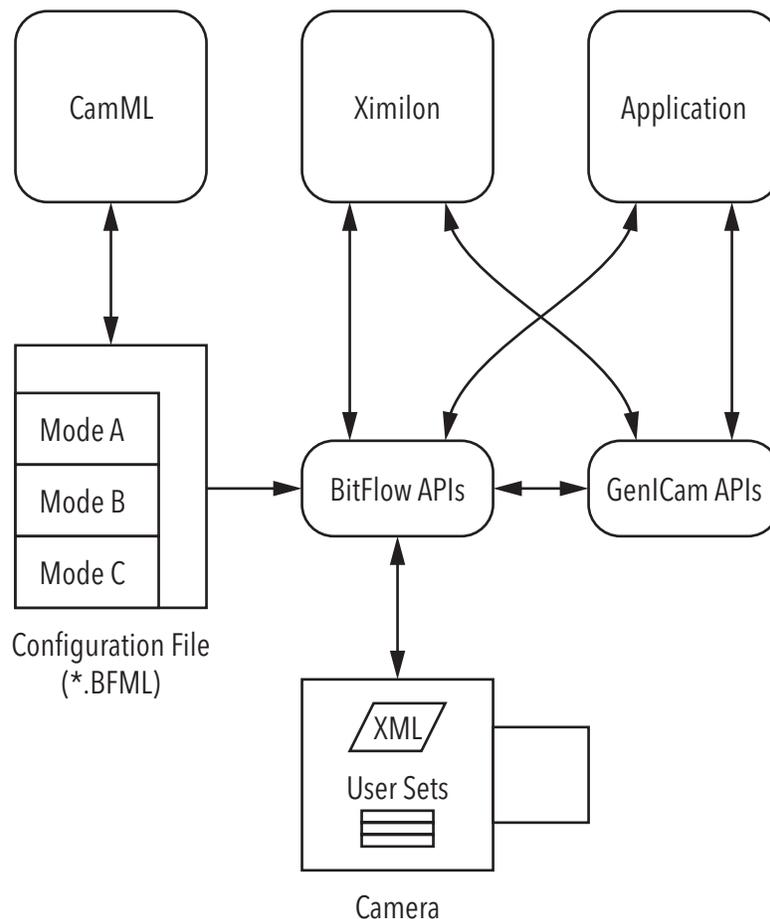**BitFlow**™

Controlling CoaXPress Cameras from The BitFlow SDK Tools, Configuration File and APIs
Revision 1.1

Introduction

The BitFlow SDK offers a number of different methods for programming and controlling a CoaXPress (CXP) camera connected to a BitFlow CXP frame grabber. While number of workflows might seem confusing, the design goal was to provide a methodology that meets the needs of a wide variety of users. The purpose of the this document is to explain all the CXP tools that are available in the BitFlow SDK, and how their flexibility is actually quite powerful, and can be molded to meet each different application's needs.

Components

The image above shows the major CXP camera control blocks. Below is a brief description, they are explained in more detail later.

> CamML - This the GUI application for editing configuration files (*.BFML)
> Ximilon - BitFlow's GenICam based interactive camera control utility
> Application - A user's application which needs to both control the camera and acquire images form the camera.
> Configuration File (*.BFML) - These are BFML files that can program both the frame grabber and the camera (if needed)
> BitFlow APIs - There are three different APIs that can be used to control the camera: 1. BitFlow's Low level register API. 2. Generic GenICam API. 3. BitFlow's super simple BFGTLUtil API. Bit-Flow also provides an extremely CPU efficient acquisition API.
> GenICam APIs - This consist of both a camera control API and an acquisition (streaming) API. Both are camera and frame grabber agnostic.
> Camera - There are two relevant components here. 1. The camera's internal XML file which describes the camera's control parameters. 2. User Sets - these are non-volatile memory banks in the camera that can be programmed by the user to exactly define all of the camera's parameter when the camera powers up

All of these components work together in different ways. Each customer's workflow may use all or just some of these components in their application development and possibly a different set when the deploy their application to their customers. It's important to understand how each component works individually, and how they interact with each other at various times of an application life cycle.

GenICam and GenTL

GenICam is powerful set of tools for connecting machine vision cameras (or devices) to computers (host). It supports a wide range of hardware and protocols. Come components of the CoaXPress specification are built on GenICam, we discuss the relevant parts in this document. The full details of GeniCam are beyond the scope of this manual. Further information can be found on the EMVA's website.

One component of GenICam is the "GenICam Transport Layer" or GenTL. GenTL is a software definition that is used to interface cameras or devices to host application software in a hardware agnostic way. The camera could be a GigE camera connected directly to a PC, or a CXP camera connected to a frame grabber plugged into a PC. The software access to these camera is provide by a GenTL *producer*. The application that uses the camera is called a GenTL *consumer*.

The BitFlow SDK provides a GenTL producer. which provides access in generic way to all BitFlow frame grabbers installed in a system. A GenTL consumer "consumes" or interfaces with a GenTL producer, they plug together in a standard, well defined way. Any GenTL consumer can "plug in" to any GenTL consumer. This means that any GenTL producer application written by any organization can plug into the BitFlow GenTL producer.

The GenTL specification defines two interfaces, one for device control and one for data streaming. Bit-Flow's GenTL producer supports both control and streaming. BitFlow also offers its own streaming API, which is easier to use and more efficient than the GenTL streaming interface. By offering both streaming interfaces, customer can chose which best suits their application.

Ximilon

Ximilon is generic GenICam based camera control utility. Technically it is a GenTL *consumer*. It interfaces with the GenTL *producer* automatically installed by the BitFlow SDK. Ximilon's main purpose is to provide complete access to all of a cameras control parameters. Ximilon can also stream live video from any connected CoaXPress camera.

It's important to understand that Ximilon is a generic camera control application. There are no specific camera controls built into Ximilon, it reads information from the camera and creates a graphical user interface (GUI) that offers controls for all of the connected camera's parameters. The GUI present in Ximilon will be different for each camera that it is used with, though there will be many similarities as all cameras have common features. Changes made by the user in the GUI will immediately be sent to the camera. Any changes to the camera parameters will be updated automatically in the GUI (this is done by polling which is user programmable).

HOW XIMILON READS CAMERA PARAMETERS

This section goes into detail about how Ximilon (or any GenTL consumer) reads the programmable parameters from the camera. This information is not strictly needed in order to work with CXP cameras and BitFlow frame grabbers, so feel free to skip this section if you are not interested.

Every CXP camera has an on-board XML file the conforms to a GenICam schema. This file contains all the information about all the camera's parameters than can be programmed by the user. All GenICam cameras are "register based". This means that the camera is controlled by writing values to register addresses in the camera. The XML file describes all of the addresses that corresponds to all of the camera's features. In short, the XML file is a kind of guide that correlates friendly names and parameter ranges to camera registers and addresses.

When Ximilon is first run with a new camera, it reads some bootstrap registers in the camera, which provide information on the location and size of the camera's XML file. Ximilon then downloads the XML file from the camera and writes it to cache on the local hard disk. It then creates a node based tree that provides ordered access to all of the camera's parameters.
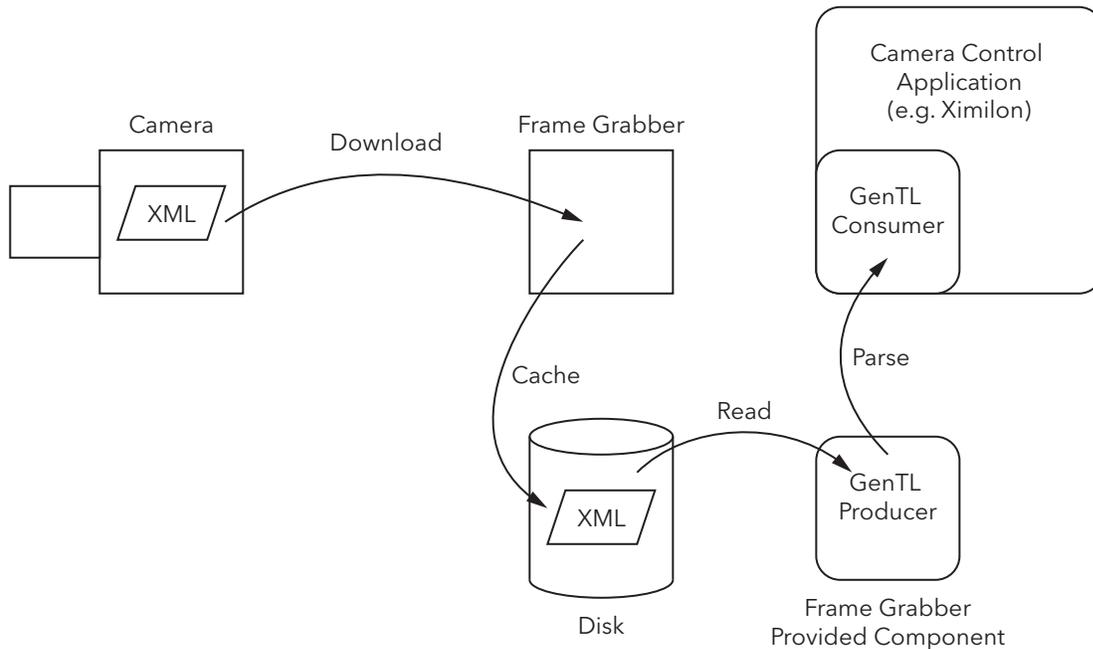
USING XIMILON

Ximilon is fairly easy to use. It's beyond the scope of this document to provide a detailed descriptions of all of its features, however it does have a built in manual available (see the Ximilon > Help menu command). Generally you navigate the feature tree to get to the parameter you are interested in. Depending on the type of feature, there will be a slider, button, drop-down list or other widget that lets you modify the parameter. When the change is made in the GUI, the new value for the parameter is sent immediately to the camera.

It is possible and also quite helpful to run a viewing application simultaneously with Ximilon. This lets you make changes to the camera and immediately see the results in the video. You can run another application, such as BitFlowPreview, or you can run Ximilon's built in image viewer.

It's important to understand that when the camera is acquiring and sending images to the frame grabber, certain camera parameters will be locked out. When this happens you will see "RO" next to the locked parameter. If you need to modify a locked parameter, you will need to stop acquisition (which you can do from within Ximilon), change the parameter and re-start acquisition.

Camera User Sets

When a camera powers up, all of its internal parameters must be set to some value. Normally there is a bank of non-volatile memory which holds these parameters when the camera is off. When the camera is powered up, the values from these memories are written to all control registers. These memory banks are usually called "user sets". Most cameras have a "factory set", which holds the default values for all parameters. The factory set usually can not be overwritten. Then there is usually one or more user sets that can be freely programmed by the user. The idea is that once you have the camera set up exactly as you need it, you can save its state to a user set. The next step is to tell the camera to power up from your user set. This way you don't have to re-program the camera every time the camera is power is cycled.

User sets are a very powerful part of the camera control workflow, they should be used to program most of the camera's parameters whose values are different than the factory default. The important thing to understand, is that any camera parameter that you do not anticipate changing as your application runs, should be stored in the camera's user set. This minimizes the number of camera parameters that your application will need to program.

CamML - the BFML File Editor

CamML is an application for editing BFML files. BFML files are just XML files with a different extension, and they can be edited in any text editor. However, CamML is specifically designed to conform to the BFML file schema (it will only output valid BFML files). More importantly, CamML will display all the possible options for any parameter, eliminating the need to look up the choices for each parameter. In addition, there are a number of GUI widgets which make understanding and creating BFML files much easier.

CamML has a built in video viewer window which can be opened at any time to test your current BFML file mode. Any changes made to the GUI, will be sent to the camera and/or frame grabber as soon as you open the viewer. This includes the various CXP commands that are discussed in the "BFML Files" section.

BFML Files

BFML files are configuration files that are used to configure BitFlow's frame grabbers and optionally configure the attached camera. BFML files are actually XML files that are using a custom BitFlow created schema. BFML files can be edited in any text editor, but are more easily modified using BitFlow's CamML utility. BitFlow ships a large number of BFML files with its SDK. There is generally a BFML file for each major camera model. BFML files contain multiple modes, each mode can be used to customize some aspect of the system configuration. For example, one BFML mode might set the system up for free running acquisition, while another mode might set it up to use a trigger.

The BFML file can configure all aspects of the frame grabber, essentially it can program all of the frame grabbers registers to any value. In addition, it can also send commands to the camera. **This means that by switching BFML modes, you can reprogram both the frame grabber's parameters and the camera's parameters so that they will work with each other.** In essence, the BFML modes can be thought of as system mode, changing the BFML mode changes the frame grabber/camera system to be in a certain

desired mode. This idea can be extended further into the system as the BFML file can also change the frame grabbers I/O programming. For example, changing from a free running to a triggered mode, might not only reprogram the frame grabber and the camera, but could also set up the board's pulse generator to fire a strobe at the correct time based on the trigger.

BFML DEVICE COMMANDS

BFML file Device Commands were introduced in SDK 6.40. Device Commands can be used to program both the frame grabber and the camera. We will add the ability to send commands to other types of devices in a future release. Device Commands can be issued either before setup (e.g. stop camera acquisition before reprogramming the camera) or after setup (e.g. set the camera's ROI). For commands sent to CXP cameras, the protocol can either be GenICam GenTL based, or low level register based. The GenTL commands are much easier to use and are generally "future proof". In other words, a camera company might update the firmware and change the address of a control register, but the GenTL command to set a certain feature to a certain value, will always work. These camera control commands, both GenTL and register based, are the same as you would use if you were programming the camera from software via one of the camera control APIs (see below). The table below lists all of the possible types of Device Commands

Interface	Purpose	Protocols
Camera GenTL	Program camera parameters	GenICam
Camera CXP Register	Program camera parameters	Low level CXP
BitFlow Register	Program frame grabber parameters	Internal
BitFlow Control	Execution delay, future expansion	TBD

Device Commands are sent in order they are listed in the BFML file. Device commands "invoke" = "before_setup" are sent before the board is initialized and configured. Device commands "invoke" = "after_setup" are sent after the board is initialized and configured, these are the very last step of the board initialization process.

Note: Prior to SDK 6.40, CXP cameras were programed via the BFML "cxp_regs_before" and "cxp_regs_after" features. These features only support camera control via low level register access. As discussed, register access is more complicated to set up and is subject to obsolescence issues if the camera's firmware is updated. These BFML file features are deprecated as of SDK 6.40 and BFML Device Commands should be used for all current and future camera programming needs.

Note: Despite what is said in the note above, Device Commands do support low level register access. This type of access is provided for two reasons. First, if there is a feature in a camera that is not mapped to a GenTL parameter, but is available via registers, this mechanism provides a means of programming it. Second, some customers may not want the overhead of GenICam in their application, but still need to program their cameras, they can use Device Command register access. If no GenTL commands are used, the GenICam system will not be loaded. This provides the ability to build and deploy an application with no GenICam dependencies.

THE AUTO-CONFIGURE OPTION IN BFML FILES

Traditionally BFML files program the frame grabber with a fixed image resolution and bitdepth. This must match the cameras setting in order to successfully acquire images. This setup can make experimenting with various camera setups time consuming as the BFML file must be updating every time one of the major camera parameters is changed. Starting with SDK 6.40 we have added the ability for the BFML file to the system auto-configure to the camera's parameters. Thus the BFML file mode will not have a fixed image resolution or bitdepth, but instead will tell the driver to inquire from the camera what it's settings are and then configure the frame grabber to match. Auto-configure is enabled in the BFML file by using the word "Default" in the parameter where you want the camera to determine the value. The table below lists all the frame grabber parameters the can be set to auto-configure.

Parameter	Meaning
xsize	Number of pixels per line
ysize	Number of lines per frame
bitdepth	Number of bits per pixel

If any of these parameters are set to the value "Default" in the main Camera features section or in one of the modes in the section "mode_roi", then the camera's value for these parameters will be used. The SDK ships with some BFML files that are set up this way, which in many cases is all that is needed in order to acquire from a camera.

The following shows how these parameters can be set in CamML

Application Programming Interfaces

The BitFlow SDK provides three camera programming APIs for use form your application software. The first is a low level camera registers access API. The second is the GenICam API. The third is BitFlow's BFGTLUtil API.

CXP REGISTER ACCESS

BitFlow's CXP register access functions are quite simple. They provide methods for reading and writing both single camera memory location or a block of memory. The functions, enumerated in the table below, are described in detail in the BitFlow SDK Reference manual.

Function	Description
BFCXPReadReg	Read a register from a CoaXPress device
BFCXPWriteReg	Write a register to a CoaXPress device
BFCXPReadData	Read a block of memory from a CoaXPress device.
BFCXPWriteData	Write a block of memory to a CoaXPress device.

The functions can be called at any time to modify the behavior of the attached camera. The only difficulty with using these functions is that user must know the address of the particular feature of interest. Some cameras have well documented address maps, others are not documented at all. Further, some cameras have parameters require complicated access (i.e. multiple reads/writes) in order to modify a setting. The advantage of these functions is that they are very simple to use and do not require the overhead of the GenICam infrastructure. This means your application does not have to be deployed with the GenICam DLLs and other support files.

GENICAM CAMERA CONTROL

The GenICam camera control system is based around a standard API for camera control and video streaming. The system works in conjunction with an in-camera XML file that enumerates the camera's parameters and corresponding register map. The GenICam API is well documented and the details are beyond the scope of this white paper. The basic principle is that the frame grabber manufactures provides a GenTL *producer*, which is a custom DLL that exposes the standard API. Users then write or purchase a GenTL *consumer*, which is an application that calls functions in the frame grabber's producer. Behind the scenes, the producer handles downloading the XML file (caching is locally) while GenICam components handle translating the high level commands to low level register commands. Finally the frame grabber's functions are called to send the commands "up the wire" to the camera. Ximilon is an example GenTL consumer.

While the GenICam system sounds complicated, once a framework for a producer is built, it's fairly easy to use in an application. There are skeleton frameworks available that can get you started (contact Bit-Flow customer support for more information). There are many advantages to using GenICam: no need to worry about potential register map changes caused by firmware updates; the code can be similar or the same between different camera models (and even different frame grabbers); complicated camera commands (e.g. indirect register accesses) are automatically handled; the API can provide upper/lower parameter limits, documentation, access control (i.e. locking features when camera is running) for each parameter. The only real downside is that the learning curve and the overhead of the GenICam components.

Note: the GenICam API is provided by the GenICam committee. The BitFlow SDK does install some of the binary components and is used in some of our applications. Please contact the GenICam committee for downloads, documentation and support.

BFGTLUTIL

GenICam is a high level API, which manufacturer agnostic control and acquisition from a wide variety Machine Vision components. It somewhat unwieldy and has a fairly steep learning curve. BitFlow has watched many of its customers struggle with the GenICam API even though they might need just use just a few camera control commands. There was clearly a need for a wrapper around GenICam that makes it very easy to use, but does not give up any camera control functionality.Thus BFGTLUtil was born. It provides simple functions that take as input standard parameter names (SFNC) and values to set the parameter to. All the complexity of downloading, cache, parsing the XML file and navigating the device trees is done for the customer. It's really just as simple as "Set exposure time to 100 msec".

Below is a short section of code that illustrates how easy BFGTLUtil is to use. In the example a camera node (name "ExposureTime") is opened. Then the node is written from local string.

Note: Returns values are ignore here to simplify the code. In general, the returned value should always be checked.

```
BFGTLDev hBFGTLUtilDevice = BFNULL;
BFGTLNode hNode;
BFCHAR NodeName[MAX_STRING];
BFCHAR ValueBuf[MAX_STRING];
BFSIZET ValueBufLen;
```

```
// Open the camera
BFGTLDevOpen(hBoard, &hBFGTLUtilDevice);

// Open the node to program
strcpy(NodeName, "ExposureTime");
BFGTLNodeOpen(hBFGTLUtilDevice, NodeName, &hNode)

// put the desired exposure in a string
strcpy(ValueBuf, "100");

// Write ValueBuf string to node
ValueBufLen = sizeof(ValueBuf);
BFGTLNodeWrite(hNode, BFGTL_NODE_VALUE_STR, ValueBuf, ValueBufLen);

// close the node
BFGTLDevClose(hBFGTLUtilDevice);
```

COMPARING APIS

The table below shows compares the three APIs

	BFCXPRegXXX	GenICam	BFGTLUtil
Complexity	Low	High	Medium
Learning Curve	None	Steep	Low
Access all features	Most	All	All
Supports Acquisition	No	Yes	No
Requires GenICam Libraries	No	Yes	Yes
Works with any camera make andmodel	No	Yes	Yes
Requires camera register map	Yes	No	No
Camera Firmware updates can break existing code	Yes	No	No
Works on other manufacturer's frame grabber	No	Yes	No
Documentation	BitFlow SDK Manual	GenICam Committee	BitFlow SDK Manual

Workflows

All of the components discussed above can be used in various combinations in order to setup and control a CXP camera connected to a BitFlow frame grabber. The workflow that you use will very much depend on the details of your application. The section describes some possible workflows and the pros and cons of each.

PROGRAM THE CAMERA WITH XIMILON, USE THE AUTO-CONFIGURE BFML FILE

This is the simplest way to work with a CXP camera. Using the auto-configure BFML file, "Generic-CXP.bfml" causes all applications to automatically configure the frame grabber to match the camera. Anytime a change is made in Ximilon, just re-running an application will update the frame grabber to match the camera. If you need to set up I/O or the on-board timing generator, you can chose one of the triggered modes in this BFML file or modify one of the modes to match your I/O needs.

Once the camera is in the mode you need, you can save the camera's parameters to one of its user sets, and tell the camera to power up from that user set.

Pros

> Very easy to configure
> All Camera settings are done in the Ximilon interactive GUI
> Configuring a user set means the camera will always start up in the exact same way
> Frame grabber I/O, timing sequencers and other options can still be setup via the BFML file

Cons

> Camera configuration is static (of course it can be changed via software, see workflows below)

USING BFML FILE MODES TO CHANGE THE CAMERA ON THE FLY

The technique provides some flexibility for changing the camera's configuration on the fly from software, but does not involve the complexity of directly programing the camera for one of the available APIs. The workflow here is to work with Ximilon and CamML to create a BFML file with all the modes that you will need in your application. The camera can be programmed to power up in one of these modes by saving it to a user set (then program the camera to power up from this set). The BFML file modes can then be used to change just the camera settings needed to switch between the different system configurations.

For example, you might want to always use an image resolution of 1024 x 1024 in a camera that has 2048 x 2048 sensor. Using Ximilon, program the camera to this ROI, and save these settings to the cameras power up user set. Now in your application, you might want to trigger the camera some times, and have it free run other times. Using CamML you can create two modes, one for triggered and one for free run. One mode will use the device commands to put the camera in triggered mode, the other mode will use the device commands to put the camera in free running mode. Both modes will set the resolution to 1024 x 1024 (or just use the "Default" auto-configure flag for xsize and ysize).

In your application, you can switch between BFML modes using the CiCamModeSet() function. This function will program the camera with any Device Commands that are part of the given mode. It will also program the frame grabber with any features of the mode.

Pros

> No direct camera programming in your application software
> Changing modes is very simple using CiCamModeSet()
> Flexible enough to modify any camera parameter on the fly

Cons

> Still requires some knowledge of using CamML and building up BFML modes
> In some cases, it will take more time to switch modes than to directly modify a camera parameters from software

USING APIS TO CHANGE THE CAMERA'S PARAMETERS ON THE FLY

This technique involves use one of three available APIs to modify the camera's parameters. Generally it still makes sense to set up a camera's user set to put the camera in a base mode that serves your application needs. You can change camera parameters as needed by your application software. This technique works using any of the available APIs discussed above.

In some cases this can still be fairly simple. For example, you may only need to modify the camera's exposure time and frame rate on the fly. You can program the camera exactly as you need (with a default exposure time and frame rate), then save the settings to a power up user set. Then when your application is running, you can use one of the available APIs to modify the exposure time and frame rate on the fly, as needed. With some cameras this can be as simple as writing an exposure value to a register address in memory using BFCXPWriteReg(). In other situations, it may warrant building a full blown GenTL producer into your application and use the GenICam to change the camera's settings. In the middle (complexity-wise) is using BitFlow' BFGTLUtil. This API provides access to all camera parameters just as the GenICam API, but is much easier and simpler to use.

Pros

> Provides the ability to dynamically change any camera parameter at any time
> Can still take advantage of user sets and/or BFML file modes to program camera into a require startup state
> Using the GenICam API can support more then one camera model and/or make

Cons

> Camera control APIs can be somewhat complex: BitFlow's CXP API requires a camera register map, the GenICam API has a learning curve and requires third party binary components. BFGTLUtil provides the best of both, but still needs some coding.

OVERVIEW

The table below shows the various camera control methods and the tools that are needed to implement them.

Method	User Sets	BFML File Mode	CXP API
Program the camera with Ximilon, use the auto-configure BFML file	X		
Using BFML file modes to change modify the camera on the fly	X	X	
Using APIs to modify the camera's parameters on the fly	X		X

The workflow that makes the most sense for your application really depends on what type of camera parameters you need to change, how often you need to change them, and how complex the process of

changing them is. Hopefully the information in this document will help you figure out the methodology that will work best for you application.

As always, we are here to help. Feel free to contact BitFlow's customer support at any time.

email: support@bitflow.com
phone 781-932-2900

Happy programming!