Allied Vision

APPLICATION NOTE

File Access Control - Reading and Writing User Data in Alvium Flash Memory

V1.0.0
2025-Jul-21

Table of contents

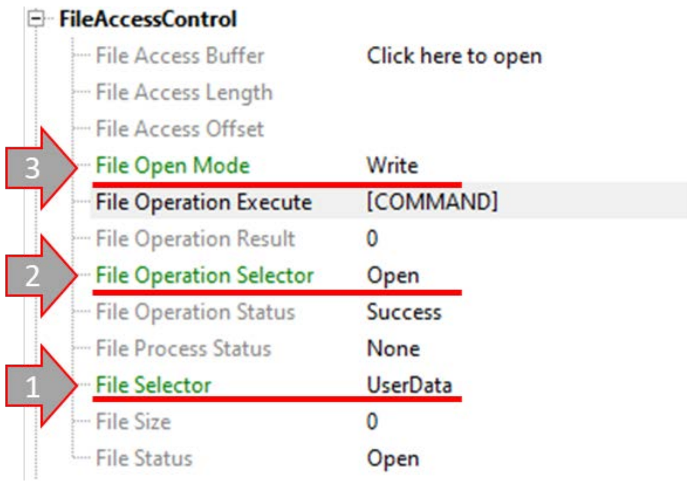
Introduction

This application note gives a basic overview on how to use the File Access Control features to write and access User Data in the flash memory of Alvium cameras. The first focus will be using Vimba X Viewer, followed by C++ code using the Vimba X SDK.

File access in Vimba X Viewer

Writing User Data

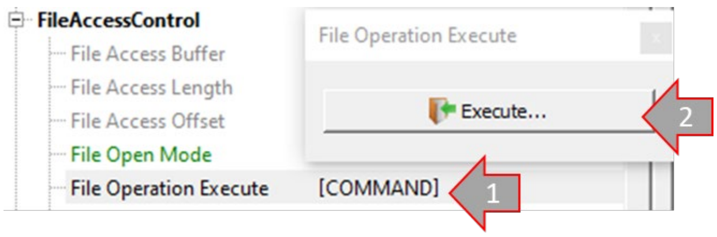
Writing User Data to the flash memory will always append to the already written User Data. The write pointer cannot be changed, except if the file was deleted. In this example, there is no User Data in the flash memory yet, so the pointer is at 0. First, the features `FileOperationSelector`, `FileOpenMode` and `FileSelector` need to be set as shown in the screenshot.



FileAccessControl	
File Access Buffer	Click here to open
File Access Length	
File Access Offset	
File Open Mode	Write
File Operation Execute	[COMMAND]
File Operation Result	0
File Operation Selector	Open
File Operation Status	Success
File Process Status	None
File Selector	UserData
File Size	0
File Status	Open

Figure 1 - Feature Selection for Write Operation

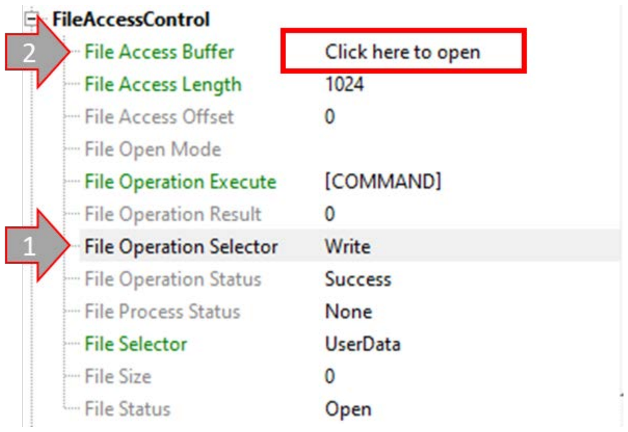
After completing these settings, the feature FileOperationExecute needs to be executed.



FileAccessControl	
File Access Buffer	
File Access Length	
File Access Offset	
File Open Mode	
File Operation Execute	[COMMAND]

Figure 2 - Execute Command Feature "FileOperationExecute"

Now that the file is open for writing, the next operation to be selected in the `FileOperationSelector` is to Write to make the `FileAccessBuffer` accessible.

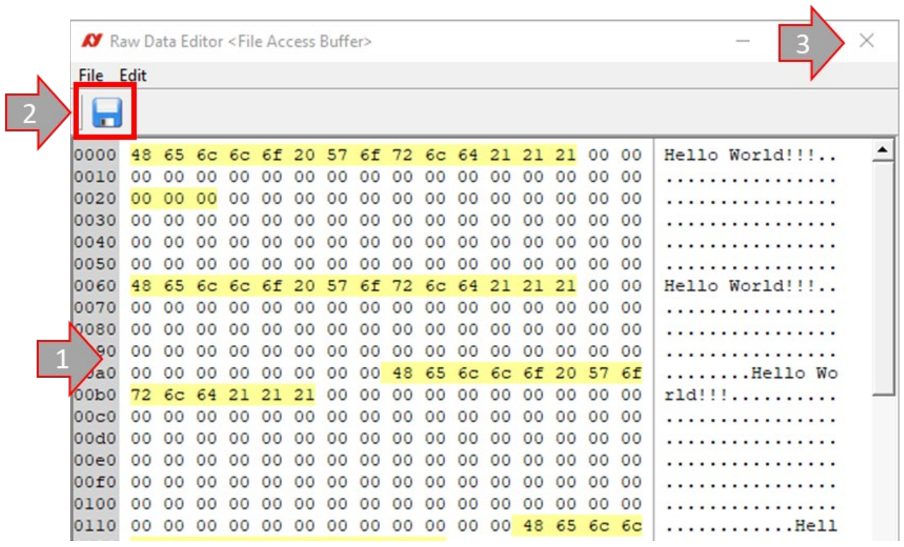


FileAccessControl

File Access Buffer	Click here to open
File Access Length	1024
File Access Offset	0
File Open Mode	
File Operation Execute	[COMMAND]
File Operation Result	0
File Operation Selector	Write
File Operation Status	Success
File Process Status	None
File Selector	UserData
File Size	0
File Status	Open

Figure 3 - Set FileOperationSelector to Write and open the FileAccessBuffer

Clicking on **Click here to open** opens the integrated Raw Data Editor, where you can see the HEX and ASCII values of the buffer. After editing the HEX values, click the save icon and close the buffer editor.



```
0000  48 65 6c 6c 6f 20 57 6f 72 6c 64 21 21 21 00 00   Hello World!!!..
0010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0040  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0060  48 65 6c 6c 6f 20 57 6f 72 6c 64 21 21 21 00 00   Hello World!!!..
0070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
  90  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
  a0  00 00 00 00 00 00 00 00 48 65 6c 6c 6f 20 57 6f   ........Hello Wo
00b0  72 6c 64 21 21 21 00 00 00 00 00 00 00 00 00 00   rld!!!..........
00c0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00d0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00e0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00f0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0100  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0110  00 00 00 00 00 00 00 00 00 00 00 00 48 65 6c 6c   ............Hell
```

Figure 4 - Edit the Values of the FileAccessBuffer in the Raw Data Editor

Once the data in the buffer is complete, the buffer can be written into the camera memory. For that, after closing the Raw Data Editor, if none of the features have been changed in the meantime, only `FileOperationExecute` needs to be executed. If your data does not need the 1024 bytes of the buffer, you can limit the amount of buffer written into the camera, by editing `FileAccessLength.` Every ASCII sign should take up one byte, so for "Hello World!" your `FileAccessLength` should be 12.

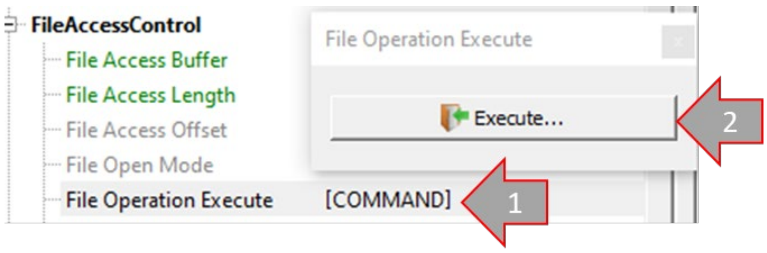
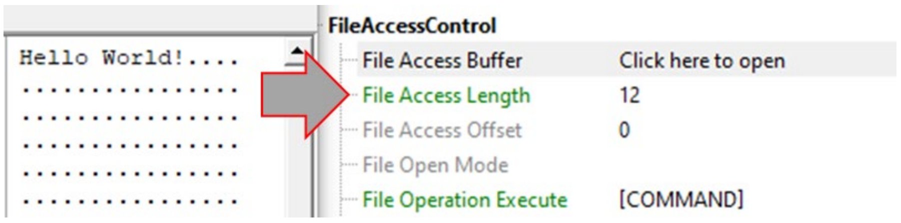


Figure 5 - Execute the Write Operation to write the Buffer into the Camera Memory

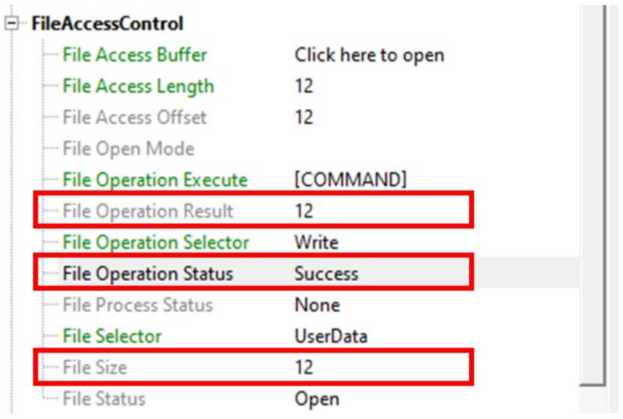


```
Hello World!....
```

FileAccessControl	
File Access Buffer	Click here to open
File Access Length	12
File Access Offset	0
File Open Mode	
File Operation Execute	[COMMAND]

Figure 6 - For smaller Data, use FileAccessLength to only write the Amount needed

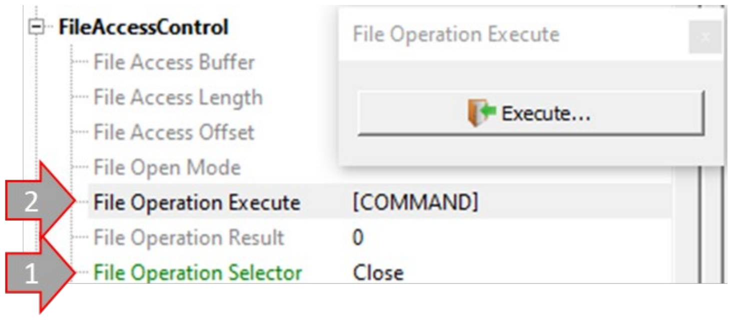
To check if the Write operation was successful, `FileOperationStatus` can be checked for ***`Success`*** and `FileSize` can be checked to see if the number matches the `FileAccessLength` plus the previous size and if `FileOperationResult` matches the `FileAccessLength`.



FileAccessControl	
File Access Buffer	Click here to open
File Access Length	12
File Access Offset	12
File Open Mode	
File Operation Execute	[COMMAND]
File Operation Result	12
File Operation Selector	Write
File Operation Status	Success
File Process Status	None
File Selector	UserData
File Size	12
File Status	Open

Figure 7 - Indicators to check if Write Operation was successful

Note: The write pointer can be reset only by deleting the file, power cycling the camera is not enough. If the file is not closed, the User Data is not saved into the camera.

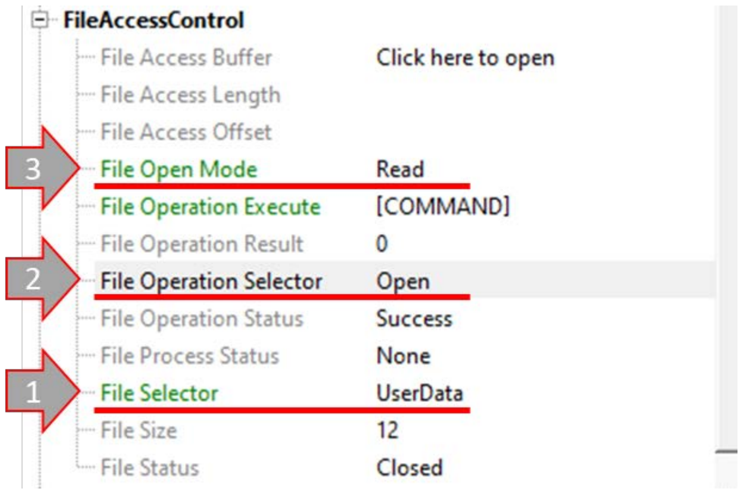


FileAccessControl	
File Access Buffer	
File Access Length	
File Access Offset	
File Open Mode	
File Operation Execute	[COMMAND]
File Operation Result	0
File Operation Selector	Close

Figure 8 - If the File is not closed, close it with these two Selections before reading it

Reading User Data

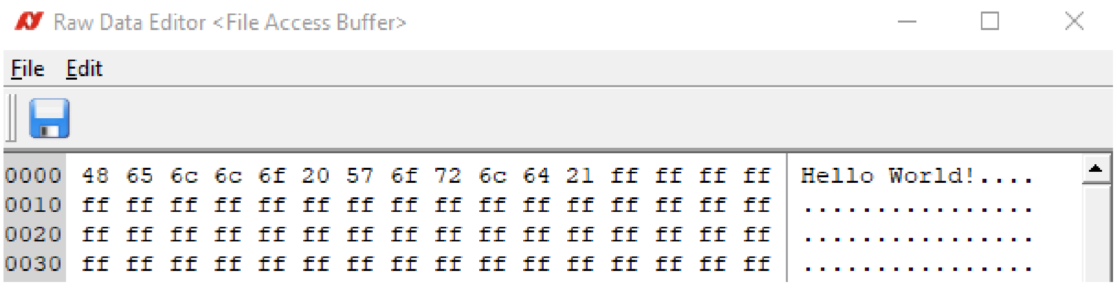
If the `FileStatus` is not *`Closed`*, then close the file first, before opening it in Read mode. To do that select Close in the `FileOperationSelector` and execute the `FileOperationExecute` command. Now that the file is for sure closed, open it in Read mode, by selecting the following features and values as shown in the screenshot and executing the `FileOperationExecution` command.



```
FileAccessControl
  File Access Buffer        Click here to open
  File Access Length
  File Access Offset
  File Open Mode            Read
  File Operation Execute    [COMMAND]
  File Operation Result     0
  File Operation Selector   Open
  File Operation Status     Success
  File Process Status       None
  File Selector             UserData
  File Size                 12
  File Status               Closed
```

Figure 9 - Open UserData in Read Mode

Next, the opened file needs to be loaded into the `FileAccessBuffer` with a Read operation. Select Read for the `FileOperationSelector` feature and execute the `FileOperationExecution` command. Clicking to open the Raw Data Editor will now show the User Data that was loaded from the camera.



```
0000  48 65 6c 6c 6f 20 57 6f 72 6c 64 21 ff ff ff ff   Hello World!....
0010  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff   ................
0020  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff   ................
0030  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff   ................
```

Figure 10 - UserData is successfully read out from the Camera Flash Memory

File access with Vimba C++

Of course, Vimba C++ SDK uses the same features as shown in the GUI of the Vimba X Viewer, so this section is to summarize possible implementations of functions that close, open, read, write and delete User Data from the camera flash memory. If you would like to see the example program that these code snippets are based on, please contact support@alliedvision.com. The example is named UserData_FileAccess.cpp.

Closing files

To do anything with the files, it is important to open them in the right mode. Therefore, any function that opens the file should first check that it is closed and close it if necessary. It is also best practice to close the file after each write and read action to make sure that the file was written correctly in case of a program crash or the camera being physically disconnected and that there is no confusion about the state of the file.

```
void close_file(CameraPtr cam){
        VmbError_t err;
        FeaturePtr pFeature;
        string stringVal;
        // Check if the file is closed
        cam->GetFeatureByName("FileStatus", pFeature);
        err = pFeature->GetValue(stringVal);
        if (stringVal == "Closed") {
                cout << "File is already closed." << endl;
        }
        else {
                cam->GetFeatureByName("FileOperationSelector", pFeature);
                err = pFeature->SetValue("Close"); // Select “Close” as operation
                cam->GetFeatureByName("FileOperationExecute", pFeature);
                err = pFeature->RunCommand(); // Execute the selected operation
        }
}
```

Opening User Data in Read Mode

The file can only be opened if it is not already opened. Therefore, it is important to check for the file status before trying to open it, to ensure that the right file is opened in the right mode.

```
Void open_UserData_r(CameraPtr cam){
        VmbError_t err;
        cam->GetFeatureByName("FileSelector", pFeature);
        err = pFeature->SetValue("UserData"); // Select UserData file
        // Check if selected file is opened, if yes close file
        cam->GetFeatureByName("FileStatus", pFeature);
        err = pFeature->GetValue(stringVal);
        if (stringVal == "Open") {
                close_file(cam);
        }
```

```

        cam->GetFeatureByName("FileOperationSelector", pFeature);
        err = pFeature->SetValue("Open"); // Select “Open” as operation
        cam->GetFeatureByName("FileOpenMode", pFeature);
        err = pFeature->SetValue("Read"); // Select “Read” as mode
        cam->GetFeatureByName("FileOperationExecute", pFeature);
        err = pFeature->RunCommand(); // Execute the selected operation
}
```

Opening User Data in Write Mode

```
void open_UserData_w(CameraPtr cam){
        VmbError_t err;
        FeaturePtr pFeature;
        string stringVal;
        cam->GetFeatureByName("FileSelector", pFeature);
        err = pFeature->SetValue("UserData"); // Select UserData file
        // Check if selected file is opened, if yes close file
        cam->GetFeatureByName("FileStatus", pFeature);
        err = pFeature->GetValue(stringVal);
        if (stringVal == "Open") {
                close_file(cam);
        }
        cam->GetFeatureByName("FileOperationSelector", pFeature);
        err = pFeature->SetValue("Open"); // Select “Open” as operation
        cam->GetFeatureByName("FileOpenMode", pFeature);
        err = pFeature->SetValue("Write"); // Select “Write” as mode
        cam->GetFeatureByName("FileOperationExecute", pFeature);
        err = pFeature->RunCommand(); // Execute the selected operation
}
```

Deleting files

```
void delete_file(CameraPtr cam) {
        VmbError_t err = 0;
        FeaturePtr pFeature;
        string stringVal;
        VmbInt64_t nFileSize = 0;
        err = close_file(cam); // Close file before deleting
        cam->GetFeatureByName("FileStatus", pFeature);
        err = pFeature->GetValue(stringVal);
        if (stringVal == "Closed") { // Check if FileStatus is “Closed”
                cam->GetFeatureByName("FileOperationSelector", pFeature);
                err = pFeature->SetValue("Delete"); // Select “Delete” as mode
                cam->GetFeatureByName("FileOperationExecute", pFeature);
                err = pFeature->RunCommand(); // Execute the selected operation
                cam->GetFeatureByName("FileSize", pFeature);
                pFeature->GetValue(nFileSize);
                if (nFileSize == 0) { // Check if FileSize is 0
                        cout << "File deleted." << endl;
                }
                else {
                        cout << "Delete error: File size is not zero." << endl;
                }
        }
        else {
                cout << "Delete error: File was not closed successfully.” << endl;
        }
}
```

Reading out User Data

The size of the file access buffer is only 1024 bytes maximum. So, if the file size of the User Data is larger than that, the file needs to be read out in several steps. This can be done by editing the feature `FileAccessOffset`. For example: First read out the first 1024 bytes and save it into a variable. Then set `FileAccessOffset` to 1024, read out more of the file and add it to your existing variable. This example code does not take larger file sizes into account, but that can be changed easily by for example making the offset an argument of the function.

```
UcharVector read_UserData(CameraPtr cam){
        VmbError_t err;
        FeaturePtr pFeature;
        UcharVector UserData;
        VmbInt64_t nFileSize = 0;
        string stringVal;
        err = open_UserData_r(cam); // Open file in read mode
        cam->GetFeatureByName("FileSize", pFeature);
        pFeature->GetValue(nFileSize); // Check file size
        if (nFileSize > 0) {
                cam->GetFeatureByName("FileOperationSelector", pFeature);
                err = pFeature->SetValue("Read"); // Select “Read” as operation
                cam->GetFeatureByName("FileAccessLength", pFeature);
                // Set file access length depending on file size
                if (nFileSize > 1024)
                {
                        pFeature->SetValue(1024);
                }
                else {
                        pFeature->SetValue(nFileSize);
                }
                cam->GetFeatureByName("FileAccessOffset", pFeature);
                pFeature->SetValue(0); // Select 0 as file access offset
                cam->GetFeatureByName("FileOperationExecute", pFeature);
                err = pFeature->RunCommand(); // Execute the selected operation
                cam->GetFeatureByName("FileOperationStatus", pFeature);
                pFeature->GetValue(stringVal);
                if (stringVal == "Success"){ // Check for read success
                        cam->GetFeatureByName("FileAccessBuffer", pFeature);
```

```
                    // Read out buffer into variable UserData
                            pFeature->GetValue(UserData);
                    }
            }
            else {
                    cout << "Can't read UserData, file size is 0." << endl;
            }
            close_file(cam); // Close file after reading -> good practice
            return UserData;
}

int main(){
            …
            UcharVector file_access_buffer;
            file_access_buffer = read_UserData(cam);
            for (int i = 0; i < file_access_buffer.size(); i++){
                    cout << file_access_buffer[i];
            …
            }
```

Writing User Data

By default, the write operation will append the data in the file access buffer to the existing data. Modifying the data can only be achieved by reading it out, modifying it, deleting the old file, and writing the modified file on the camera. This example includes an argument to differentiate between add and overwrite.

```
UcharVector write_UserData(CameraPtr cam, UcharVector UserData,
                                 bool overwrite = false) {
        VmbError_t err;
        FeaturePtr pFeature;
        VmbInt64_t nFileSize = 0;
```

```
        VmbInt64_t nDataSize = 0;
        string stringVal;
        nDataSize = UserData.size();
        if (nDataSize > 0)
        {
                // Overwrite current file on camera
                if (overwrite == true) {
                        err = delete_file(cam); // Closes and deletes file
                        err = open_UserData_w(cam); // Open file in write mode
                        cam->GetFeatureByName("FileOperationSelector", pFeature);
                        err = pFeature->SetValue("Write"); // Select “Write”
                        cam->GetFeatureByName("FileAccessLength", pFeature);
                // Write only as much memory, as needed for data (recommended)
                        err = pFeature->SetValue(nDataSize);
                        cam->GetFeatureByName("FileAccessBuffer", pFeature);
                        err = pFeature->SetValue(UserData); // Load into buffer
                        cam->GetFeatureByName("FileOperationExecute", pFeature);
                        err = pFeature->RunCommand(); // Execute write operation
                        // Optional: Check file size for successful write
                        cam->GetFeatureByName("FileSize", pFeature);
                        pFeature->GetValue(nFileSize);
                }
                else {
                        // Add to current file on camera
                        err = close_file(cam);
                        err = open_UserData_w(cam); // Open file in write mode
                        cam->GetFeatureByName("FileOperationSelector", pFeature);
                        err = pFeature->SetValue("Write"); // Select "Write"
                        cam->GetFeatureByName("FileAccessLength", pFeature);
                        err = pFeature->SetValue(nDataSize);
                        cam->GetFeatureByName("FileAccessBuffer", pFeature);
                // Write only as much memory, as needed for data (recommended)
                        err = pFeature->SetValue(UserData);
```

```
                        cam->GetFeatureByName("FileOperationExecute", pFeature);
                        err = pFeature->RunCommand(); // Execute write operation
                }
        }
        else {
                cout << "Data is empty, no data was written." << endl;
        }
        close_file(cam); // Close file after writing -> necessary!
        return UserData;
}
int main(){
        …
        // Write data in ASCII as string, to be formatted into UcharVector
        string stringBaseData = "Hello World!!!!!!!!!!!!!!!!!!!";
        UcharVector new_UserData;
        VmbUchar_t temp;
        // Separate string into VmbUchar and stack them into a UcharVector
        for (int i = 0; i < stringBaseData.length(); i++){
                temp = (VmbUchar_t)stringBaseData[i];
                new_UserData.push_back(temp);
        }
        write_UserData(cam, new_UserData, true); // Write new data into camera
        …
}
```

Contact us

Website, email

General
www.alliedvision.com/en/contact
info@alliedvision.com

Distribution partners
www.alliedvision.com/en/avt-locations/avt-distributors

Support
www.alliedvision.com/en/support
www.alliedvision.com/en/about-us/contact-us/technical-support-repair-/-rma

Offices

Europe, Middle East, and Africa (Headquarters)
Allied Vision Technologies GmbH
Taschenweg 2a
07646 Stadtroda, Germany
T// +49 36428 677-0 (Reception)
T// +49 36428 677-230 (Sales)
F// +49 36428 677-28

North, Central, and South America, Canada
Allied Vision Technologies Canada Inc.
300 – 4621 Canada Way
Burnaby, BC V5G 4X8, Canada
T// +1 604 875 8855

Asia-Pacific | China
Allied Vision Technologies Shanghai Co Ltd.
B-510, Venture International Business Park
2679 Hechuan Road
Minhang District, Shanghai 201103
People's Republic of China
T// +86 21 64861133

USA
Allied Vision Technologies, Inc.
102 Pickering Way - Suite 502
Exton, PA 19341, USA
Toll-free// +1-877-USA-1394
T// +1 978 225 2030

Singapore
Allied Vision Technologies Asia Pte. Ltd
82 Playfair Rd, #07-01 D'Lithium
Singapore 368001
T// +65 6634 9027

Japan
Allied Vision Technologies
Yokohama Portside Bldg. 10F
8-1 Sakae-cho, Kanagawa-ku
Yokohama-shi, Kanagawa, 221-0052
T// +81 (0) 45 577 9527

Liability, trademarks, and copyright