this case it offers simpler error handling.

```
try
{
     Res = CoInitialize(NULL);
     if((Res != S_OK)&&(Res != S_FALSE))
     {
           // CoInitialize() function has failed.
           throw(ErrorException("",-1));
     }

     Res = CoCreateInstance(CLSID_SystemDeviceEnum,
                  NULL,
                  CLSCTX_INPROC,
                  IID_ICreateDevEnum,
                  (void **) &pDevEnum);

     if(FAILED(Res))
     {
           // Failed to create system device enumerator.
           throw(ErrorException("",-1));
     }

     Res = pDevEnum->CreateClassEnumerator(
                  CLSID_VideoInputDeviceCategory,
                  &pClassEnum,
                  0);

     if((FAILED(Res)) || (pClassEnum == NULL))
     {
           // Failed to get Class enumerator interface.
           throw(ErrorException("",-1));
     }
```

<*Continued*>

UFG-05 Device Search

The class enumeration interface to find the UFG-05 capture card.

```
while((pClassEnum->Next(1, &pMoniker, NULL) == S_OK) &&
      (pConfig == NULL))
{
     IPropertyBag *Props;
     Res = pMoniker->BindToStorage(0, 0,
                IID_IPropertyBag, (void **)&Props);

     if(FAILED(Res))
     {
           // Failed to get the assoaciated property bag.
           pMoniker->Release();
           pMoniker = NULL;
           continue;
     {

     tagVARIANT  FriendlyName;
     FriendlyName.vt = VT_EMPTY;

     Res = Props->Read(L"FriendlyName", &FriendlyName,
                        NULL);
     Props->Release();
     if(strcmp(UFGName, DisplayName.bstrVal) == 0)
     {
           if(SUCCEEDED(pMoniker->BindToObject(0,0,
                      IID_IBaseFilter,(void**)&pSrc)))
           {
                 if(FAILED(pSrc->QueryInterface(
                      __uuidof(IFG04KsproxySampleConfig),
                      (void**)&pConfig)))
                 {
                      // EMI Interface not available.
                      throw(ErrorException(“”,-1));
                 }
           }
           else
           {
                 // Failed to get basefilter interface.
                 throw(ErrorException(“”,-1));
           }
     }
     pMoniker->Release();
     pMoniker = NULL;
}

if(pConfig == NULL)
{
     // We either have EMI and Source filter, or nothing.
     throw(ErrorException(“”,-1));
}
```

<*Continued*>

DirectShow GraphBuilder.

Create a Graph builder which creates the DirectShow filter graph, also the code below creates the VMR9 filter instance and adds it to the filter graph. Also, the previously created video source filter is added on the graph.

```
Res = CoCreateInstance(CLSID_FilterGraph,
           NULL,
           CLSCTX_INPROC_SERVER,
           IID_IGraphBuilder,
           (void **)&pGraphBuilder);

if(FAILED(Res))
{
     // Failed to create instance of graphbuilder
     throw(ErrorException("",-1));
}

Res = CoCreateInstance(CLSID_VideoMixingRenderer,
           NULL,
           CLSCTX_INPROC,
           IID_IBaseFilter,
           (void**)&pVmr);

if(FAILED(Res))
{
     // failed to create instance of the VMR9 filter
     throw(ErrorException("",-1));
}

Res = pGraphBuilder->AddFilter(pVmr,
           L"Video Mixing Renderer");

if(FAILED(Res))
{
     // Failed to add VMR9 filter on the graph
     throw(ErrorException("",-1));
}

Res = pGraphBuilder->AddFilter(pSrc,L"Video Capture");

if(FAILED(Res))
{
     // Failed to add video source filter on the graph
     throw(ErrorException("",-1));
}
```

<*Continued*>

Windowless Mode

This sample uses windowless mode. There is need to have your own window in which the video is displayed. The below sample retrieves IFilterConfig interface in order to configure the VMR9 to use Windowless mode.

```
try
{
     Res = pVmr->QueryInterface(IID_IVMRFilterConfig,
                 (void**)&pFilterConfig);

     if(FAILED(Res))
     {
           // Failed to get IFilterConfig interface.
           throw(ErrorException("",-1));
     }

     pFilterConfig->SetRenderingMode(VMRMode_Windowless);
}
__finally
{
     // This code is executed regardless of above code...
     if(pFilterConfig != NULL) pFilterConfig->Release();
     pFilterConfig = NULL;
}
```

Clipping Window

Next, set the clipping window. The sample assumes a variable called “WindowHandle” contains the wanted clipping window's handle.

```
try
{
     Res = pVmr->QueryInterface(IID_IVMRWindowlessControl,
                 (void**)&pWindowlessControl);

     if(FAILED(Res))
     {
           // Failed to get WindowlessControl interface.
           throw(ErrorException("",-1));
     }

     pWindowlessControl->SetVideoClippingWindow(Handle);
}
__finally
{
     // This code is executed regardless of above code...
     pVmr->Release();
     pVmr = NULL;
}
```

<*Continued*>

Find Output Pin on Source Filter

Get the source filter's output pin. There is only one, so make a search for the correct pin here.

```
Res = pSrc->EnumPins(&pEnumPins);
if(FAILED(Res))
{
     // Failed to get IEnumPins interface
     throw(ErrorException("",-1"));
}

try
{
     pEnumPins->Reset();
     pEnumPins->Next(1, &pCamOutPin, NULL);
}
__finally
{
     pEnumPins->Release();
     pEnumPins = NULL;
}
```

Decide the format type

Next decide the DirectShow graph's format type. Practically just enumerate all types, and take the first FORMAT_VideoInfo type can be found.

```
pCamOutPin->EnumMediaTypes(&pMedia);
if(FAILED(Res))
{
     // Failed to create IEnumMediaTypes interfae
     throw(ErrorException("",-1));
}

try
{
     while(pMedia->Next(1, &pmt, 0) == S_OK)
     {
           if( pmt->formattype == FORMAT_VideoInfo )
           {
                 pfnt = pmt;
                 // Overrid resolution and framerate here
                 // by chaning VideoInfoHeader, if necessary
                 break;
           }
           DeleteMediaType( pmt );
     }
}
__finally
{
     pMedia->Release();
     pMedia = NULL;
}
```

<*Continued*>

Configure Stream Format

Now configure stream to use the media-type just found. IStreamConfig interface needs to do that. Once there is the interface, set the media format and the get the media actual media format because it is possible that one or more of the filters won't accept the media format we want.

```
Res = pCamOutPin->QueryInterface(IID_IAMStreamConfig,
           (void **) &pStreamConfig );
if(FAILED(Res))
{
     // Failed to get IStreamConfig interface
     DeleteMediaType(pfnt);
     throw(ErrorException("",-1));
}

try
{
     Res = pStreamConfig->SetFormat(pfnt);
     DeleteMediaType(pfnt);
     if(FAILED(Res))
     {
           // failed to set media type
           throw(ErrorException("",-1));
     }

     Res = pStreamConfig->GetFormat(&pfnt);
     if(FAILED(Res))
     {
           // Failed to get actual media type
           throw(ErrorException("",-1));
     }

     // Read the video size here, if necessary

     DeleteMediaType(pfnt);
}
__finally
{
     pStreamConfig->Release();
     pStreamConfig = NULL;
}
```

<*Continued*>

Render and Start the Graph

Now just render our graph and start it.

```
Res = pGraphBuilder->QueryInterface(IID_IMediaControl,
           (void**)&pMediaControl);

if(FAILED(Res))
{
     // Failed to get IMediaControl interface.
     throw(ErrorException("",-1));
}

::GetClientRect (Handle,&rcDest);
pWindowlessControl->SetVideoPosition(NULL, &rcDest);

pGraphBuilder->Render(pCamOutPin);
pMediaControl->Run();

Success = true;
```

<Continued>

End of Function

Almost done. There can dispose a bunch of unnecessary COM interfaces. Remember started function with a try block. This terminates block and also handles error situations properly.

```
}
__finally
{
     if(pDevEnum != NULL) pDevEnum->Release();
     if(pClassEnum != NULL) pClassEnum->Release();
     if(pFilterConfig!=NULL) pFilterConfig->Release();
     if(pVmr!=NULL) pVmr->Release();
     if(pMoniker!=NULL) pMoniker->Release();
     if(pEnumPins!=NULL) pEnumPins->Release();
     if(pMedia!=NULL) pMedia->Release();
     if(pStreamConfig!=NULL) pStreamConfig->Release();

     pMoniker = NULL;
     pDevEnum = NULL;
     pClassEnum = NULL;
     pFilterConfig = NULL;
     pVmr = NULL;
     pEnumPins = NULL;
     pStreamConfig = NULL;
     pMedia = NULL;

     if(!Success)
     {
           if(pCamOutPin != NULL) pCamOutPin->Release();
           if(pMediaControl != NULL)
           {
                 pMediaControl->Release();
           }
           if(pGraphBuilder != NULL)
           {
                 pGraphBuilder->Release();
           }
           if(pWindowlessControl != NULL
           {
                 pWindowlessControl->Release();
           }
           if(pConfig != NULL) pConfig->Release();
           if(pSrc != NULL) pSrc->Release();
           if(pMediaEvent != NULL) pMediaEvent->Release();

           pCamOutPin = NULL;
           pMediaControl = NULL;
           pGraphBuilder = NULL;
           pWindowlessControl = NULL;
           pConfig = NULL;
           pSrc = NULL;
           pMediaEvent = NULL;
     }
}
```

<*Continued*>

Clean-up on Program Exit

When the program is closing, delete the rest of the COM interfaces.

```
if(pConfig != NULL)
{
     pConfig->Commit();
     pConfig->Release();
}

if(pCamOutPin != NULL) pCamOutPin->Release();
if(pMediaControl != NULL) pMediaControl->Release();
if(pGraphBuilder != NULL) pGraphBuilder->Release();
if(pWindowlessControl != NULL)
{
     pWindowlessControl->Release();
}
if(pSrc != NULL) pSrc->Release();
if(pMediaEvent != NULL) pMediaEvent->Release();
```

6. ERROR-EXCEPTION CLASS

A Class used to pass error information in exception form.

6.1. Public Members

6.1.1. Class Constructor

```
ErrorException
(
     char *iErMsg,
     unsigned int iErCode
);
```

Parameters

iErMsg
Pointer to a NULL terminated string. The string content is copied to the public variable ErMsg. The string must be shorter than 256 characters, and should not special characters.

iErCode
Error code. The code is copied into the public variable ErCode.

6.1.2. Public Variables

The following variables are freely accessible.

```
char ErMsg[256];
```

ErMsg public variable holds a human readable message describing the error which caused the exception to get thrown.

```
unsigned int ErCode;
```

ErCode public variable holds the error code associated to the error which caused the exception to get thrown.

6.2. Using ErrorException Class

6.2.1. Throwing ErrorException

The ErrorException class can be used in error messaging as follows

```
throw(ErrorException("Your Error message",ERROR_CODE));
```

6.2.2. Catching ErrorException

Whenever a function can potentially throw ErrorException objects, one should use the try … catch statements to catch the exceptions.

```
try
{
    UFG = new UFG05_Cap(0,512,384,6000);
}
catch(ErrorException &Except)
{
    // Process the error here
}
```

7. UFG-05 PROPERTIES

UFG05_WR_AUTOADJUST
Write only. Write value "1" to trigger an automatic adjustment of incoming video.

UFG05_WR_DETECT_TIMING
Write only. Write value "1" to detect timing values.

UFG05_WR_CALIBRATE_ADC
Write only. Write value "1" to cause ADC to be re-calibrated.

UFG05_WR_RESET_ADC
Write only. Write value "1" to reset ADC.

UFG05_BRIGHTNESS
Controls brightness. Range 0-255. Default = 128.

UFG05_CONTRAST
Controls contrast. Range 0-255. Default = 135.

UFG05_HUE
Controls hue. Range 0-255. Default = 128.

UFG05_SATURATION
Controls saturation. Range 0-255. Default = 128.

UFG05_SOURCE
Source control.
0 = D-SUB, 1 = DVI, 2 = CVBS / S-VIDEO.

UFG05_COLORSPACE
Selects color space for D-SUB input.
0 = RGB, 1 = Component Video (YPbPr).

UFG05_VIDEOTYPE
Selects video type for Composite (CVBS) / S-VIDEO input.
0 = Composite (CVBS), 1= S-Video

UFG05_VIDEOSTANDARD
Select video standard for Composite (CVBS) / S-VIDEO input.
0 = PAL, 1 = NTSC, 2 = SECAM

UFG05_HRES
Horizontal resolution, pixels. This value has no effect if source is DVI.

UFG05_VRES
Vertical resolution, pixels. This value has no effect if source is DVI.

UFG05_HSTART
H-START sync value. This value has no effect if source is DVI.

UFG05_VSTART
V-START sync value. This value has no effect if source is DVI.

UFG05_HTOTAL
H-TOTAL sync value. This value has no effect if source is DVI.

UFG05_PHASE
Pixel phase lock adjustment. Range 0-255. This value has no effect if source is DVI.

UFG05_AUTOTIMING
Automatic timing control. 0 = MANUAL, 1 = AUTOMATIC

UFG05_RD_MIN_WIDTH
Read only. Minimum width available, pixels.

UFG05_RD_MIN_HEIGHT
Read only. Minimum height available, pixels.

UFG05_RD_MAX_WIDTH
Read only. Maximum width available, pixels.

UFG05_RD_MAX_HEIGHT
Read only. Maximum height available, pixels.

UFG05_RD_NATIVE_WIDTH
Read only. Incoming video native resolution (width).

UFG05_RD_NATIVE_HEIGHT
Read only. Incoming video native resolution (height).

UFG05_RD_NATIVE_REFRESH_RATE
Read only. Incoming video refresh rate (Reported as tenth of Hz).

UFG05_RD_NATIVE_INTERLACED
Read only. Indicates if incoming video is interlaced. 0 = Non-interlaced, 1 = Interlaced.

UFG05_RD_SYNCLOSS
Read only. Indicates if sync is lost. 0 = Sync valid, 1 = Sync lost.

UFG05_RD_SYNCVALID
Read only. Indicates if sync is valid. 0 = Sync lost, 1 = Sync valid.

UFG05_RD_DVI_STATUS
Read only. Indicates DVI connection status. 0 = No cable connected, 1 = Cable connected.

UFG05_RD_VGA_STATUS
Read only. Indicates VGA connection status. 0 = No cable connected, 1 = Cable connected.

UFG05_RD_VIDEO_STATUS
Read only. Indicates Video connection staus. 0 = No cable connected, 1 = Cable connected.