PROGRAMMER'S USER GUIDE

for the

PAKON F235 SCANNERS

TLA Version 0.0.30.2

15 June 2004

Disclaimer

Trademarks

Pakon Inc
5950 Clearwater Drive
Suite 100
Minnetonka, MN 55343
(952) 936-9500
www.pakon.com

Table of Contents

1 Introduction

The Pakon F235*plus* and F235C scanners are input scanners that convert images from developed film into digital images. They can scan APS or 35mm color negative, color reversal, or black & white film. The filmstrips can be as short as two frames and as long as forty frames. The F235C has a built-in APS cartridge loader and MOF reader. The F235*plus* can scan APS film, but it has no cartridge loader, the film must be manually fed in and it has no MOF reader.

The scanner will be connected to a host computer via a USB 2.0 cable, and software on the host will need to be developed that communicates with and controls the scanner. A software development kit (SDK) is supplied with the scanner that allows programmers to do this. This SDK consists of a DLL called *TLA* that utilizes a COM interface. The main purpose of this document is to describe the use of this COM interface.

Through TLA, the client software can operate the film scanner, including setting parameters for framing and cropping, scanning, performing color corrections, and saving pictures to disk or memory. The scanner will utilize buffer space on the host computer in order to provide for fast scans, and to allow complete control over the final product, digital images.

Included with the SDK is a sample client application called *TLAClientDemo*. This sample client was produced with Microsoft Visual C++ using MFC and the COM interface. It will be used in this manual as a source for sample code for programmer's to learn how to operate the scanner. Feel free to use it as a starting point for developing your own client application for the F235 scanners.

1.1 Minimum Host Requirements

- CPU: 733 MHz Intel Pentium III or AMD Athlon processor (with MMX support).
- 2 IDE Hard Drives: One 5GB capable of 24MB/sec sustained data rate (7200 RPM), dedicated to the scan data. One for OS, application programs, etc. with 50MB of free space.
- OS: Windows 2000 Professional, Service Pack 2 or better.
- RAM: 256 MB.
- Port: one USB 2.0 (a dedicated port is highly recommended), Microsoft USB 2.0 driver.

1.2 Recommended Host Requirements

- CPU: 1 GHz Pentium IV or AMD Athlon (or higher).
- 2 IDE Hard Drives: one 20GB capable of 24MB/sec sustained data rate, dedicated to the scan data. One for OS, application programs, etc. with 50MB of free space.
- OS: Windows 2000 Professional, Service Pack 2 or better.
- RAM: 512 MB.
- Port: one USB 2.0 in the chipset dedicated to the scanner, Microsoft USB 2.0 driver.

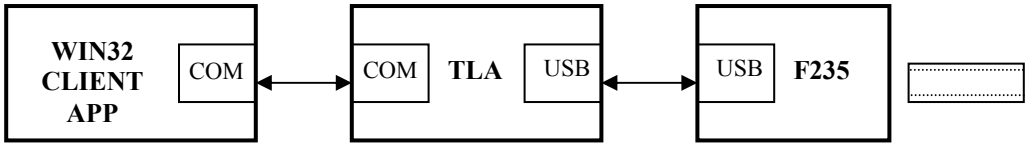
See the F235 Service Manual (part #124794) for further specifications.

2 Overview

The Software Development Kit provided with the F235 Scanner consists of a library, TLA.DLL, a *Programmer's Reference Manual*, this *Programmer's User Guide*, and the sample client application, TLAClientDemo, written in Visual C++ with MFC (source and executable).

2.1 Architecture

Shown below is the architecture of TLA integrated with a client application and communicating with the F235 Scanner.



TLA sits between the client application and the scanner, providing a COM interface and making the USB communication transparent to the client. The "COM" within TLA refers to the COM Server and the "COM" within the client application refers to the COM Client.

2.2 System Requirements

The development environment for the client must meet the following minimal requirements. The operating system must be Windows 2000 Professional with Service Pack 2 or better. Any programming tool that works with COM can be used, however, the programming tool must be able to implement the callback interface that TLA uses for progress notification (see section 2.6 COM Implementation). Microsoft Visual C++ with MFC is the preferred tool for development. Other tools may be used, but Pakon may not provide a sample client application and may not have the expertise to be able to support the development.

2.3 Installation

Follow these steps to setup the scanner and the SDK:

- Configure the drive for the high-resolution buffer. This should be N: and unformatted.
- Run Setup.exe from the SDK disk and follow the instructions.
- Setup will install TLA in the "C:\Program Files\Pakon\TLA COM Server" folder.
- Connect the USB 2.0 cable from the scanner to the PC.
- Make sure the USB 2.0 port is working properly. We require the Microsoft driver, version 5.1.2600.0 which can be obtained from Pakon.
- Turn on the scanner; Windows should report a new USB device. The F235 drivers will be loaded automatically assuming you accept the unsigned digital signature warnings.
- Start TLAClientDemo and try scanning some film.
- If you want to run without a scanner, there is a simulator version of TLA also, TLAs.dll.

2.4 Scanner Features

The Pakon F235*plus* and F235C Scanners can scan APS or 35mm color negative, color reversal, or black & white film. The filmstrips can be as short as two frames and as long as forty frames. The F235C scanner has the capability to read MOF data from APS film (in a cartridge). See section 5.6 Reading MOF from APS for more on this. Both scanners will automatically read the DX and DXn codes from APS or 35mm film.

The film is fed into the scanner (emulsion side up) via the film guide, which automatically feeds the film through the scanner. A stepper motor powers the film drive and a lamp provides the light that passes through the film for scanning. When first turned on, the lamp will take some time to warm up. After a scan, the lamp can be dimmed to prolong its life. If the lamp burns out, the client will be informed with a callback message. There is an access panel for replacing the lamp.

The scanner can scan at base4, base8 or base16 resolution. Base8 results in twice as many pixels as base4, and base16 gives twice as many pixels as base8. The table below lists the default number of pixels for each resolution, film type and buffer:

	35 mm		24 mm	
	High resolution	Low resolution	High resolution	Low resolution
Base4	1000 x 1500	250 x 375	857 x 1500	214 x 375
Base8	1400 x 2100	175 x 262	1200 x 2100	150 x 262
Base16	2000 x 3000	250 x 375	1714 x 3000	214 x 375

Each scanner is calibrated prior to shipment. If a scanner needs to be recalibrated, a qualified technician must perform the calibration.

2.5 TLA Features

When TLA is installed, a hard drive on the host is established as the buffer drive. Then, when the software is run and a roll of film is scanned, TLA creates two buffers and fills them with the scan information. There is a low-resolution buffer stored in RAM and a high-resolution buffer stored on the buffer drive. The information stored in these buffers represents the entire strip of film including the area between frames. The low-resolution buffer is ¼ or ⅛ the size of the high-resolution buffer, depending on the resolution (see the table above).

When film is scanned, TLA establishes a frame (rectangle) for each picture and stores this framing information. The client can alter the default framing parameters prior to scanning (e.g. for panoramic or wide-lux film). The client can also alter the frames resulting from a scan (for cropping pictures or moving frames) prior to saving. See section 5.3 Framing and Cropping for further explanation of the subject.

When film is scanned, TLA keeps track of each roll, each strip, and each picture. Each picture belongs to a strip, and each strip belongs to a roll. A roll can consist of one strip or multiple strips. If a roll has been cut into multiple strips, they can be scanned in one after another and assigned to one roll; this is called scanning in strip mode. Each roll requires one high-resolution buffer and one low-resolution buffer.

Each roll, strip and picture is referenced with an index. The indexes start at zero and each group is numbered independent of the others. For example, if two rolls have been scanned, they are numbered 0 to 1; if these two rolls consist of four strips, they are numbered 0 to 3; if these four strips consist of 24 pictures, they are numbered 0 to 23. TLA keeps a current index for both strips and pictures.

Some functions can operate on multiple pictures in one call. TLA provides multiple index enumerations for this case: INDEX_All and INDEX_AllSelected. In addition, pictures can be individually marked as hidden or selected. Pictures marked as hidden will be skipped with multiple indexes. Pictures marked as selected will be the only ones used with an index of INDEX_AllSelected.

To help organize data, TLA has created the concept of a scan group and a save group. When pictures are scanned, they are added to the scan group. From there, the client can move them into the save group. Pictures in the save group can then be saved to disk or memory. The reason for having two groups is for processing speed; once a roll of film is in the save group, the client can start a second scan and at the same time, start to process the pictures in the save group. When the pictures have been processed and saved, the second scan will be complete or nearly complete. Then the process can start a third roll and so on.

In order for this to work, TLA needed one COM interface for scanning and one COM interface for saving (see the next section). It also needed to define operations that would take a long time ("long operations"), and establish a separate processing thread for each of these. When a long operation – such as scanning – is called, the function starts a separate worker thread and returns immediately. The client is then free to do other things, including starting another long operation in the other group – such as saving. The client is informed of the progress of long operations by way of a callback mechanism (see the next section).

2.6 COM Implementation

In this implementation of COM, there is a main class defined, TLAMain. The globally unique identifier, `CLSID_TLAMain`, is provided by TLA to identify the class. The client application can use this identifier to create an instance of TLA.

There are also five interfaces defined, three for general use and two for a callback mechanism. These are summarized in the table below.

Interface	Description	Identifier
Callback Client	An interface defined by the client that allows the server to send progress and error messages back to the client.	IID_ICallBackClient
Long Ops CB	The interface that establishes the callback mechanism with the server. (The callback is used mainly in long operations.)	IID_ILongOpsCB

TLA Main	The interface to the main functions of TLA, including initializing the scanner and handling errors.	IID_ITLAMain
Scan Pictures	The interface for performing scanning operations, such as setting up scanning parameters, adjusting the scanner, and actual scanning.	IID_IScanPictures
Save Pictures	The interface for processing images and saving images to disk or memory.	IID_ISavePictures

Note that the Callback Client interface must be defined by the client application. The other four interfaces are defined in TLA. The identifiers listed are globally unique identifiers provided by TLA and can be used to establish the interfaces in your client application.

In this implementation of COM, the rules of COM have been followed with one exception. In order to save images to client memory, a pointer to this memory must be passed to the COM Server. Passing pointers through COM is not standard. TLA gets around this rule by casting the pointer to an integer, passing the integer across, then converting the integer back to a pointer.

2.6.1 Callback Client Interface

In order for the client to receive progress and error messages from the server for long operations, the client must implement a callback interface. A sample implementation is shown in the TLAClientDemo source code. Near the beginning of the file TLAClientDemoDlg.cpp, the interface is defined as `ICallBackClient`. Also defined are the standard interface methods, `QueryInterface`, `AddRef` and `Release`; these are required for any interface.

Besides the standard methods, the method `Awake` must be defined. This is where the progress and status messages will be sent to the client. This method takes two `long` arguments, the first indicates the operation being reported and the second indicates the status. Enumerations are provided in TLA.idl for both arguments. `WORKER_THREAD_OPERATION_000` defines enumerations for the first argument and `WORKER_THREAD_PROGRESS_000` defines enumerations for the second argument. The second argument may report a percentage or absolute progress; for example, while saving images to disk, TLA will report the progress as a percentage or as a count of the number of images saved. You can specify which method to use in the call to InitializeScanner.

The `Awake` method no longer has the restriction that it return quickly. In TLAClientDemo, the implementation of the `Awake` method simply posts a message to the main window and returns immediately. The main window can then handle the message at its own pace. You may implement the `Awake` method in any manner you see fit. See 4.1.1 Callback Example for a sample implementation of the Awake method.

Besides progress and error messages, the callback mechanism will be used to report changes in the status of hardware components. TLA may send these callback messages to the client at any time. The *lOperation* parameter will be `WTO_HardwareError` and the *lStatus* parameter will be one of the `HARDWARE_CB_000` enumerations. `HARDWARE_CB_LAMP_BURN_OUT_ERROR` is one of the more common ones, which means the lamp has burned out and needs to be replaced. Two others are `HARDWARE_CB_FILM_EMULSION_DOWN` and `HARDWARE_CB_FILM_TAIL_FIRST`, which means the film was inserted incorrectly. See 6.8 Hardware Callbacks for more on this.

Throughout this manual, and also in the *F235 Programmer's Reference Manual*, the use of callback messages will be pointed out.

2.6.2 Error Handling

Three types of error conditions are possible with TLA:

1. Starting a TLA server long operation that returns an error through the callback interface.

   In this case, the TLA server calls `Awake` with an error code as the first argument (something like WTO_xxxError). An example is getting a WTO_ScanError during a scan operation. To get more information from the server, you should call ITLA_Main::GetAndClearLastError. As an aid in troubleshooting, this function gives a call stack trace of what the error was and which class and function started the process that resulted in the error. This information is also entered into an error log that is saved to disk. The filename depends on the interface, but will be either PakonErrorLogMain.txt, PakonErrorLogScan.txt or PakonErrorLogSave.txt. These files can be found in the "C:\Program Files\Pakon\TLA COM Server" folder.

2. Calling a TLA server function that returns an error code.

   An example is a call to CBAdvise, which returns an error code (hr). How the error is handled is up to the client. The Win32 GetErrorInfo function could be called, GetAndClearLastError could be called, and an error message could be displayed. In TLAClientDemo, there is the `AnalyzeComError()` function (defined in the Globals.cpp file) that does this.

3. Calling a COM function that returns an error code.

   An example is a call to ::CoInitializeEx(), which returns an error code (hr). The Win32 `FAILED(hr)` macro may be used to identify that there was an error in the COM call. How the error is handled is up to the client. TLAClientDemo displays a message and aborts the operation. It uses the `FormatHResult()` function (defined in globals.cpp) to format the message to the user.

See 4.1.2 Error Handling Examples for examples of handling errors in TLAClientDemo.

3 Scanning Basics

This section describes the basic steps in scanning with the F235 Scanner and how to implement each step using TLA.

The basic steps are initialization, then scan setup, performing a scan, processing the resulting images, then saving to disk or memory, and finally shutting down.

3.1 Initialization

There are two steps to initialization: first the COM interface must be initialized, then the scanner must be initialized. The scanner can be initialized only after COM is initialized.

To initialize COM, first decide on a threading model (see 5.1 Thread Model), then make a call to CoInitializeEx. Next, make a call to CoCreateInstance to create an instance of the TLA class, using LongOpsCB as the interface. Then, make a call to CBAdvise to inform TLA of the client's callback interface. Lastly, call QueryInterface for the TLAMain, ScanPictures and SavePictures interfaces. See the `bInitCom()` and `bInitScanner()` functions in TLAClientDemo for examples of these calls. **NOTE**: The client must insure that only one instance of TLA is running at a time. If two clients try to start TLA, TLA won't work properly.

To initialize the scanner, simply make a call to InitializeScanner. Once initialization is complete (InitializeScanner is a long operation, you'll have to wait until it is complete), you should call GetInitializeWarnings to see if any warning messages were issued. See the `bInitScanner()` and `CompleteInitialization` functions in TLAClientDemo for examples of these calls. Before scanning the first time, it is recommended that ForceCorrections be called in order to warm up the lamp and exercise the stepper motors. This call can be made with any configuration.

3.2 Scan Setup

Prior to scanning, correcting for gain & offset and fixed pattern should be done for the configuration – resolution and film format – about to be scanned. In this version of TLA, these corrections are done automatically when a scan is requested. (They are done only if you are scanning with a different configuration than the previous scan, or if scanning with the same configuration for a long period, and the *iDarkPointCorrectInterval* has expired. This parameter can be set with the PutScannerInfo000 function.)

Calls to other functions are optional based on the particular situation. For example, if panoramic or wide-lux or half-frame film is to be scanned, or if a special cropping window needs to be used, a call to PutScannerInfoPreFrameUser should be made. This will adjust the frame width and/or cropping rectangle for the scan.

3.3 Scanning

There is only one step necessary for scanning: making a call to ScanPictures. You can specify the desired resolution and whether you are scanning a single strip or multiple strips. Once scanning starts, the client will get progress messages through the callback interface (Awake function). If you need to stop the scan, you can make a call to ScanCancel().

3.4 Processing

When scanning is complete, a call to GetPictureCountScanGroup should be made. This will report the number of strips and the number of pictures scanned, but more importantly, it will report any warning messages for the scan. These include framing warnings, motor speed warnings, and warnings concerning reading DX codes, film product and film specifier.

If the roll is unacceptable, the client can delete the roll with a call to DeleteRollInScanGroup. If the roll is acceptable, a call to MoveOldestRollToSaveGroup will move it to the save group.

Once a roll is in the save group, the pictures can be processed. If the operator needs to see the pictures in order to properly process them, a call to SaveToClientMemory could be used to download the pictures to the client where they can be displayed. Once the save has begun, or before the save is begun, the client must add client memory buffers for the pictures to be saved with calls to ClientMemoryBufferAdd. Two buffers should be sufficient.

If one of the pictures is bad, a call to DeletePicture will remove it. If a picture was missed in the scan, a call to InsertPicture will extract a picture from the scan data and insert it into an existing strip.

If a picture needs to be rotated 90 or 180 degrees, a call to PutPictureInfo or PutPictureRotation will do that. If a picture's color settings need to be altered, a call to PutPictureColorSettings or PutPictureColorSettingsDifferential will do that. If a picture needs to be cropped or the framing alignment needs to be adjusted, a call to PutPictureFramingUserInfo will do that.

3.5 Saving

Once a set of pictures has been processed, they can be saved either to memory with a call to SaveToClientMemory, or to permanent files with a call to SaveToDisk. You can specify an individual picture, or all selected pictures (use PutPictureInfo or PutPictureSelection to mark a picture as selected), or all pictures. For SaveToDisk, the destination of individual pictures can be specified by setting the filename and directory with a call to PutPictureInfo or PutPictureInfo1. The destination can be any valid media (hard drive, floppy disk, etc.).

If a save operation needs to be stopped for any reason, the client can call SaveCancel().

After pictures have been saved, the client can call ReleaseSaveGroup() to remove all the pictures from the save group. This also releases all buffers and framing information. If client memory buffers were added, they can be released with a call to ClientMemoryBufferDismissAll() and then deallocated.

3.6 Shutdown

When the client is done with TLA and ready to shut down, it should cancel any current scan, cancel any current save, and delete all pictures and client memory buffers. Then the client should release the connection to TLA with a call to CBUnadvise. Next, the client should release the other three interfaces: SavePictures, ScanPictures and TLAMain. Lastly, the client should call CoUninitialize() to shut down COM.

4 Scanning Scenarios

This section will present various scenarios for scanning. The first will cover a basic scan operation, and subsequent scenarios will get more complex. Each scenario will describe the steps that need to take place with sample function calls.

4.1 Scenario I

In this scenario, we will scan in one roll (24 pictures) of 35mm color negative film at base4 and save the pictures to disk as JPEG files. The first step is to initialize COM. The sample code below comes from the bInitCom() and bInitScanner() functions in TLAClientDemo (the error handling has been removed to keep things simple):

```
HRESULT hr;
// Initialize COM with the multithreaded model
hr = ::CoInitializeEx(NULL, COINIT_MULTITHREADED);

// Create an instance of TLA
hr = ::CoCreateInstance(CLSID_TLAMain,
                        0,
                        CLSCTX_INPROC_SERVER,
                        IID_ILongOpsCB,
                        (void**)&m_pILongOpsCB);

// Create a callback object
CCallBackClient *pICallBackClient = new CCallBackClient();

// Set up the callback connection
hr = m_pILongOpsCB->CBAdvise(pICallBackClient, &m_lCookie);

// Server did an AddRef we need to release
pICallBackClient->Release();

// Create the other three interfaces
hr = m_pILongOpsCB->QueryInterface(IID_ITLAMain,
                                   (void**)&m_pITLAMain);
hr = m_pILongOpsCB->QueryInterface(IID_IScanPictures,
                                   (void**)&m_pIScanPictures);
hr = m_pILongOpsCB->QueryInterface(IID_ISavePictures,
                                   (void**)&m_pISavePictures);
```

We call CoInitializeEx to initialize COM and specify the multithreaded model. Then we create an instance of TLA, also creating the long ops interface. Next we create a callback interface object, then use the long ops interface to advise TLA of this new interface, storing the identifier of the connection in m_lCookie. CBAdvise will do an AddRef in the callback client, so we need to do a Release. Lastly, the other three interfaces are created with calls to QueryInterface.

The next step is to initialize the scanner. This sample comes from the bInitScanner() and CompleteInitialization functions in TLAClientDemo (the error handling has been removed to keep things simple):

```
#define INITIALIZE_CONTROL (INITIALIZE_ProgressUpdatesAsPercent |
                            INITIALIZE_FirmwareUpdate)
#define SAVE_TO_MEMORY_TIMEOUT 20000  // 20 secs
#define SAVE_TO_SHARED_MEMORY_SIZE 0

// Initialize the scanner and TLA
hr = m_pITLAMain->InitializeScanner(INITIALIZE_CONTROL,
                                    SAVE_TO_MEMORY_TIMEOUT,
                                    SAVE_TO_SHARED_MEMORY_SIZE);
...
// Wait for InitializeScanner to finish - monitor callbacks.
// (see UpdateInitializeProgress() and CompleteInitialization()
//  in TLAClientDemo)
...
// Get warning codes
int iInitializeWarnings;
hr = m_pITLAMain->GetInitializeWarnings(&iInitializeWarnings);

// Check for warnings
if (iInitializeWarnings & INITIALIZEW_EEPROM_BLANK)
{
   // EEPROM not initialized
}
else if(iInitializeWarnings & INITIALIZEW_EEPROM_CHECKSUM_BAD)
{
   // EEPROM read failed
}
```

Here we initialize the scanner with appropriate parameters. The InitializeControl parameter should always include INITIALIZE_FirmwareUpdate so the scanner is operating with the latest firmware. Then we wait for initialization to finish (it's a long operation) by waiting for a callback message with WTO_InitializeProgress as the operation and WTP_ProgressComplete as the status.

Once finished, we call GetInitializeWarnings to see if there were any warning messages. If the EEPROM is blank or the checksum is bad, that indicates that the nonvolatile parameters could not be read. If this is the first time you've used the scanner, it should be returned for repair. If you've been scanning previously and one of these warnings appears, you can continue scanning, although the results should be monitored and the scanner should be returned for repair when it's convenient.

Assuming we received no warnings, we are ready to scan. When the film is ready, we call ScanPictures, feed the film into the scanner and wait for the results. See the OnScan() function in TLACLientDemo for a sample call to ScanPictures.

```
HRESULT hr;
int iScanControl = 0;

iScanControl = SCAN_Read_DX | SCAN_LampWarmUp
```

```
hr = m_pIScanPictures->ScanPictures(RESOLUTION_BASE_4,
                                    FILM_COLOR_NEGATIVE,
                                    FILM_FORMAT_35MM,
                                    STRIP_MODE_FULL_ROLL,
                                    iScanControl);
```

In this example, we first set the scan control features we want to use for this scan. In this case, we have requested the scanner to read the DX codes, and to warm-up the lamp (if it isn't already).

Then we call the ScanPictures function with our parameters. The fifth parameter specifies the strip mode. Since we are scanning a full roll, we specify STRIP_MODE_FULL_ROLL in this example.

This call to ScanPictures will start the scan. The motor drive will start and the operator will need to feed in the roll of film within a timeout period (defined with PutScannerInfo000). This is a long operation, so you will have to monitor callback messages through the Awake function and wait for the scan to complete. At the end of this section is an example that shows how this could be accomplished (see Callback Example).

When the scan is complete, the roll of scanned pictures will reside in the scan group, but we can't do much with them until we move them into the save group. We should, however, see if there were any scan warnings. We can do this with a call to GetPictureCountScanGroup. The sample code below (extracted from the TLAClientDemo function OnMoveToSaveGroup(), with error handling removed) shows how this could be done:

```
HRESULT hr;
hr = m_pISavePictures->GetPictureCountScanGroup(0,
                                                &m_iScanStripCount,
                                                &m_iScanPictureCount,
                                                &m_iScanWarnings);
if(m_iScanWarnings != 0)
{
   CheckScanWarnings(m_iScanWarnings);
}
```

In the call to GetPictureCountScanGroup, the first argument represents the roll index. Since we've scanned only one roll so far, the index is zero. If there are scan warnings, then we call CheckScanWarnings, which can be defined any way the client sees fit. In TLAClientDemo, we simply examine the scan warnings and build a message string that is displayed to the user (see the source code for more detail). Also, if there was any warning about the motor speed, we ask the user if they would like to have the motor speed adjusted (with a call to AdjustMotorSpeed). A code excerpt from TLAClientDemo is shown below:

```
if(!bMotorSpeedOk)
{
   if(IDYES == MessageBox(_T("Adjust Motor Speed?"),
                          _T("Scan Warnings"),
                          MB_YESNO | MB_ICONEXCLAMATION))
   {
```

```
        HRESULT hr = m_pIScanPictures->AdjustMotorSpeed();
    }
}
```

If you get a warning about the motor speed, it's a good idea to make a call to AdjustMotorSpeed so that future scans are as good as they can be. AdjustMotorSpeed uses the warnings to determine how to adjust the speed. Also, if you feel the last scan wasn't as good as it could be, you can call AdjustMotorSpeed, then rescan.

Now we're ready to move the group of scanned pictures into the save group where they can be processed and saved. We do this with a call to MoveOldestRollToSaveGroup:

```
hr = m_pISavePictures->MoveOldestRollToSaveGroup();
```

Once this is done, there are no pictures or rolls in the scan group, they are all in the save group. A call to GetRollCountScanGroup will report zero. A call to GetPictureCountScanGroup will return an error since there are no valid roll indexes, as will a call to DeleteRollInScanGroup. However, you can call GetPictureCountSaveGroup, which will report the number of rolls in the save group (1 in our case), the number of strips (1), and the number of pictures (24, assuming all pictures were scanned successfully).

At this point, we are ready to process and save our pictures to disk. In this example, the only processing we will do to the scanned pictures is in the call to SaveToDisk:

```
HRESULT hr;
int iSaveOptions;

iSaveOptions = SAV_SizeOriginal |
               SAV_UseCurrentRotation |
               SAV_UseColorCorrection |
               SAV_UseColorSceneBalance |
               SAV_UseScratchRemovalIfAvailable;

hr = m_pISavePictures->SaveToDisk(INDEX_All,
                                  NULL,
                                  iSaveOptions,
                                  0,
                                  0,
                                  0,
                                  0,
                                  iFILE_FORMAT_JPG,
                                  90,
                                  0,
                                  0);
```

In the code above, we first assign the save options we want to use. We don't want to scale the pictures, so we use SAV_SizeOriginal. We don't want the pictures rotated at all, so we use

SAV_UseCurrentRotation. We do want automatic color correction and color scene balance, and we want to use scratch removal if available.

In the call to SaveToDisk, INDEX_All indicates that all pictures will be saved. The second parameter specifies the filename, and since we are saving multiple pictures, this must be NULL. TLA will use the default directory ("c:\temp") and filename (the frame name) of each picture. You can change the default directory with a call to PutSaveInfo. You can specify a different directory and filename for each picture with a call to PutPictureInfo or PutPictureInfo1. If changing directories or filenames, make these calls before the call to SaveToDisk.

The third parameter is our save options we set earlier. The next two parameters (0, 0) define our scaling window, and since we are not scaling (we specified SAV_SizeOriginal), these values are not used. The sixth and seventh parameters (0, 0) specify a rotation and a scaling method, and since we are not rotating the pictures (SAV_UseCurrentRotation) or scaling, these are ignored. The file format is JPEG and we use a compression value of 90. The last two parameters (0, 0) are not yet implemented.

Assuming the pictures were saved successfully, the only step left is to clear out the save group; this removes all pictures and frees up the buffers. We do this with a call to ReleaseSaveGroup:

```
HRESULT hr = m_pISavePictures->ReleaseSaveGroup();
```

If the client stored any picture information internally, this would be the time to delete the information and free up the memory. For an example of this, see the OnReleaseSaveGroup() function in TLAClientDemo.

4.1.1 Callback Example

The sample code below shows how callback messages could be handled; this includes progress messages as well as error messages. See TLAClientDemo for more detail.

```
// This function is passed to the server for the server to
// provide status callbacks during long operations
STDMETHODIMP CCallBackClient::Awake(long lOperation,
                                    long lStatus)
{
   AfxGetMainWnd()->PostMessage(WM_COM_CALLBACK,
                                (ULONG)lOperation,
                                lStatus);
   return S_OK;
}

// This maps messages to function calls
BEGIN_MESSAGE_MAP(CTLAClientDemoDlg, CDialog)
   ...
   ON_MESSAGE(WM_COM_CALLBACK, OnMsgComCallBack)
END_MESSAGE_MAP()

// This function handles the callback messages
void CTLAClientDemoDlg::OnMsgComCallBack(UINT ulOperation,
                                         LONG lStatus)
{
   switch(ulOperation)
   {
      ...
      case WTO_ScanProgress:
      {
         UpdateScanProgress(lStatus);
         break;
      }
      case WTO_ScanError:
      {
         HandleWTOError(INT_IID_IScanPictures,
                        IDS_COM_ERROR_SCAN, lStatus);
         break;
      }
   }
}
```

The first function defines our Awake method. It simply posts a message to the main window, passing through the lOperation and lStatus parameters. Next is code that maps these messages to a function call, OnMsgComCallBack. Last is the definition of this function. It takes the ulOperation parameter and calls a certain function based on this operation. It handles both the progress and error messages from TLA.

The only task left is to define the UpdateScanProgress and HandleWTOError functions. Examples from TLAClientDemo are shown below (see the source code for more detail):

```
void CTLAClientDemoDlg::UpdateScanProgress(LONG lStatus)
{
   switch(lStatus)
   {
      case WTP_Initialize:
      {
         SetDlgItemText(IDC_SCAN_STATUS, _T("Scanning"));
         m_ProgressScan.SetPos(WTP_ProgressStart);
         break;
      }

      case WTP_ProgressStart:
      {
         SetDlgScanning(TRUE);
         break;
      }

      case WTP_ProgressComplete:
      {
         UpdateScanCounts();
         SetDlgScanning(FALSE);
         break;
      }

      default:
      {
         m_ProgressScan.SetPos(lStatus);
         break;
      }
   }
}
```

This function switches on the lStatus parameter, which provides the state of progress. If this is initialize, then we show "Scanning" as the scan status and set the progress bar at the start. If lStatus is at the start of progress, then we set up our dialog box for scanning (disable some controls, enable others). If progress is complete, then we update the number of scanned rolls and set our dialog box back. Otherwise, we update our progress bar.

The HandleWTOError function is defined in the next section.

4.1.2 Error Handling Examples

As stated earlier, three types of error conditions are possible with TLA:

1. Starting a TLA server long operation that returns an error through the callback interface.

In this case, TLA sends an error message through the Awake method. In TLAClientDemo, the Awake method calls the HandleWTOError function if the operation parameter indicates an error (WTO_xxxError). The HandleWTOError function is written to handle any error message we receive through the Awake method. This could be due to an initialize error, a hardware error, a film track test error, a force diagnostics error, a focus error, a force corrections error, a lamp warm-up error, an advance film error, a film guide position error, an APS loader error, a scan error or a save error. Below is an excerpt from TLAClientDemo that shows how it handles a scan error (see the source code for more detail).

```
void CTLAClientDemoDlg::HandleWTOError(int iShort_IID,
                                       int iOperationID,
                                       LONG lStatus)
{
   BSTR bstrError = NULL;
   BSTR bstrErrorNumbers = NULL;
   int iError;
   CString strCaption;
   strCaption.LoadString(iOperationID);

   HRESULT hr = m_pITLAMain->GetAndClearLastError(iShort_IID,
                                                  &bstrError,
                                                  &bstrErrorNumbers,
                                                  &iError);

   if(iOperationID == IDS_COM_ERROR_HARDWARE)
   {
      ...
   }
   else
      MessageBox(W2T(bstrError), strCaption,
                 MB_ICONERROR | MB_OK);
   }

   ::SysFreeString(bstrError);
   ::SysFreeString(bstrErrorNumbers);

   switch (iShort_IID)
   {
   case INT_IID_IScanPictures:
      SetDlgScanning(SAVE_MODE_IDLE);
      break;
   ...
   }
}
```

This function first calls GetAndClearLastError; this function should be called whenever you get an error (it doesn't hurt to call it too often). This will give us the trace of the error and also clear the error. Next, we display a message box with the text of the trace of the error (class, method and error message). After freeing the error strings, we switch based on the interface where the error occurred. If this was scan-pictures, then we set up our dialog box to allow scanning again then return.

2. Calling a TLA server function that returns an error code.

In this case, we will call AnalyzeComError, which is defined in the file globals.cpp. The code below shows how this is done when a call to CBAdvise fails:

```
hr = m_pILongOpsCB->CBAdvise(pICallBackClient, &m_lCookie);
if(FAILED(hr))
{
   ::AnalyzeComError(hr, IID_ILongOpsCB, m_pILongOpsCB,
                     NULL, FALSE);
   pICallBackClient->Release();
   return FALSE;
}
```

See the TLAClientDemo source code for the definition of AnalyzeComError.

3. Calling a COM function that returns an error code.

In this case, all we do is display an error message to the user and abort. The sample below shows how this is done when a call to CoInitializeEx fails:

```
hr = ::CoInitializeEx(NULL, COINIT_MULTITHREADED);

if(FAILED(hr))
{
   CString strResult;
   ::FormatHResult(hr, strResult);
   CString str;
   str.Format(_T("CoInitialize Failed\n%s"), strResult);
   AfxMessageBox(str, MB_OK | MB_ICONEXCLAMATION);
   return FALSE;
}
```

4.2 Scenario II

In this scenario, we will scan a roll (24 pictures) of 35mm color negative film at base8, do some processing, then save the pictures to client memory (so they can be printed by the client). We'll assume the scanner is already initialized (see Scenario I to learn how to do this). So, the first step is to call ScanPictures, feed the film into the scanner and wait for the results.

```
HRESULT hr;
int iScanControl = 0;

iScanControl = SCAN_UseScratchRemoval | SCAN_Read_DX |
               SCAN_LampWarmUp
hr = m_pIScanPictures->ScanPictures(RESOLUTION_BASE_8,
                                    FILM_COLOR_NEGATIVE,
                                    FILM_FORMAT_35MM,
                                    STRIP_MODE_FULL_ROLL,
                                    iScanControl);
```

This time, we set the scan control features to use scratch removal, to read the DX codes, and to warm-up the lamp (if it isn't already).

Then we call the ScanPictures function with our parameters. The fifth parameter specifies the strip mode. Since we are scanning a full roll in this scenario, it's STRIP_MODE_FULL_ROLL.

When the scan is complete, we should call GetPictureCountScanGroup to see if there were any scan warnings. If there were any scan warnings about the motor speed, we should call the function AdjustMotorSpeed(). Next, we can move the scanned pictures into the save group with a call to MoveOldestRollToSaveGroup. (See Scenario I.)

At this point, we are ready to process and save our pictures. This time, we want to do a little more processing to the pictures, and we want to display them to the operator so he/she can adjust the color, adjust the framing and rotate pictures. We will load the pictures into client memory and the first step is to allocate buffers (this code is excerpted from the OnMoveToSaveGroup() function in TLAClientDemo):

```
CSize sizeImageDisplay;
CWnd *pWndPicture = GetDlgItem(IDC_PICTURE);
CRect crPicture;
pWndPicture->GetClientRect(crPicture);
crPicture.DeflateRect(MARGIN, MARGIN);

sizeImageDisplay.cx = crPicture.Width();
sizeImageDisplay.cy = crPicture.Height();

m_ClientBuffers.bAllocateBuffers(sizeImageDisplay,
                                 iFILE_FORMAT_SAVE_TO_MEMORY_DIB_8,
                                 TRUE)
```

Here, we get the picture object from the dialog, and store the dimensions in sizeImageDisplay. Then we call bAllocateBuffers to set up two buffers. The picture will be displayed on-screen so we specify DIB 8 as the file format. The definition of the bAllocateBuffers function is shown below (error handling has been removed for clarity):

```
BOOL CClientBuffers::bAllocateBuffers(CSize csSaveSize,
                                      int iSaveToMemoryFormat,
                                      BOOL bMultiplePictures)
{
   // make sure the buffers are empty
   ASSERT(m_pClientBuffer1 == NULL);
   ASSERT(m_pClientBuffer2 == NULL);

   m_bUsingBuffer1 = FALSE;

   m_iBufferSize = iRequiredBufferSize(csSaveSize,
                                       iSaveToMemoryFormat);

   // allocate buffer one and add to server
   if (m_pClientBuffer1 = new UCHAR[m_iBufferSize])
   {
      m_pISavePictures->ClientMemoryBufferAdd(
                                       (int)m_pClientBuffer1,
                                       m_iBufferSize);
      m_bUsingBuffer1 = TRUE;
   }

   // if two buffers are required
   if (bMultiplePictures)
   {
      m_bUsingBuffer1 = FALSE;
                        // continued...
      // allocate buffer two and add to server
      if(m_pClientBuffer2 = new UCHAR[m_iBufferSize])
      {
         m_pISavePictures->ClientMemoryBufferAdd(
                                       (int)m_pClientBuffer2,
                                       m_iBufferSize);
         m_bUsingBuffer1 = TRUE;
         }
      }
   }
   // return success/fail status
   return m_bUsingBuffer1;
}
```

This routine calculates the required buffer size, then calls ClientMemoryBufferAdd twice, once for each buffer. The definition of iRequiredBufferSize() is shown below:

```
int CClientBuffers::iRequiredBufferSize(CSize& csSaveSize,
                                        int iSaveToMemoryFormat)
{
   int iBufferSize;
   ...
   if (iSaveToMemoryFormat == iFILE_FORMAT_SAVE_TO_MEMORY_DIB_8)
   {
      // DIB line may contain a 3-byte padding
      // Assume narrowest dimension is the line
      if (csSaveSize.cx > csSaveSize.cy)
      {
         iBufferSize = csSaveSize.cy + 3;  // 3-byte padding
         iBufferSize *= csSaveSize.cx * 3; // 3 channels
      }
      else
      {
         iBufferSize = csSaveSize.cx + 3;
         iBufferSize *= csSaveSize.cy * 3; // 3 channels
      }

      iBufferSize += sizeof(BITMAPFILEHEADER) +
                     sizeof(BITMAPINFOHEADER) + 16;
   }
   return iBufferSize;
}
```

To calculate the buffer size, this routine adds 3 bytes (for the DIB padding) to the narrowest dimension of the rectangle and multiplies by the longest dimension, then multiplies by 3 (RGB channels). Then it adds room for a file header and an info header, and adds an extra 16 bytes just to be safe.

Now that we have our buffers set up, we can call SaveToClientMemory to fill them:

```
// set the save options for a display save
int iSaveOptions = 0;
iSaveOptions |= (
   SAV_SizeLimitForDisplay |
   SAV_TopDownDib |
   SAV_FastUpdate8BitDib |   // keep thumb in memory
   SAV_UseCurrentRotation |
   SAV_UseLoResBuffer |        // Since displaying thumb size
   SAV_DoNotScaleUp |
   SAV_UseColorSceneBalance |
   SAV_UseColorAdjustments |
   SAV_UseScratchRemovalIfAvailable );

m_pISavePictures->SaveToClientMemory(
   INDEX_All,
   iSaveOptions,
   sizeImageDisplay.cx,      // fill the picture control
   sizeImageDisplay.cy,      // fill the picture control
   0,   // ignored since iSaveOptions has SAV_UseCurrentRotation
```

```
    SCALING_METHOD_BILINEAR,       // Normal scaling method
    iFILE_FORMAT_SAVE_TO_MEMORY_DIB_8,
    TRUE);                         // UseWorkerThread
```

Since we have 24 pictures but only two buffers, we need to manage the buffers. As soon as one buffer is filled (by TLA), we will copy the image into another location, then reuse the buffer by making another call to ClientMemoryBufferAdd. This puts the buffer back into the queue. As we are doing this, TLA is filling our second buffer. When this is complete, we copy the image and reuse the buffer. This process repeats until all 24 pictures are processed.

If TLA is ready to use a buffer before we have put it back into the queue, TLA will wait for a timeout period specified by *iSaveToMemoryTimeout* in the call to InitializeScanner(). If the timeout period expires, we will get an error code of EC_TimeOut. We start the implementation of this buffering scheme by handling the callback progress messages from the save operation:

```
void CTLAClientDemoDlg::OnMsgComCallBack(UINT ulOperation,
                                         LONG lStatus)
{
    switch(ulOperation)
    {
          ...
          case WTO_SaveProgress:
          {
                UpdateSaveProgress(lStatus);
                break;
          }
    }
}
```

When we get a callback message reporting save progress (see also Callback Example), we call the function UpdateSaveProgress. Below is a simplified excerpt from TLAClientDemo:

```
void CTLAClientDemoDlg::UpdateSaveProgress(LONG lStatus)
{
    switch(lStatus)
    {
       case WTP_Initialize:
       case WTP_ProgressStart:
       {
          break;
       }
       case WTP_ProgressComplete:
       {
          m_ClientBuffers.Delete();
          UpdatePicture();
          break;
       }
       default:
       {
          UCHAR * pDib = m_ClientBuffers.GetActiveBuffer();
```

```
            AddBitmap(pDib);
            m_ClientBuffers.bNextBuffer();
            break;
         }
      }
}
```

In this routine, we switch based on the status (from the callback message). If the status is 'initialize' or 'progress start', we do nothing. If the status is progress complete (all pictures have been saved), then we delete our client buffers and update the current picture in our display. Otherwise, the status is some partial progress; in other words, a picture has just been saved to one of our buffers (the first buffer in the queue).

In this case, we get that buffer, then call AddBitmap. This function will take the image in the buffer (a DIB) and convert it to a device-dependent bitmap (DDB), then store this DDB into a picture object we have setup. This picture object can then be displayed on-screen. After calling AddBitMap, we call bNextBuffer, which adds the last buffer back into the queue by calling ClientMemoryBufferAdd.

The next time we get a callback message regarding a save progress (another picture has been saved), we will repeat these steps. When all pictures have been saved, progress is complete and we delete our buffers. The Delete() function calls ClientMemoryBufferDismissAll, then deletes the buffer objects.

See the TLAClientDemo source code for more detail about the code.

Now that we've saved our pictures and can display them, we can let the operator adjust the results. In TLAClientDemo, there are options for inserting pictures, deleting pictures, reframing pictures, cropping pictures, rotating pictures, and adjusting the color. We won't go through the code here, but look at the TLAClientDemo source code to see how these operations might work.

Once the operator has made all the adjustments necessary, we are ready to save the pictures. Since we'll be printing the pictures, we want to save them to client memory, but this time we want to save high-resolution images in planar format. As before, the first step is to allocate two buffers, then call SaveToClientMemory:

```
iFileFormat = iFILE_FORMAT_SAVE_TO_MEMORY_PLANAR_16;
if (m_ClientBuffers.bAllocateBuffers(sizeMax, iFileFormat, TRUE))
{
   hr = m_pISavePictures->SaveToClientMemory(
                                          INDEX_All,
                                          iSaveOptions,
                                          iPictureBoundaryWidth,
                                          iPictureBoundaryHeight,
                                          iNewRotation,
                                          SCALING_METHOD_BILINEAR,
                                          iFileFormat,
                                          TRUE);
}
```

The values of iSaveOptions, iPictureBoundaryWidth, iPictureBoundaryHeight and iNewRotation are determined by how the operator adjusted the pictures.

Once the pictures are all saved, we can print them. When printing is complete, we can call the function ReleaseSaveGroup() to remove the roll of pictures from TLA's memory.

5 Miscellaneous

This section covers miscellaneous topics concerning TLA and the scanners.

5.1 Thread Model

There is a significant issue to consider when implementing your client: the thread model. The server supports both the Apartment Threaded (STA) and Free Threaded (MTA) models. The server operates in a similar manner whether you are using STA or MTA, however, there are differences.

In STA, the COM subsystem provides the server with a message pump that processes one request at a time. Therefore, until one function call has returned, another will not be allowed to start. This prevents the client from calling one short operation while another short operation is in progress. All long operations the server performs, such as scanning, return as quickly as possible after starting a worker thread. In this case, the STA mechanism would be inadequate to prevent the client from calling a short operation function while the server is in the middle of a long operation.

If using either the STA or the MTA model, the server has functionality to prevent the client from performing two operations at the same time. An example would be trying to change a scanning parameter while you are scanning – this would lead to unpredictable results.

Using the MTA model should make the software more efficient. When using the STA model, the client must marshal the interface pointer to utilize that interface from a new window.

5.2 Basic Strings

Basic strings (BSTRs) are the mechanism for passing strings to and from the COM server. You should be careful to handle BSTR variables in the manner they are used in the server. If the TLA.IDL file describes a BSTR as an [in] parameter (client sends to server), you must allocate the string using the Win32 ::SysAllocString() function or the CString::AllocSysString() function, and the server will deallocate properly.

If the TLA.IDL file describes a BSTR as an [in, out] parameter (server sends to client), declare the BSTR and set it to NULL. When the call returns, copy the BSTR to a CString, if desired, and use SysFreeString() to deallocate. There are classes available to handle BSTRs in a simpler manner; use them at your own risk.

5.3 Framing and Cropping

When TLA scans a roll of film, it scans the entire roll of film, including the area between frames. Then it moves the scan information into both the high-resolution buffer and the low-resolution buffer. Once there, it uses a set of framing parameters to try to establish the frame of each picture. The client can alter some of the default framing parameters prior to scanning, but also after the scan to alter the picture frames that TLA established. This section discusses this process.

Prior to scanning, there are certain parameters the client can view and/or change that affect the area of film scanned and also the resulting frames in the buffers. Figure 1 shows the parameters used to determine the area of film scanned.

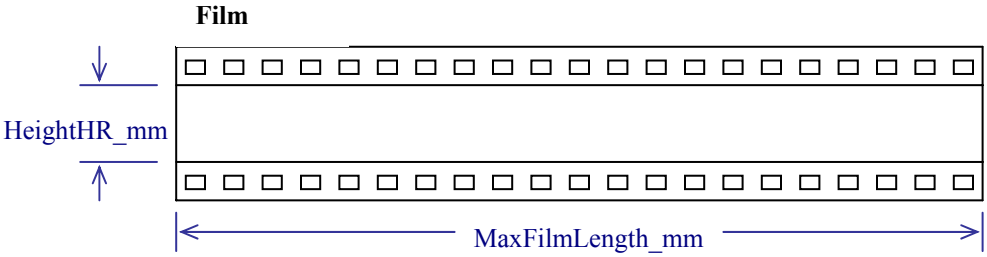


Figure 1. Parameters defining scan area

The value of HeightHR_mm cannot be changed by the client, it will be 23.7 mm for 35mm film and 16.2 mm for 24mm film. The value of MaxFilmLength_mm can be changed with a call to PutScannerInfo001; the default value is 1600 mm. This parameter is used by TLA to determine the size of the buffers.

Before scanning, there are also framing parameters the client can view and/or change. The parameters for the high-resolution buffer are shown in Figure 2.

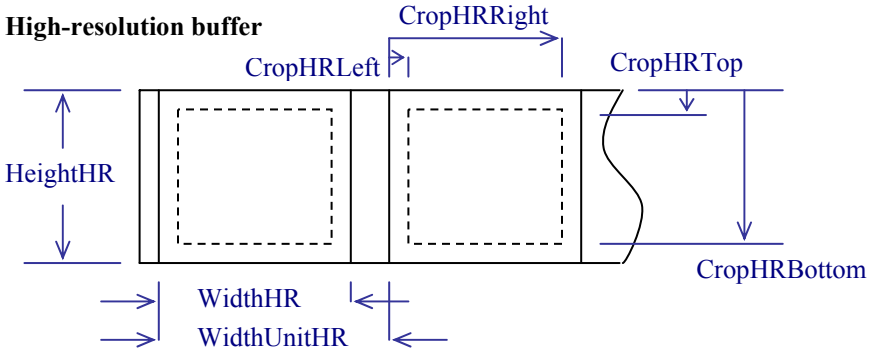


Figure 2. Pre-Scan Framing Parameters

The value of HeightHR cannot be changed by the client. The values of WidthHR, WidthUnitHR, CropHRLeft, CropHRRight, CropHRTop and CropHRBottom can be changed by the client with a call to PutScannerInfoPreFrameUser. The crop values must be such that the cropping window fits within the frame. Since the Pakon scanner will scan images right to the edge and maybe even slightly beyond, it is recommended to do some cropping. Typically, 2 to 2.5 % should be done on each side (4-5% total).

There is also framing parameters for the low-resolution buffer; these are determined by the high-resolution to low-resolution ratio. With a call to GetScannerInfoPreFrame, the client can get the value of HeightLR and determine this ratio.

These parameters can be used to determine how large the high-resolution buffer needs to be for a particular scan. The required space will be HeightHR * WidthUnitHR * number-of-pictures * 2 bytes/channel * 4 channels. Of course, blank space at either end of the film will add to this.

During the scanning and framing operation, TLA looks for good frame lines and sets the frames according to the parameters described above. After this is done, if there are large spaces without a frame, TLA may add frames to fill these spaces. If this is done, you will get framing warnings.

Once a scan is complete, you can call GetPictureCountScanGroup, which will report any warnings from the scan. Among these may be warnings about the framing. If the warnings include SCANW_FRAMING_GOOD, then every picture was framed properly. If the warnings include any other framing warnings, then frames were added by TLA. If you make a call to GetPictureFramingInfo, TLA will report the "risk" - whether a particular picture's frame was found initially, or if TLA added the frame between other frames, or at the end or beginning of the strip. If the risk is high, you may want to ask the operator to review and correct the picture.

A call to GetPictureFramingInfo will also report the post-scan framing parameters. Figure 3 shows these parameters.

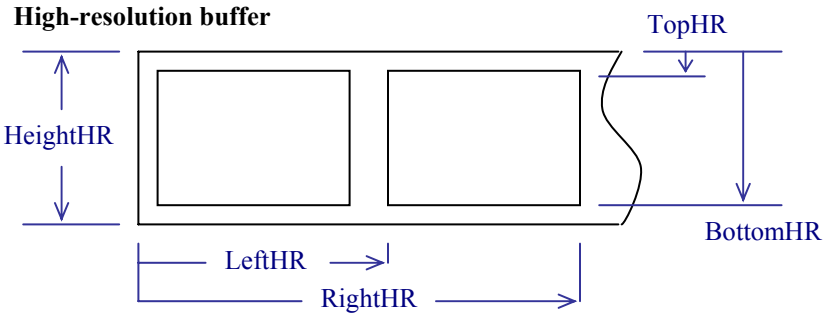


Figure 3. Post-Scan Framing Parameters

Initially, TopHR will be zero and BottomHR will be HeightHR – 1, but all four parameters can be changed with a call to PutPictureFramingUserInfo. TopHR can never be less than zero and BottomHR can never be more than HeightHR – 1. LeftHR and RightHR can range from zero to the length of the buffer. However, the resulting frame must be at least 16 x 16 pixels.

Even if you change a picture's frame with a call to PutPictureFramingUserInfo, you can get the original values with a call to GetPictureFramingInfo. Then, if you want to reset the frame to the defaults, call PutPictureFramingUserInfo with the values you got from GetPictureFramingInfo.

If the default frames are changed in the high-resolution buffer, the frames in the low-resolution buffer are changed at the same time. If you want to find out the actual low-resolution values, make a call to GetPictureFramingUserInfoLowRes.

These parameters can be used to determine how large the client buffer needs to be for a SaveToClientMemory operation. The size will depend on the format, either planar or DIB. For sample code, see the iRequiredBufferSize function in TLAClientDemo.

5.4 Saving Pictures

As you've seen, pictures can be saved to memory or to disk. This section will discuss these options in more detail.

5.4.1 Saving to Memory

When saving pictures to memory using the SaveToClientMemory function, the options include scaling and rotating, as well as specifying the format and numerous control options.

For scaling a picture, you specify the bounding height and width, and the picture will be scaled up or down to fit this bounding rectangle. You can also specify the scaling method: nearest, bilinear or bicubic. These methods are listed from quickest to slowest and from lowest to highest quality. If the control options include SAV_DoNotScaleUp, then the picture will not be scaled up regardless of the size of the bounding rectangle.

For rotating a picture, you specify a rotation parameter, one of the ROTATE_000 enumerations. This may include the MIRROR_L_R parameter to mirror the image left-to-right. If the control options include SAV_UseCurrentRotation, then any rotation parameter you specify will be ignored and the original rotation of the picture will be used. In addition, if the control options include SAV_SizeLimitForSave, then the bounding rectangle may be rotated to best fit the picture's rotation.

There are other control options available, as well. First, you must specify a save size: original, size-for-display, or size-for-saving. This will affect how the picture is fit into the bounding rectangle. See the F235 Programmer's Reference Manual for further explanation.

You can also specify that the pixel information should come from the low-resolution buffer by including the SAV_UseLoResBuffer option. The default is to use the high-resolution buffer. Using the low-resolution buffer will decrease processing time, but also decrease quality. If you want scratch removal to be performed, include the SAV_UseScratchRemovalIfAvailable option. If you want a file header included, specify the SAV_FileHeader option.

Other options will affect the quality of the resulting pictures. If you want raw, 14-bit ("linear transmittance") data from the buffers, do not specify the SAV_UseColorCorrection option. If the SAV_UseColorCorrection option is specified, then the result will be a 12-bit "reference print density" image. These images will be very dark. However, if the SAV_UseColorSceneBalance option is also specified, then the result is an 8-bit standard RGB image.

If you plan to save images using the color scene balance in TLA and you plan to adjust the color, brightness or contrast, then it is better to use the functions PutPictureColorSettingsDifferential or PutPictureColorSettings to adjust the pictures before saving, instead of adjusting them after the save is done. This way, the adjustments will be done with more data in 14-bit space. If the adjustments are done later, then you are working in 8-bit space and clipping can occur.

The picture formats available for the SaveToClientMemory operation include 16-bit planar and 8-bit DIB. If color scene balancing is being done, then the format must be 8-bit DIB. If it is not being done, then the format must be 16-bit planar.

5.4.2 Saving to Disk

Saving to disk using the SaveToDisk function has identical parameters and options as saving to memory, with a few exceptions. The SAV_FileHeader option cannot be used when saving to disk, and the file formats are different. The available file formats are JPEG, bitmap, TIFF and EXIF. If JPEG is chosen, you can also specify a compression factor from 0 to 100. Zero is maximum compression and 100 is minimum compression.

5.5 Custom Color Correction

When saving pictures, you have the option of using color correction (by specifying the option SAV_UseColorCorrection). Pakon has developed a color correction algorithm whose purpose is to remove the results of any crosstalk in the CCD (red in the blue channel, for example) and convert 14-bit linear data into 12-bit logarithmic data. The algorithm uses a lookup table and a matrix filled with values that work well in many cases. However, if you want to customize the color correction, you have the option of using your own lookup table (LUT) and matrix (MAT).

To use your own LUT and MAT, you need to create text files and place them in a specific folder. If these files exist in the folder, TLA will read them when it starts up and use these instead. There are files for color negative and color reversal (positive) film. If you're interested in customizing the color correction, contact Pakon for an explanation of how to do it and obtain samples of custom LUT and MAT files.

Another option is to use the "defaults.ini" file found in the "C:\Program Files\Pakon\TLA COM Server\config\ColorCorrection" folder. Here, you can specify the color, contrast and brightness adjustments to make for any film based on product code. Every time TLA scans film and reads the product code, it will use the adjustments specified in the "defaults.ini" file for that product code. See *Pakon Technical Note 5* for more information.

5.6 Reading MOF from APS

APS film may have magnetic data written on it called MOF (magnetics on film) which holds scene and exposure information, frame titles, date & time, etc.

The F235C scanner has a built-in cartridge loader and MOF reader. You have the option of reading the MOF data or not. If it is read, it is read while the film is being retracted back into the cartridge. This happens automatically after the roll has been scanned. To scan APS film and read the MOF data, call ScanPictures with the *iFilmFormat* parameter set to `FILM_FORMAT_24MM`, and the *iScanControl* parameter including both `SCAN_Use24mmAutoLoader` and `SCAN_Use24mmAutoLoaderMOF`. To scan APS film but not read the MOF data, leave `SCAN_Use24mmAutoLoaderMOF` out of the *iScanControl*. If a roll of APS film does not automatically retract, you can call the function ApsManualRetract.

The MOF data that is read from APS film and reported by TLA includes the film ID, the picture title, the print aspect ratio of each picture, and the date and time each picture was taken. If MOF data is not successfully read for any picture, TLA reports this in the "magnetic data status" parameter of the GetPictureInfo2 function.

Note that MOF data is read only on an F235C scanner and only when scanning APS film from the cartridge loader. If APS film is removed from the cartridge and manually fed in, MOF data cannot be read. The F235*plus* scanner can scan APS film, but it has no cartridge loader, the film must be removed from the cartridge and manually fed in. It also has no MOF reader.

5.7 Additional Features

Besides the features illustrated in the scenarios, there are a few other features worth noting.

If you need to advance the film forward through the scanner without scanning, make a call to AdvanceFilm. You can also move the film backward by specifying a negative speed parameter. **WARNING**: If the end of the film is already inside the scanner and the film is reversed, it's possible for the film to jam. **NOTE**: on some computers, the actual time that the film advances may vary by 10-15%.

If film does get jammed, you can make a call to PutFilmPressureRollerPosition to disengage the rollers and remove the film. It is not necessary to reengage the rollers, they will automatically reengage when the next scan starts.

If you need to switch film formats between 35mm and 24mm, use the Get/PutFilmGuidePosition functions. This will move the film guide to the correct position for the type of film. Make sure there is no film in the scanner prior to moving the film guide. Moving the film guide is not required, TLA will move them automatically any time a scan is performed.

Multiple strips can be scanned with the F235 and they will be combined into one roll. The strips can be 4, 5 or 6 frames long. When scanning multiple strips of film, the scanner continues to scan until either ScanCancel is called, a film timeout occurs, or if 40 frames have been scanned (and the *MaxFilmLength_mm* has been reached). The operator is responsible for insuring there is at least a two-inch gap between strips.

When film is scanned, the scanner attempts to read the DX codes on the film in order to determine the frame numbers and frame names. If there are no DX codes, or if the DX codes cannot be read, then the frame numbers will be negative values. The frame names are determined from the frame numbers and the frame types. Sample values are "3A" or "6P". If DX is not read, all the frame names will be "DX_Error." When calling the SaveToDisk function, the frame names are used as the default filenames. If DX codes were not read, then the call to SaveToDisk will fail because all the filenames are the same. Use the functions Get/PutPictureInfo and Get/PutPictureInfo1 to retrieve or change the default filenames.

If the scanner is having trouble reading DX codes, you can call the FilmTrackTest function. This function will perform tests and calibrations on the film track, maximizing the ability to read DX codes. Film with DX codes must be fed into the scanner for this test. The results can be retrieved with a call to FilmTrackTestResults or by looking at the log file, *PakonFilmTrackTestLog.txt*, found in the "C:\Program Files\Pakon\TLA COM Server" folder.

If the focus is off, you can call the ForceCorrections function to adjust the focus. When making this call, film must be fed into the scanner and the focus correction must be done by itself. Calling ForceCorrections for gain & offset or fixed pattern should not be necessary, these

corrections are done automatically every time you scan with a new configuration. If you've been scanning at the same configuration for a long period, TLA will perform the corrections if the *iDarkPointCorrectInterval* has expired.

When scanning film, it is important for the scanner to use the appropriate motor film speed. Because APS film from a cartridge has more drag on it than 35mm film fed in, there are two sets of motor speed values stored, one for film without drag and one for film with drag. Which one the scanner uses is specified with the SCAN_HasFilmDrag option of the SCAN_CONTROL_000 enumeration. The option is automatically used for APS film from a cartridge, but it can also be specified by the client if the film is being fed from some auto loader that has drag, for example.

When scanning a roll of film with at least 24 frames, and assuming the DX codes can be read, TLA will monitor the motor speed and warn the client if the motor speed is incorrect. This will be reported in the scan warnings (obtained with a call to GetPictureCountScanGroup). If the speed is incorrect, the client should call AdjustMotorSpeed immediately after the scan is complete. (Remember that there are two sets of motor speeds, one for drag, one for no drag. AdjustMotorSpeed will adjust the set that was last used, so it is important to call it right away.)

When monitoring the motor speed, TLA will average the last five scans to determine if the motor speed is correct or not. So, a scan warning about motor speed won't appear until after five scans. If the resolution or film format is changed, then TLA resets the speed monitoring, and another five scans will need to be performed before any motor speed warning is issued.

The scanner's stepper motors may settle during shipment or when moving the scanner. In order to ensure that they can move freely and accurately, the steppers should be exercised whenever the client starts by calling InitializeScanner with the INITIALIZE_MotorSelfTest option and then calling ForceCorrections with the CALIBRATE_ExerciseSteppers option (as well as the CALIBRATE_LampWarmUp option).

Also within InitializeScanner is the option to check for and download any new firmware that exists in the folder "C:\Program Files\Pakon\TLA COM Server\config\Firmware". If the call includes the INITIALIZE_FirmwareUpdate option, TLA will search for any firmware newer than what is in the scanner and attempt to download it automatically. It is **highly** recommended that this option be included. A scanner may not operate properly without the latest firmware.

When calling ScanPictures, you can specify a PreScan. This will perform all the steps necessary for a scan without starting the actual scan. It will adjust the film guides, warm up the lamp, and correct for fixed pattern and gain & offset, if necessary.

TLA has an obstruction detection feature which informs the client whenever significant dust or dirt is detected in the light path. This will be discovered when TLA does a fixed pattern correction (either automatic or manual). If dust/dirt is detected, the client will receive an unsolicited callback with the code `HARDWARE_CB_CLEANING_REQUIRED`.

If errors occur while scanning or otherwise operating the scanner, the LEDs may be turned to yellow or red. With a call to ResetStatusLeds, they can be reset to their initial state, off or green.

6 Troubleshooting

This section discusses the various errors that can occur during the primary operations of the COM Server (TLA) as well as the errors that can occur at anytime (unsolicited hardware callbacks). The purpose is to help programmers and system integrators using the F235 to understand the various error conditions and how to handle them. The descriptions that follow will list when errors can occur and what can be done to try to recover.

6.1 InitializeScanner

6.1.1 Warnings

After calling InitializeScanner, the client can call the GetInitializeWarnings function to see if there were any initialization warnings.

```
INITIALIZEW_EEPROM_CHECKSUM_BAD
INITIALIZEW_EEPROM_BLANK
```

These warnings indicate there was a problem reading the calibration data in the scanner's EEPROM. The data in question includes the Maxwell 3x4 Mats, lens and CCD stepper positions (focus), and film drive motor speed. The scanner will function; however, whatever data was in the host computer's registry will be used to configure the scanner.

6.1.2 Errors

This section discusses the error codes returned either by GetAndClearLastError or by iGetComErrorNumber (see TLAClientDemo for the definition of iGetComErrorNumber). GetAndClearLastError is called after the client receives a callback with the WTO_InitializeError as the indicated operation. iGetComErrorNumber is called if the InitializeScanner function returns an error code in HRESULT. The WTO_InitializeError callback can occur between the time the client calls the InitializeScanner function and the time the client receives the WTO_InitializeProgress callback with the status set to WTP_ProgressComplete.

6.1.2.1 Host computer configuration

```
EC_WorkerThreadCoInitialize
EC_WorkerThreadCoGetInterfaceAndReleaseStream
EC_WorkerThreadClientSignal
EC_WIN_CreateEvent
EC_UnableToCreateWorkerThread
EC_WIN_LoadLibrary (DMLDICELib.dll or PakonImau.dll)
EC_PI_NO_MMX_PROCESSOR
EC_PI_CANT_WRITE_PAKONERRORLOGPI
EC_WIN_* (any Windows error code)
```

These error codes indicate a problem with TLA receiving services from the operating system. A user may try rebooting the computer to clear the error.

6.1.2.2 TLA configuration

6.1.2.2.1 Buffer Drive Configuration Incorrect

```
EC_InvalidParameter "HiResPath"
EC_InvalidParameter, "HiResMegabytesTotal or HiResMegabytesRoll"
EC_PFS_NotEnoughDiskSpace
```

These error codes indicate a problem with the TLA configuration and may be remedied by reinstalling TLA or manually adjusting registry values.

6.1.2.2.2 Driver Configuration Incorrect

```
EC_WIN_FileOpen, \\.\Loopback
EC_WIN_DeviceIoControl
EC_TimeOut
EC_WIN_WaitForSingleObject
EC_WIN_GetOverlappedResult
```

These error codes indicate a problem with the TLA configuration.

- Verify that the scanner is connected to the host and the scanner is powered on
- Cycle power on the scanner and reconnect to TLA
- Cycle power on the computer and reconnect to TLA
- Reinstall TLA.

6.1.2.2.3 Color Correction Configuration Error

```
EC_PI_INPUT_PROFILE
EC_PI_OUTPUT_PROFILE
EC_PI_RPD2ROMM_PROFILE
EC_PI_CR_INPUT_PROFILE
EC_PI_CR_LUTS6
```

These error codes indicate a problem locating the necessary files on the host computer that support the color correction data path. The problem may be remedied by reinstalling TLA or manually adjusting registry values.

```
EC_PI_COMBINE_INPUT_OUTPUT_PROFILE
EC_PI_CR_COMBINE_INPUT_OUTPUT_PROFILE
```

These error codes indicate a problem with TLA combining profile files. The problem may be remedied by reinstalling TLA.

```
EC_FileNotFound, "PakonImau.dll"
EC_DICE_CouldNotLoad
```

These error codes indicate a problem with TLA loading the specified supporting dll file. The problem may be remedied by reinstalling TLA.

6.1.2.3 Programming Error (TLA/F235 Firmware not client)

```
EC_InvalidParameter
EC_StepperAlreadyMoving (TLA/F235 Firmware)
```

6.1.2.4 Communications (TLA/USB/F235 Firmware) Error

```
EC_DRV_InvalidPacketType
EC_DRV_PacketBusy
EC_DRV_FifoOverflow
EC_DRV_PacketChecksumErr
EC_DRV_PacketCommErr
EC_DRV_PacketCmdErr
EC_DRV_PacketHostErrorNoAck
EC_DRV_PacketHostErrorFormat
EC_DRV_PacketHostErrorCkSum
EC_DRV_PacketHostErrorEndPointFormat
EC_DRV_PacketHostErrorEndPointTimeOut
EC_DRV_PacketHostErrorEndPointLength
EC_DRV_PacketHostErrorAlgo
EC_DRV_PacketHostErrorBus
EC_DRV_PacketHostErrorUndefined
EC_DRV_PacketReadWriteMismatch
```

These errors indicate a breakdown in the communications link between the F235 driver on the host computer and the F235 scanner via the USB bus.

- Verify the USB cable is of good quality and proper length
- Try cycling power on the scanner and restarting TLA
- Reinstall the drivers
- Try replacing USB boards

6.1.2.5 Host Error

`EC_CBAdviseNotCalled` - CBAdvise must be called before calling InitializeScanner.

`EC_CBAdviseAlreadyCalled` - You are calling CBAdvise for a second time, it needs to be called only once.

`EC_PicVersion` – Some firmware in the scanner is not the latest.

`EC_SystemInfo` – The computer's processor does not meet Pakon's specs.

`EC_BadSimulatorFile` - You are running the simulator version of TLA, but the scan file is missing or corrupt. Try reinstalling TLA.

`EC_FileNameListEmpty` - ImportFromFile was called with an empty list. You must supply at least one filename.

`EC_ImportedFileColor` – ImportFromFile was called with a file whose number of channels is not one or three.

6.2 ForceCorrections (Gain and Offset)

This section discusses the error codes returned either by GetAndClearLastError or by iGetComErrorNumber (see TLAClientDemo for definition of iGetComErrorNumber). GetAndClearLastError is called after the client receives a callback with the WTO_CorrectionsError as the indicated operation. iGetComErrorNumber is called if the ForceCorrections function returns an error code in HRESULT. The WTO_CorrectionsError callback can occur between the time the client calls the ForceCorrections function and the time the client receives the WTO_CorrectionsProgress callback with the status set to WTP_ProgressComplete.

6.2.1 Errors

6.2.1.1 Host computer configuration

```
EC_WorkerThreadCoInitialize
EC_WorkerThreadCoGetInterfaceAndReleaseStream
EC_WorkerThreadClientSignal
EC_WIN_CreateEvent
EC_UnableToCreateWorkerThread
```

These error codes indicate a problem with TLA receiving services from the operating system. A user may try rebooting the computer to clear the error.

6.2.1.2 Hardware Error

```
EC_ScanLineAcquisition
```

This error indicates a problem with the TLA software communicating with the F235 scanner. The scanner will probably need to be diagnosed and repaired by a technician.

- Verify the USB cable is of good quality and proper length
- Verfiy the USB cable is plugged into the USB 2.0 Port on the host computer
- Verify no other USB devices are connected to the same USB Hub as the F235

6.2.1.3 Communications (TLA/USB/F235 Firmware) Error

```
EC_DRV_InvalidPacketType
EC_DRV_PacketBusy
EC_DRV_FifoOverflow
EC_DRV_PacketChecksumErr
EC_DRV_PacketCommErr
EC_DRV_PacketCmdErr
EC_DRV_PacketHostErrorNoAck
EC_DRV_PacketHostErrorFormat
EC_DRV_PacketHostErrorCkSum
EC_DRV_PacketHostErrorEndPointFormat
EC_DRV_PacketHostErrorEndPointTimeOut
EC_DRV_PacketHostErrorEndPointLength
EC_DRV_PacketHostErrorAlgo
EC_DRV_PacketHostErrorBus
EC_DRV_PacketHostErrorUndefined
EC_DRV_PacketReadWriteMismatch
```

These errors indicate a breakdown in the communications link between the F235 driver on the host computer and the F235 scanner via the USB bus.

- Verify the USB cable is of good quality and proper length
- Try cycling power on the scanner and restarting TLA
- Reinstall the drivers
- Try replacing USB boards

6.2.1.4 Programming Error

`EC_WorkerThreadExists`

This error indicates a scan interface long operation is in progress. The client should ensure that a new long operation in the scan interface is not attempted until the previous long operation is complete.

`EC_InvalidParameter`

This can result from a programming error within either TLA or the client software.

6.3 ScanPictures

This section discusses the warnings and errors that can occur during the ScanPictures operation. Error codes marked with an * apply to ScanPictures without the SCAN_Prescan flag set (an actual scan).

6.3.1 Warnings

After calling ScanPictures, the client can call the GetScanWarnings function to see if there are any scan warnings.

`SCANW_DX_BAD*`

This warning indicates there was a problem reading the DX codes on the film and interpreting them as frame numbers. All of the frames in the scanned roll will have DX_ERROR for a frame number. It may be that the film that was scanned does not have any DX codes or the film is damaged or notched in the DX code area. In this case, the user will be required to manually change the frame numbers (PutPictureInfo1). If the film does have DX coding, the user may optionally rescan the roll. If the warning continues to occur, the scanner film track test can be performed. If the film track test reports a problem, first try blowing compressed air in the area of the track where the four DX sensors are located then rerun the film track test. If the test continues to fail, the track will probably need to be serviced.

`SCANW_FILM_PRODUCT_AND_SPECIFIER_BAD*`

This warning indicates there was a problem reading the DX codes on the film and interpreting them as product and specifier numbers. The product and specifier will be -1 or the product will be 0 and the specifier will be some other number. It may be that the film that was scanned does not have any DX codes or the film is damaged or notched in the DX code area. In this case, the

user will be required to manually change the frame numbers (PutPictureInfo1). If the film does have DX coding, the user may optionally rescan the roll. If the warning continues to occur, the scanner film track test can be performed. If the film track test reports a problem, first try blowing compressed air in the area of the track where the four DX sensors are located then rerun the film track test. If the test continues to fail, the track will probably need to be serviced.

```
SCANW_MOTOR_SPEED_GOOD*
SCANW_MOTOR_SPEED_HALF_PERCENT_SLOW*
SCANW_MOTOR_SPEED_ONE_PERCENT_SLOW*
SCANW_MOTOR_SPEED_HALF_PERCENT_FAST*
SCANW_MOTOR_SPEED_ONE_PERCENT_FAST*
```

These warnings indicate that TLA has detected the motor speed is off, which means the pictures may be squashed or stretched. The client can do any of the following:

- Ignore the warning and proceed with the current scan as is. The client will likely continue to get scan warnings on subsequent scans.
- Call AdjustMotorSpeed to update the scanners motor speed settings and proceed with the current scan as-is. The client should no longer get scan warnings on subsequent scans.
- Call AdjustMotorSpeed to update the scanners motor speed settings and rescan the roll. The client should no longer get scan warnings on subsequent scans.

```
SCANW_FRAMING_IN_MIDDLE*
SCANW_FRAMING_AT_END*
SCANW_FRAMING_AT_BEGINNING*
SCANW_FRAMING_BAD*
```

These warning give an indication of how the framing algorithm performed. No warnings means the framing algorithm detected a good sequence of frame lines and little guessing was involved. If warnings are present there is some risk associated with the framing method employed. The lower the warning is in this list, the less accurate the framing method.

Individual pictures can be interrogated to determine the risk (see GetPictureInfoFraming in the Programmer's Reference Manual). If the risk is high, a user should review the frame and make any necessary corrections such as re-framing the picture or deleting the picture if it is not a valid picture.

```
SCANW_MAX_FILM_LENGTH_EXCEEDED*
```

This warning indicates that the TLA scan buffers were filled prior to the detection of the end of film and the scan was cancelled. This can be caused either by scanning a filmstrip longer than the MaxFilmLength_mm or by the filmstrip jamming in the film track. MaxFilmLength_mm has a default value of 1600 and can be increased with a call to PutScannerInfo001.

6.3.2 Errors

If any of the following errors occur, the client can make a call to GetPictureCountScanGroup and compare the counts to the previous counts to find out if there are any new pictures that may be salvaged from the scan.

6.3.2.1 Host computer configuration

```
EC_WorkerThreadCoInitialize
EC_WorkerThreadCoGetInterfaceAndReleaseStream
EC_WorkerThreadClientSignal
EC_WIN_CreateEvent
EC_UnableToCreateWorkerThread
EC_QueryInterface
EC_CoMarshalInterThreadInterfaceInStream
EC_WIN_SetProcessWorkingSetSize
EC_WIN_VirtualAlloc
EC_WIN_VirtualLock
EC_WIN_VirtualUnlock
EC_WIN_VirtualFree
EC_WIN_WaitForSingleObject
EC_WIN_ResetEvent
EC_WIN_SetFilePointerEx*
EC_WIN_FileWrite*
EC_MemoryNew
EC_TimeOut*
```

These error codes indicate a problem with TLA receiving services from the operating system. A user may try rebooting the computer to clear the error.

6.3.2.2 Hardware Error

In addition to the hardware errors described in this section, the hardware errors discussed in the Hardware Callbacks section need to be considered when creating a client error handling strategy during a scanning operation.

```
EC_ScanLineAcquisition
```

This error indicates a problem with the TLA software communicating with the F235 scanner. The scanner will probably need to be diagnosed and repaired by a technician.

- Verify the USB cable is of good quality and proper length
- Verify the USB cable is plugged into the USB 2.0 Port on the host computer
- Verify no other USB devices are connected to the same USB Hub as the F235
- Call for service

```
EC_LampWarmUpFailure
```

This error indicates the light level did not stabilize within the expected amount of time. The scan will not start.

- The user can retry the scan
- The user can replace the lamp
- Call for service

```
EC_StepperDidNotStop
```

This error indicates that a stepper motor in the scanner was commanded to a new position by TLA and TLA timed out waiting for the notification back from the scanner that the stepper is in position.

- The user can try rescanning
- Try shutting down the software and the scanner, and restarting
- Call for service

6.3.2.3 Communications (TLA/USB/F235 Firmware) Error

```
EC_DRV_InvalidPacketType
EC_DRV_PacketBusy
EC_DRV_FifoOverflow
EC_DRV_PacketChecksumErr
EC_DRV_PacketCommErr
EC_DRV_PacketCmdErr
EC_DRV_PacketHostErrorNoAck
EC_DRV_PacketHostErrorFormat
EC_DRV_PacketHostErrorCkSum
EC_DRV_PacketHostErrorEndPointFormat
EC_DRV_PacketHostErrorEndPointTimeOut
EC_DRV_PacketHostErrorEndPointLength
EC_DRV_PacketHostErrorAlgo
EC_DRV_PacketHostErrorBus
EC_DRV_PacketHostErrorUndefined
EC_DRV_PacketReadWriteMismatch
```

These errors indicate a breakdown in the communications link between the F235 driver on the host computer and the F235 scanner via the USB bus.

- Verify the USB cable is of good quality and proper length
- Try cycling power on the scanner and restarting TLA
- Reinstall the drivers
- Try replacing USB boards

6.3.2.4 Programming Error

```
EC_WorkerThreadExists
```

This error indicates a scan interface long operation is in progress. The client should ensure that a new long operation in the scan interface is not attempted until the previous long operation is complete.

```
EC_InvalidParameter
```

This can result from a programming error within either TLA or the client software.

```
EC_ScannerNotInitialized
```

This error indicates that InitializeScanner was never successfully completed before the ScanPictures call was made. The client should ensure that InitializeScanner is successful before calling any other scan interface methods.

```
EC_NoFixedPatternCorrection
EC_PFS_PartitionSelected
EC_PFS_FileLengthNotSet
EC_PFS_NullFilePointer
EC_PFS_InvalidPointer
EC_PFS_FilePointerDeleted
EC_PFS_NotLastStripInFile
EC_PFS_WritePastEOF
EC_NoHighResolutionBuffer
```

These error codes indicate a programming error within TLA

6.3.2.5 Other Errors

```
EC_DRV_RingTailOverflow
EC_ProcessedRingTailOverflow
EC_DRV_CannotFindStartOfScanLine
EC_DRV_FifoOverflow
EC_DRV_LostSync
```

These error codes indicate there was a breakdown in the communication of scan lines from the scanner to TLA. The user can retry the scan.

```
EC_TooManyRolls
```

This error code indicates that a scan was attempted when the TLA buffer drive was already full. One or more rolls will need to be removed from the buffer drive and the scan will have to be restarted.

```
EC_BufferDriveMegabytesRollTooSmall
```

This error indicates that TLA was configured for a maximum film length (see PutScannerInfo001 in the Programmer's Reference Manual) that results in a TLA buffer drive file that is larger than the maximum buffer drive megabytes per roll. The HiResMegabytesRoll is set during TLA installation or manually using registry settings. The maximum film length will need to be reduced, the roll will need to be scanned at a lower resolution or the HiResMegabytesRoll will need to be increased.

```
EC_NoStripsScanned
```

This error indicates that the scanner timed out before film was detected by the CCD. The user should restart the scan and insert the film in a timely manner.

```
EC_NoFilmTimeOut
```

This error indicates that the scanner timed out before film was detected by the CCD. This could be caused either by the operator failing to insert the filmstrip before the timeout period or by the filmstrip jamming in the film track prior to passing through the light source. The default for the film timeout is 60 seconds and can be changed with a call to PutScannerInfo000.

6.4 SaveToClientMemory

6.4.1 Warnings

There are no warning messages for save operations.

6.4.2 Errors

This section discusses the error codes returned either by GetAndClearLastError or by iGetComErrorNumber (see TLAClientDemo for definition of iGetComErrorNumber). GetAndClearLastError is called after the client receives a callback with the WTO_SaveError as the indicated operation. iGetComErrorNumber is called if the SaveToClientMemory function returns an error code in HRESULT. The WTO_SaveError callback can occur between the time the client calls SaveToClientMemory and the time the client receives the WTO_SaveProgress callback with a progress of WTP_ProgressComplete.

6.4.2.1 Programming Errors

```
EC_WorkerThreadExists
```

This error indicates a save interface long operation is in progress. The client should ensure that a new long operation in the save interface is not attempted until the previous long operation is complete.

```
EC_InvalidIndex
EC_InvalidParameter
EC_InvalidParameter : Color Adjustments without Image Correction
EC_InvalidParameter : File Format Incorrect for FastUpdate8BitDib
EC_InvalidParameter : iBoundingWidth_or_iBoundingHeight
EC_InvalidParameter : Image Correction without Color Correction
EC_InvalidParameter : iRotation
EC_InvalidParameter : iSaveControl
EC_InvalidParameter : iScalingMethod
```

An illegal value was passed in to TLA. Check the values and try again.

```
EC_NoPicturesOrStrips
```

There are no pictures or strips to save (client programming error).

```
EC_NoWorkerThreadForMultipleSaveToMemory
```

If saving multiple pictures, the client should have set the *bUseWorkerThread* flag.

EC_NoClientMemoryBuffer

The client has requested a save to memory without first adding a client buffer to TLA.

EC_TimeOut

During the save of the second or later picture in a multiple picture save to memory, TLA timed out waiting for a client buffer to be added to TLA.

EC_InsufficientMemoryForSaveToMemory

TLA is trying to save a requested image to the client buffer but the client buffer is too small. The client must ensure that there is enough memory for the save operation.

```
EC_ClientMemoryBufferInUse
EC_DICE_CodeErr
EC_DICE_InProgress
EC_DICE_InternalErr
EC_DICE_InvalidParameter
EC_DICE_InvalidThread
EC_DICE_NotInProgress
EC_DICE_QueueFull
EC_DICE_Unknown
EC_InvalidMemberVariable : m_pBufferLR
EC_InvalidMemberVariable : m_uiWidth
EC_InvalidMemberVariable : m_uiHeight
EC_InvalidMemberVariable : m_iImageType
EC_InvalidMemberVariable : m_uiColorsPerPixel
EC_ImageNotPlanar
EC_PI_ERROR_SETTING_LOCKBEAM
EC_PI_INVALID_FILE_FORMAT
EC_PI_INVALID_ORIENTATION
EC_PI_KNOWN_EXCEPTION_RECORDED
EC_PI_MIN_MAX_RANGE_BLUE
EC_PI_MIN_MAX_RANGE_BRIGHTNESS
EC_PI_MIN_MAX_RANGE_CONTRAST
EC_PI_MIN_MAX_RANGE_GREEN
EC_PI_MIN_MAX_RANGE_RED
EC_PI_UNKNOWN
EC_PI_UNKNOWN_EXCEPTION_RECORDED
```

These errors indicate a TLA programming error. Reboot and/or reinstall TLA. If problems persist, have the scanner repaired.

6.4.2.2 Host computer configuration

```
EC_QueryInterface
EC_CoMarshalInterThreadInterfaceInStream
EC_UnableToCreateWorkerThread
EC_WorkerThreadClientSignal
EC_WorkerThreadCoGetInterfaceAndReleaseStream
EC_WorkerThreadCoInitialize
EC_WorkerThreadStartTimeout
EC_PFS_FilePointerDeleted
EC_PFS_InvalidPointer
EC_PFS_NullFilePointer
EC_PFS_PartitionSelected
EC_PFS_ReadPastEOF
EC_WIN_SetFilePointerEx
EC_WIN_FileRead
EC_WIN_ResetEvent "EventSaveToClientMemory"
EC_WIN_WaitForSingleObject
EC_WrongByteCount
EC_MemoryNew
EC_PI_MEMORY
EC_DICE_MemErr
```

These error codes indicate a problem with TLA receiving services from the operating system. Try rebooting the computer to clear the error.

6.5 SaveToDisk

6.5.1 Warnings

TBD

6.5.2 Errors

TBD

6.6 ForceCorrections (Focus)

6.6.1 Warnings

TBD

6.6.2 Errors

TBD

6.7 FilmTrackTest

6.7.1 Warnings

None

6.7.2 Errors

`EC_DXPotsWillNotAdjust`

A pot was set to zero; the one at zero will be in the log file.

`EC_DXNoFilmFound`

User didn't insert film or it wasn't detected by the sensors.

`EC_DXNoGoodBrightSpotFound`

TLA couldn't find a spot on the film to adjust the pots on.

`EC_DXAdjustingPotsForGoodSingal`

A pot was set to zero; the one at zero will be in the log file.

`EC_DXBadSwing`

Voltage swing of sensor was below 80 counts. Return code will report which one as well as the log file.

The regular mode of TrackTest (bDxPotsAdjust = FALSE) will only return EC_DXNoFilmFound or EC_DXBadSwing. The other errors are for calibration of the pots (bDxPotsAdjust == TRUE).

6.8 Hardware Callbacks

This section discusses the error codes returned by an unsolicited callback with the WTO_HardwareError as the indicated *ulOperation* parameter. In this case, the programmer can test the *lStatus* parameter for further error code information as described below. If a call to GetAndClearLastError is made after the client receives the above callback, the error code returned will be EC_HardwareFault.

6.8.1 Errors

These errors can occur anytime after TLA initialization completes.

`HARDWARE_CB_HOST_FAULT`

If the *lStatus* parameter indicates HARDWARE_CB_HOST_FAULT, there is a problem with the USB circuit board in the scanner and the scanner should probably be repaired.

HARDWARE_CB_DX_FAULT

If the *lStatus* parameter indicates HARDWARE_CB_DX_FAULT, there is a problem with the DX circuit board in the scanner and the scanner should probably be repaired.

HARDWARE_CB_LAMP_FAULT

If the *lStatus* parameter indicates HARDWARE_CB_LAMP_FAULT, there is a problem with the lamp board. Further information about the error may be found in the *lStatus* parameter as follows.

HARDWARE_CB_LAMP_VIN_LO_WARNING
HARDWARE_CB_LAMP_VIN_LO_ERROR

If the *lStatus* parameter indicates HARDWARE_CB_LAMP_VIN_LO_WARNING, scanning can continue but a service technician should be called to troubleshoot the error.

If the *lStatus* parameter indicates HARDWARE_CB_LAMP_VIN_LO_ERROR, the lamp board shuts powers off and a service technician would be required to troubleshoot and diagnose the error before scanning can continue. If the error occurred during the scan, the operator may try to salvage the scan by processing the order normally to see how many pictures from the roll were captured prior to the lamp board shutting down.

HARDWARE_CB_LAMP_TEMP_HI_WARNING
HARDWARE_CB_LAMP_TEMP_HI_ERROR

If the *lStatus* parameter indicates HARDWARE_CB_LAMP_TEMP_HI_WARNING, scanning can continue but a service technician should be called to troubleshoot the error.

If the *lStatus* parameter indicates HARDWARE_CB_LAMP_TEMP_HI_ERROR, a service technician would be required to troubleshoot and diagnose the error before scanning can continue. If the error occurred during the scan, the operator may try to salvage the scan by processing the order normally to see how many pictures from the roll were captured prior to the lamp board shutting down.

HARDWARE_CB_LAMP_BURN_OUT_ERROR

If the *lStatus* parameter indicates HARDWARE_CB_LAMP_BURN_OUT_ERROR, the operator will need to replace the lamp. If the error occurred during the scan, the operator may try to salvage the scan by processing the order normally to see how many pictures from the roll were captured prior to the lamp burning out.

HARDWARE_CB_CCD_FAULT

If the *lStatus* parameter indicates HARDWARE_CB_CCD_FAULT, there is a problem with the CCD circuit board in the scanner and a service technician would be required to troubleshoot and diagnose the error before scanning can continue.

```
HARDWARE_CB_MOTOR_FAULT
```

If the *lStatus* parameter indicates HARDWARE_CB_ MOTOR_FAULT, there is a problem with the Motor Board. The motor that is having the trouble can be determined by testing the lStatus parameter for one of the following.

```
HARDWARE_CB_MOTOR_STEPPER_CCD
HARDWARE_CB_MOTOR_STEPPER_LENS
HARDWARE_CB_MOTOR_FILTER_WHEEL
HARDWARE_CB_MOTOR_FILM_GUIDE
```

Any one of these errors indicates a problem that may require a service technician to repair. Possible failures include:

- Faulty sensor associated with the motor
- Faulty motor
- Mechanical interference with a motor

```
HARDWARE_CB_FILM_EMULSION_DOWN
HARDWARE_CB_FILM_TAIL_FIRST
```

If the *lStatus* parameter indicates either HARDWARE_CB_FILM_EMULSION_DOWN or HARDWARE_CB_FILM_TAIL_FIRST, the film was probably inserted into the scanner incorrectly. The scan can proceed with either of these conditions, however, the image quality will be slightly degraded and the image orientation and order may be incorrect.

```
EC_WIN_DeviceIoControl
```

This error indicates the driver has stopped communicating with the scanner. The most likely cause for this is either the USB cable was disconnected or the scanner was powered off. The client should finish processing any rolls already in the scan or save group and then release the TLA COM server. The operator should then verify power is on and the cable is connected. Then try reconnecting to TLA.