CFX Manager™ Software API Reference Guide

For use with all CFX Real-Time PCR Detection Systems

Catalog #184-5096
#185-5196
#184-4096
#185-4096
#185-5200
#184-5384
#185-5484

BIO-RAD

Bio-Rad Laboratories, Life Sciences Group

CFX Manager™ API Reference Guide

Revision:1.3 Date: November 21, 2014

Legal Notices

TABLE OF CONTENTS

ABOUT THIS DOCUMENT

SCOPE

This document describes the Bio-Rad CFX Manager™ Application Programming Interface (API). It provides an overview of the API and gives its technical specifications. It also provides an overview of the associated CFX Manager API Source Code Examples Kit (API Examples Kit), and includes answers to frequently asked questions about the API.

The objectives of this document are to:

- Provide an overview and technical specifications of the CFX Manager API
- Provide an overview of the CFX Manager API Examples Kit
- Answer frequently asked questions about the CFX Manager API

AUDIENCE

This document is intended for software developers who want to integrate the capabilities of Bio-Rad CFX Manager into custom software solutions. It assumes that the reader has a working knowledge of the Microsoft WCF framework and XML, and is familiar with Bio-Rad CFX Manager.

ORGANIZATION

This document is organized into two chapters and one appendix:

- **Chapter 1: Introduction -** Provides an introduction to the CFX Manager API. Includes overviews of the API schema and the API Examples Kit
- **Chapter 2: Technical Specifications -** Provides technical specifications of the API. Includes important details related to the configuration and implementation of API clients
- **Appendix A: Frequently Asked Questions -** Answers frequently asked questions about working with the CFX Manager API

CHAPTER 1 INTRODUCTION

This Chapter provides information on the following topics:

- Introduction to the Bio-Rad CFX Manager API Service
- Overview of the CFX Manager API Schema
- Overview of the CFX Manager API Examples Kit

1.1 INTRODUCTION TO THE BIO-RAD CFX MANAGER API

The **CFX Manager API** is a Windows Communication Foundation (WCF) service published by the Bio-Rad CFX Manager application. It allows local or remote client applications to programmatically manage the basic instrument control and monitoring functions of CFX Manager, such as opening and closing motorized lids, starting and stopping PCR runs, and obtaining instrument status.

Supported API operations are:

- **Register** – Access to the API is limited to a single client through a registration procedure
- **Unregister** – Registered clients need to unregister before closing in order to permit other clients to access the API
- **Instrument Status** – Provides detailed status of all CFX Systems connected to the hosting computer
- **Open Lid** – Open the mechanized lid of a CFX System
- **Close Lid** – Close the mechanized lid of a CFX System
- **Flash LED** – Cause the LED indicator on a CFX System to blink for approximately 10 seconds
- **Run Protocol** – Start a protocol run on a CFX System with specified plate, data processing, and reporting options
- **Stop Run** – Stop (cancel) a running protocol
- **Pause Run** – Pause a running protocol
- **Resume Run** – Resume a paused protocol
- **Generate Report** – Generate a report from a specified data file and report template
- **Shut Down** – Shut down a server mode instance of CFX Manager that was started by the client application

All API requests and responses are communicated via XML documents adhering to a schema that is provided with CFX Manager. The schema is designed to support status-driven, as opposed to event-driven, client application architectures. All instrument control requests return the current status of the target instrument, along with any associated error information. The Instrument Status operation returns the detailed status of all connected instruments, and is designed to be polled intermittently as a means of detecting instrument status changes.

For example, after running a protocol, a CFX System's status will transition from *Running* to *Processing*, and then to *Idle*. By polling the instrument's status via the Instrument Status request, a client application can detect changes in instrument status in granular detail, and choose to ignore or react to specific status transitions based on application specific requirements.

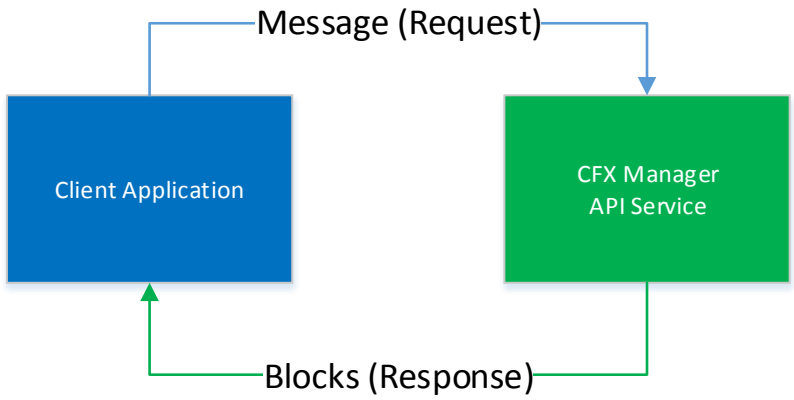
By integrating instrument control and status monitoring capabilities into one simple interface, the CFX Manager API provides a level of flexibility and reliability that can't be matched by systems that rely on asynchronous callback events or third party event engines to provide instrument status updates.

The remainder of this chapter presents an overview of the CFX Manager API Schema, which defines the operational characteristics of the API service; and an overview of the API Examples Kit, which demonstrates the use of the API and includes source code libraries that can be used to reduce the effort of integrating the API into custom solutions.

1.2 OVERVIEW OF THE API SCHEMA

The CFX Manager API schema is provided as part of the CFX Manager Software installation. It is located in the CFX Manager Installation directory (typically ***C:\Program Files (x86)\Bio-Rad\CFX***) at: ***SupportFiles\CfxComSchema.xsd***.

The root elements defined by the schema are **Message** and **Blocks**. Message instances are API service requests, and Blocks instances are API service responses. The following figure illustrates one complete transaction between a client and the API service:



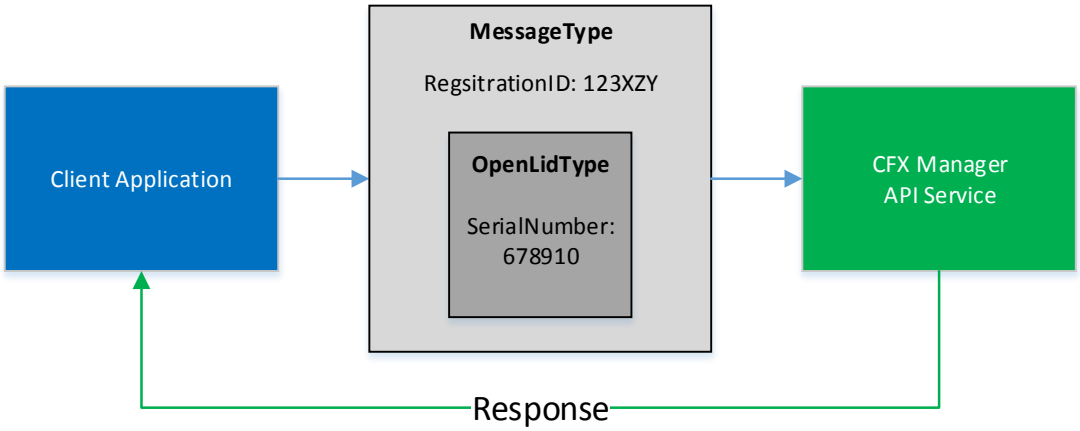
The Message/Blocks terminology has its roots in the content of the information being exchanged, but can be confusing when discussing the details of the schema because the schema contains multiple similarly named elements. So for the remainder of this chapter, we will substitute Request/Response for Message/Blocks, with the understanding that a "Request" is a Message instance and a "Response" is a Blocks instance.

1.2.1 REQUESTS

API service requests are constructed from the **MessageType** element, which consists of two sub-elements:

1. **RegistrationID** – A unique registration ID associated with the client connection. Obtained by the Register Service request.
2. **[Operation Element]** – One of a choice of API operation types defined in the schema, which are named after their associated API operations, and which contain sub-elements corresponding to the operation's parameters.

For example, the request to open a CFX System's motorized lid consists of a MessageType instance populated with the previously obtained registration ID and an **OpenLidType** instance as its Operation Element. The OpenLidType includes one sub-element, which is the serial number of the targeted CFX System. The following figure illustrates an Open Lid request:



Note that when the API receives a request, it will always attempt to respond to the request immediately. In the case of instrument control and report generation requests, it will initiate the operation and return a response that includes any immediate errors along with the current status of the target instrument (if applicable) – it *will not* block and wait for the requested operation to complete. This designed absence of blocking in API responses is what allows client applications to perform continuous status polling, and effectively monitor and control multiple instruments asynchronously.

1.2.2 RESPONSES

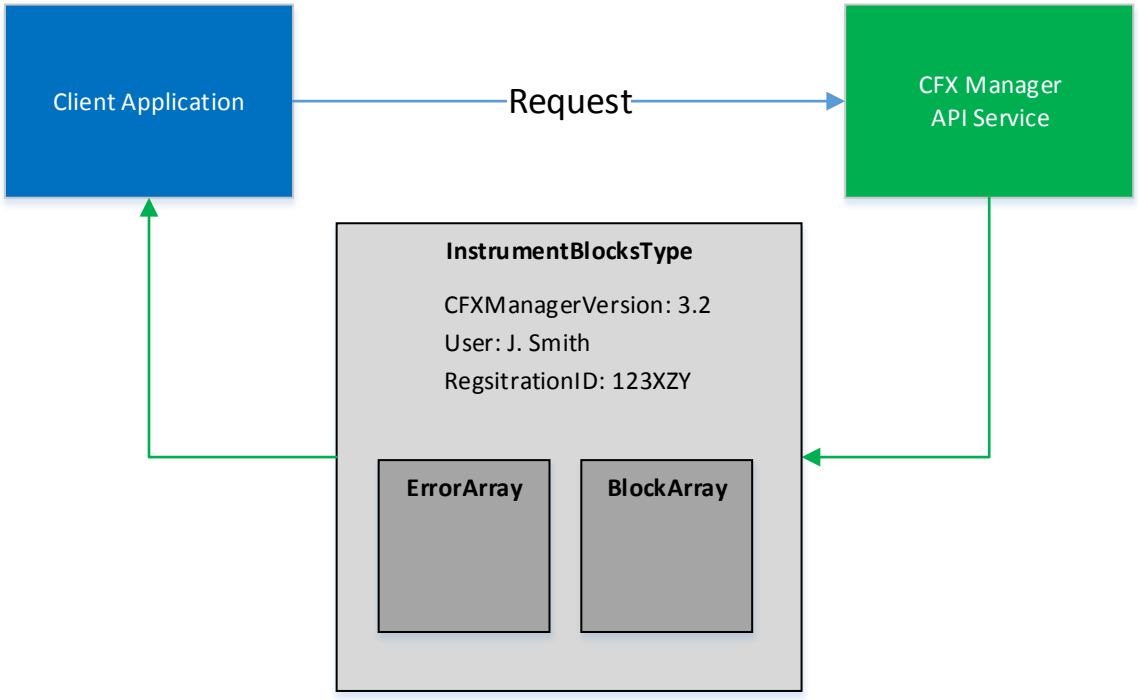
API service responses are constructed from the **InstrumentBlocksType** element, which consists of five sub-elements:

1. **CFXManagerVersion** – The software version of the CFX Manager instance hosting the service.
2. **User** – The current CFX Manager User Name, indicating which CFX Manager User preferences are in force.
3. **RegistrationID** – The unique registration ID associated with the client connection.
4. **ErrorArray** – An array of **ErrorType** elements (elements defined by the schema to convey error information)
5. **BlockArray** – An array of **InstrumentBlockType** elements (elements defined in the schema to convey instrument status).

For instrument control requests (open lid, close lid, start protocol, etc.) the BlockArray will always contain one element, containing the target instrument's status information. For the Instrument Status request, the BlockArray will contain one element for each connected CFX System, or will be null if no instruments are connected.

The ErrorArray will be null unless a server-side error occurs during processing of the request. In a stable operating environment, server-side errors will be extremely rare, and generally indicate a systems level failure.

The following figure illustrates an InstrumentBlocksType response:



The following sections describe the internal structures of the ErrorType and InstrumentBlockType elements, which are used to populate the ErrorArray and BlockArray elements in an InstrumentBlocksType response.

ErrorType Element

The ErrorType element is the structure defined in the schema to represent error information. It contains four sub-elements:

1. **DateTime** – A DateTime value used to associate the error with an instance in time.
2. **ErrorCode** – An integer value used to identify the error with a unique error code. For CFX System device errors, these should be interpreted as hexadecimal values.
3. **ErrorDescriptionCurrentCulture** – A string containing a human-readable description of the error in the language of the currently localized culture.
4. **ErrorDescriptionInvariantCulture** – A string containing a human-readable description of the error in English.

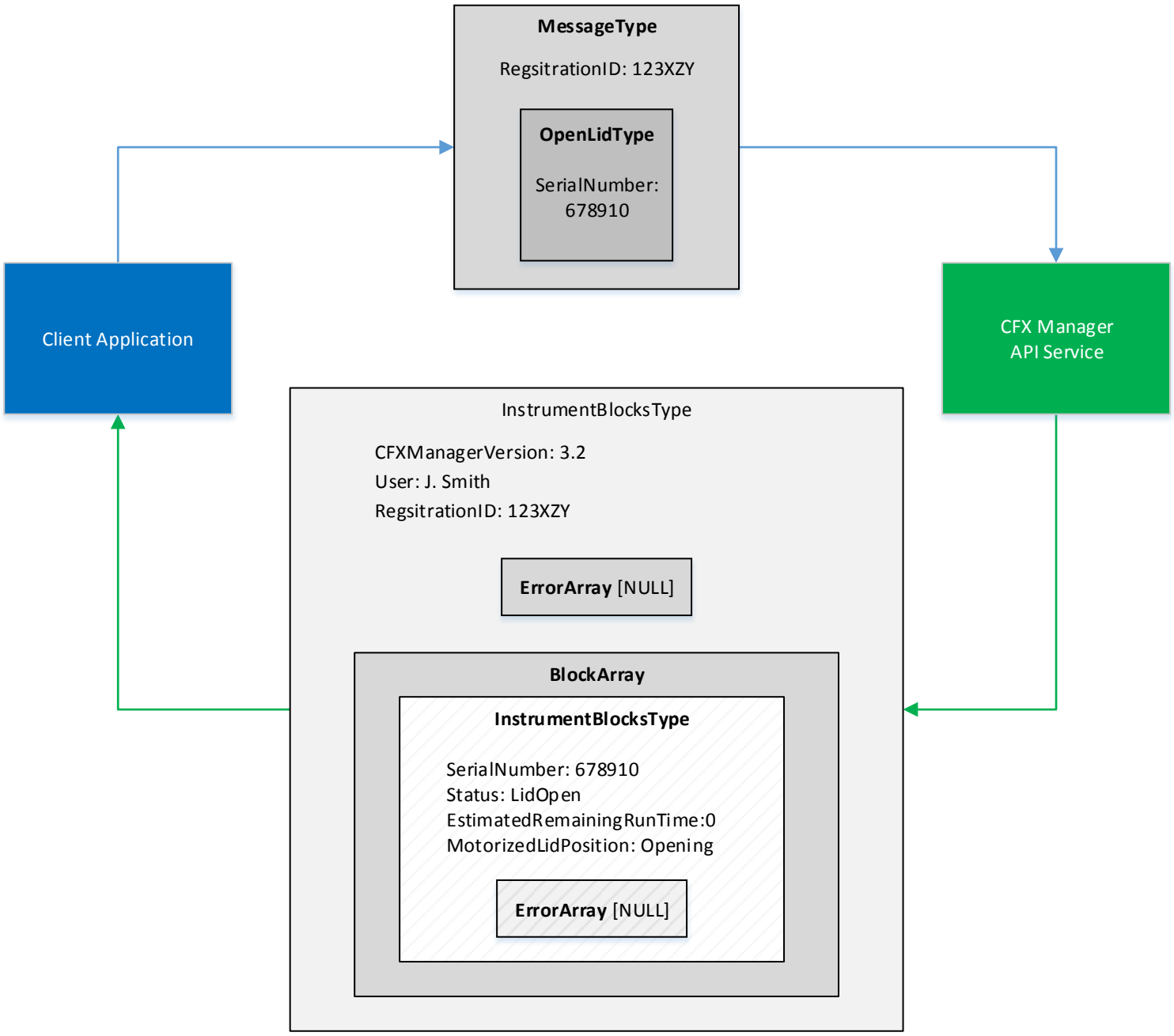
The ErrorType element is used by the API to represent server side errors, as described earlier, and also to describe instrument errors, as noted in the following section.

InstrumentBlockType Element

The InstrumentBlockType element is the structure defined in the schema to represent CFX System instrument status. It contains 26 sub-elements representing various pieces of information about the identity and current status of a specific CFX System. Many of those elements, however, are for highly specialized or Bio-Rad internal uses. The following five elements are what a typical client application of medium complexity will generally require:

1. **ErrorArray** – Zero or more ErrorType elements corresponding to any currently reported instrument errors.
2. **SerialNumber** – The serial number of the CFX System base. This is the unique system identifier used as a parameter for instrument control requests.
3. **Status** – The operational status of the instrument. Possible values are defined in an enumeration found in the schema.
4. **EstimatedRemainingRunTime** – When the instrument is running a protocol, this will be the estimated time to completion of the protocol.
5. **MotorizedLidPosition** – A more granular status of the instrument's motorized lid than is available from the instrument status. Useful for automation applications that need to know the real-time position of the lid. Possible values are defined in an enumeration found in the schema.

The following figure depicts a complete OpenLid transaction in terms of schema elements, assuming no server errors and no instrument errors occurred during the operation:



1.2.3 SUPPORTED AND UNSUPPORTED SCHEMA ELEMENTS

In addition to the supported API operations and their associated types and enumerations, the schema includes elements that are for Bio-Rad internal use. Such elements are collectively referred to as unsupported elements and should not be referenced or instantiated by client applications.

Chapter 2 specifies all supported schema elements, which include all supported API operations and their associated types and enumerations. Any schema elements that are not included in that specification are unsupported.

1.2.1 SUMMARY

The CFX Manager API schema is provided as part of the CFX Manager Software installation. It defines the XML structures used to communicate with the CFX Manager API service.

Service requests are defined in the schema by Message documents, which contain a unique registration ID and a sub-element that specifies the requested operation and contains operation-specific parameters. For instrument control and report generation requests, the API will initiate the requested operation and immediately return a response, rather than blocking until the operation completes.

Service Responses are defined in the schema by Blocks documents. They include the registration ID, CFX Manager Version and user information, error reporting information, and one or more instrument status elements.

The schema includes both supported and unsupported content. All supported content is specified in chapter 2. Any content not specified in Chapter 2 is unsupported and should not be referenced or instantiated by client applications.

1.3 OVERVIEW OF THE API EXAMPLES KIT

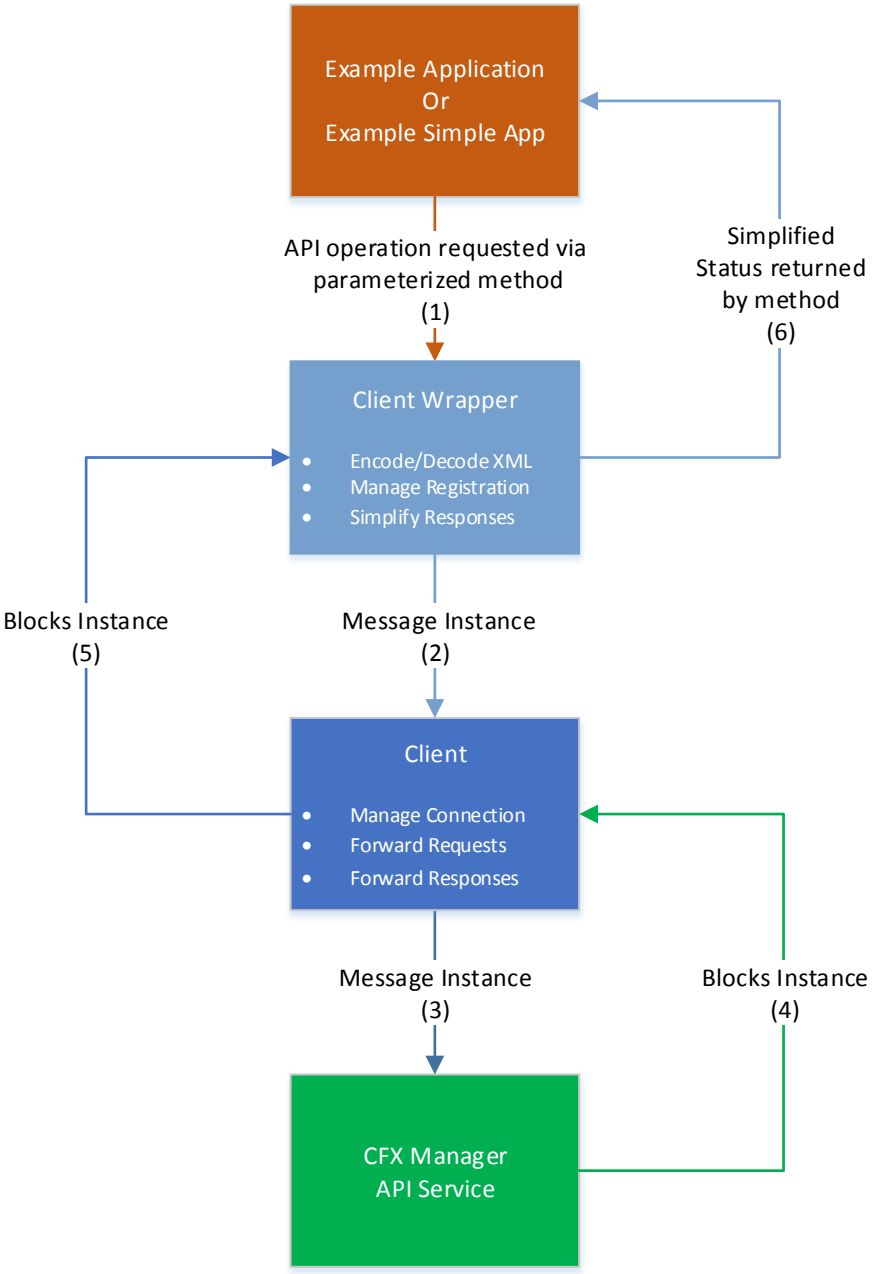
The CFX Manager API Source Code Examples Kit that accompanies this document is a fully annotated collection of sample code that demonstrates how to interact programmatically with the CFX Manager API service. It is a Visual Studio 2010 solution written in C#. Even if you will not be working in C#, the Examples Kit will prove useful by clarifying some of the more subtle aspects of creating and managing CFX Manager API clients, and by demonstrating how to implement application level logic using the API's status-driven architecture. If you will be working in C#, or in another .Net based language, the examples kit provides a set of working libraries that can be integrated directly into your solution. The complete Examples Kit consists of eight custom source files and four auto-generated source files, totaling approximately 1,300 lines of code.

1.3.1 THE API EXAMPLES KIT SOLUTION

The CFX Manager API Examples Kit is a Microsoft Visual Studio 2010 Solution written in C# and requires .Net Framework 4.0. It is composed of four interacting Visual Studio projects:

1. **Example_Client** – Implements a CFX Manager API service client.
2. **Example_Client_Wrapper** – Implements a middleware wrapper around the API service client. Encapsulates and manages client registration and XML encoding and decoding. Provides simplified consumer-facing instrument status and error reporting structures. Provides utilities to manage CFX Manager in Server Mode. Presents API operations as simple parameterized methods.
3. **Example_Application** – Implements a fully functional demonstration application that interfaces with the Example_Client_Wrapper library to control and monitor multiple CFX Systems simultaneously.
4. **Example_Simple_App** – Implements a more basic demonstration application that interfaces with the Example_Client_Wrapper library to control and monitor a single CFX System.

The following figure depicts the top-level architecture of the API Examples Kit Solution in terms of the schema structures discussed in the previous sections:



Note that in this architecture client registration tracking and the details of the API service schema, as well as the actual client implementation, are hidden from the application level logic.

Examples Kit Solution Source Files

The following tables summarize the source files that comprise the Examples Kit solution:

Source Files for Project: Example_Application		
File	**Origin**	**Description**
app.config	Customized instance of the template generated by the Microsoft ServiceModel Metadata Utility Tool (Svcutil). Refer to Chapter 2 for details of the required customization of this file.	Application configuration file containing the serviceModel configuration required by the API client. Comments in the file describe the required customizations of the Svcutil generated template.
FormStartRun.cs	Bio-Rad Example Code	A designer form used to garner protocol run parameter information from the user.
MainForm.cs	Bio-Rad Example Code	A designer form implementing the Example Application's core functionality.
Program.cs	Visual Studio generated Program.cs customized to implement unhandled exception processing.	Implements the main execution thread (entry point) of the Example application.

Source Files for Project: Example_Client		
File	**Origin**	**Description**
CfxComSchema.cs	Generated by the Microsoft Service XML Schema Definition Tool (xsd)	C# classes corresponding to the schema elements, used to instantiate schema runtime objects.
CFXManagerClient.cs	Bio-Rad Example Code	Implements a client of the API service
SOCFXCommandService.cs	Generated by the Microsoft ServiceModel Metadata Utility Tool (Svcutil)	The service model code for the API service. This code must be generated in .Net 4.0 in order to be compatible with the Examples Kit

Source Files for Project: Example_Client_Wrapper		
File	**Origin**	**Description**
CFXManagerClientWrapper.cs	Bio-Rad Example Code	Middleware wrapper encapsulating the Example_Client library and presenting consumers with a simplified API interface.
CFXManagerUtilities.cs	Bio-Rad Example Code	Utility library providing the ability to detect whether an API compatible version of CFX Manager is installed on the local host, and to start CFX Manager in server mode.

Source Files for Project: Example_Simple_App		
File	**Origin**	**Description**
app.config	Customized instance of the template generated by the Microsoft ServiceModel Metadata Utility Tool (Svcutil)	Application configuration file containing the serviceModel configuration required by the API client. Comments in the file describe the required customizations of the Svcutil generated template.
MainForm.cs	Bio-Rad Example Code	A designer form implementing the Example Simple App's core functionality.
Program.cs	Visual Studio generated Program.cs	Implements the main execution thread (entry point) of the Example Simple app.

1.3.1 INSTRUMENT STATUS POLLING

The CFX Manager API service is designed to support a status polling client application architecture. This means that instead of defining callback events or using third party frameworks to generate events, the API simply provides a service call that returns the current status of all connected instruments at any given time. Client applications can then decide for themselves which status changes need to be processed in a given context. This requires that the client application implement the logic to poll for instrument status, and to detect and interpret state changes. The example applications illustrate how this can be accomplished with a minimum amount of coding effort.

The following sections present an overview of the status polling techniques implemented by the example applications.

Using QueryInstrumentBlocksType to Poll for Status

The key to the API's status polling architecture is the instrument status query implemented by the QueryInstrumentBlocksType schema element. The response to a request whose operation element is QueryInstrumentBlocksType includes the current status of all currently connected CFX Systems. By sending QueryInstrumentBlocksType requests to the API at regular intervals, the client application can easily maintain a self-updating real-time snapshot of the status of all connected instruments. Changes to instrument status across a polling interval can then be interpreted as required to signal higher level events, such as protocols starting and completing, or motorized lids being completely opened or completely closed.

The example applications perform their own status polling and interpretation logic, based on interactions with the Example_Client_Wrapper middleware library. An alternative architecture would be to integrate the status polling logic into the middleware and provide an always-up-to-date status table and set of custom status change events to the application layer. In either case, the status polling and status change interpretation logic will always adhere to the same basic algorithm:

1. Poll the API for status updates using the QueryInstrumentBlocksType operation at some regular interval.
2. At each update, compare each instrument's previous status to its current status.
3. For each instrument, take appropriate action if the transition [previous status] -> [current status] is of interest.

The InstrumentBlockType elements returned by the QueryInstrumentBlocksType operation include a variety of real-time information about each connected CFX System, so that the precise definition of "status" in this algorithm will depend on the functional requirements of the client application. For most applications, however, the focus of status polling will be the *operational status* of the CFX Systems being controlled (refer to the following section for more information on polling for operational status). Other commonly tracked status components are MotorizedLidPosition, which reflects the real-time position of a CFX System's motorized lid, and EstimatedRemainingRunTime, which gives the approximate time to completion for a currently running protocol.

The Example_Application project demonstrates how status polling can be used to control and monitor multiple instruments simultaneously. It allows the arbitrary execution of instrument control and report generation operations, and at the same time updates the status of all connected instruments every second, logging any status changes to its console. It also performs status change interpretation for important events, such the start and completion of protocol runs.

The Example_Simple_App project demonstrates the simpler case of controlling and monitoring a single instrument, without performing event level interpretations of status transitions.

Using the StatusType Enumeration to Interpret Operational Status Changes

The operational status of a CFX System is defined as one of 13 possible operational states that can be reported by the instrument. Transitions between states constitute changes in *operational status*, and can be used to identify high level events that may be of interest to client applications. For example, the operational status of a CFX System transitioning from state *Idle* to state *Initializing* indicates that the system is coming up to temperature in preparation for starting a PCR run.

The operational status of CFX Systems is reported by the API as the *status* sub-element of the InstrumentBlockType. Its value is of type StatusType, which is an enumeration representing the 13 possible operational states. By monitoring for changes in the value of the *status* sub-element across polling intervals, client applications can easily detect and interpret operational status changes.

Table of CFX System Operational Status States

The following table describes the CFX System operational status states, and identifies state transitions that client applications may find of interest. Refer to method *ProcessStatusChange* in the Example_Application project file *MainForm.cs* for an example of how to perform status change interpretation.

CFX System Operational Status States		
StatusType Value	**Description**	**Transitions of Interest**
Idle	The lid is closed and the system is initialized and not performing any operations OR the system is transitioning between operational states	The device's primary "ready" state. Also, a device may or may not "flash" idle while transitioning between other states, so it should be ignored when looking for transitions between two non-idle states. Processing->Idle indicates that the data processing performed by the device at the end of a run has been completed. Note that this does not imply that CFX Manager has completed post-run data analysis and report generation, which may take up to a minute to complete after the device has processed the data.
Paused	The device has been paused while running a PCR protocol	Running->Paused indicates a running protocol has been paused. Paused->Running indicates a paused protocol has been resumed.

CFX System Operational Status States		
StatusType Value	**Description**	**Transitions of Interest**
Running	The device is running a protocol	Initializing->Running indicates that the requested protocol has started. Running->Processing indicates that a protocol has completed running and the associated data are being processed. Running->Paused indicates a running protocol has been paused. Paused->Running indicates a paused protocol has been resumed.
Infinite Hold	A protocol is holding the device at a fixed temperature indefinitely	This is associated with manual workflows and should not be encountered in API controlled contexts. Running->Infinite Hold indicates the sample will be held at its current temperature until manual intervention occurs. Infinite Hold->[Any state] indicates that manual intervention has transitioned the device from Infinite Hold to the current state.
Synchronizing	The device is performing initialization after being powered on or connected to the host computer	Synchronizing->Idle indicates that the device is ready for use. Synchronizing-> Lid Open indicates that the device is initialized but that its lid is open.
Calibrating	The device is being calibrated	For internal Bio-Rad use. Will not be encountered in API controlled contexts.

CFX System Operational Status States		
StatusType Value	**Description**	**Transitions of Interest**
Lid Open	The device lid is open or in the process of opening	[Any State]->Lid Open indicates the device lid is opening or open. Lid Open->Idle indicates the lid is closing or closed and the device is otherwise idle. To distinguish between opening and completely open, and between closing and completely closed, track the MotorizedLidPosition sub-element of the InstrumentBlockType. Transitions to and from Lid Open and non-idle states are unusual but possible. For example, opening a lid while running a protocol will cause the protocol to pause, and while the operational status will be Lid Open, the state of the protocol run will be paused. Similarly, when the lid is closed in that context, the protocol will resume, and the associated transitions will be Lid Open->Paused and Paused->Running. Refer to CFX Manager and CFX System documentation for additional information.
Initializing	The device is performing initialization tasks in preparation for starting a protocol	Idle->Initializing indicates the device has responded to a start protocol request and is preparing to start running the protocol. Initializing->Running indicates that the requested protocol has started.
Waiting Manual Start	Protocol and plate information have been loaded to the device form CFX Manager, along with an instruction to not start the protocol until it is started manually from the device	This is associated with manual workflows and should not be encountered in API controlled contexts. Waiting Manual Start->[Any state] indicates that manual intervention has transitioned the device from Waiting Manual Start to the current state.
Processing	A protocol has completed running and the device is processing the resulting data	Running->Processing indicates that a protocol has completed running and the associated data are being processed. Processing->Idle indicates that the data processing performed by the device at the end of a run has completed. Note that this does not imply that CFX Manager has completed post-run data analysis and report generation, which may take up to a minute to complete after the device has processed the data.

CFX System Operational Status States		
StatusType Value	**Description**	**Transitions of Interest**
Preserving	An unrecoverable device error has occurred during a protocol run and the CFX System has lowered the sample temperature to 4°C in an attempt to preserve it	Running->Preserving indicates an unrecoverable device error has occurred during a protocol run, and the sample temperature has been lowered to4°C. The CFX System will not be controllable until its power is recycled.
Error	A device error requiring that the CFX System power be recycled has occurred	[Any state]->Error indicates an unrecoverable device error has occurred. The CFX System will not be controllable until its power is recycled.
Local Run	The CFX System is running a protocol that was started manually prior to connecting to CFX Manager	May be treated analogously to state Running. Operations such as pause, resume, cancel will function as usual, and the status will transition to processing and then to idle when the protocol completes. No data analysis or reporting will be performed by CFX Manager at the end of the run.

1.3.2 CFX MANAGER MANAGEMENT

The Examples Kit provides a collection of useful utility methods for performing various CFX Manager related tasks, such as detecting installed versions, detecting running versions, checking version compatibility, and starting CFX Manager in Server Mode. The source code for these methods is in the file CFXManagerUtilities.cs, located in the Example_Client_Wrapper project, and their use is demonstrated by the Example_Application project.

This section highlights the most important aspects of managing the operation of CFX Manager from API client applications. For additional information, refer to the source code and comments in CFXManagerUtilities.cs, and to the CFX Manager User Manual.

Server Mode vs. User Mode

One of the unique features of the CFX Manager API is that it can be accessed when CFX Manager is running in *User Mode* or *Server Mode*.

User Mode is the normal, fully interactive UI state of the application, which users experience when running CFX Manager as a stand-alone application. When accessing the API while CFX Manager is in User Mode, API operations and user initiated operations may be performed simultaneously; however, certain behaviors will vary between the two points of access. For example, when a user initiated protocol completes, a data analysis screen will automatically appear to the user, while when an API initiated protocol completes, the data analysis screen will not appear. Also, user initiated actions that cause error or warning dialogues to appear will not cause dialogues to appear when they are initiated from the API. This ensures that that a user and the API can share a single copy of CFX Manager without unexpectedly interfering with one another's work, and it allows, for example, users to monitor the real-time progress of API initiated PCR runs.

Server Mode is specifically designed for use with integrated API solutions. When running in server mode, CFX Manager presents no UI, affording API client applications to run as stand-alone solutions.

The method StartCFXManagerAsServer found in CFXManagerUtilities.cs demonstrates how to start CFX Manager in Server Mode.

When running CFX Manager in Server Mode, there are two important considerations that must be kept in mind:

1. If CFX Manager is started in Server Mode, it *must* be shut down via the API using the Shutdown operation, represented in the schema by the ShutDownType element, prior to exiting the client application. If this is not done, CFX Manager will not be available for use again in either Server Mode or User mode until the host computer is re-started or the associated processes are terminated via the Windows Task Manager.

2. If a Server Mode instance of CFX Manager is executing, a User Mode instance of CFX Manager cannot be started, and vice versa.

The Example_Simple_App project provided with the Examples Kit interacts with CFX Manager only in User mode. That is, CFX Manager must be started by the user prior to using the Example_Simple_App application.

The Example_Application project supports both modes as follows: if, when the application is started, a User Mode instance of CFX Manager is already running, it will establish a client connection to that instance; if, on the other hand, CFX Manager is not running when the application starts, it will start a Server Mode instance of CFX Manager and establish a connection with that instance.

Version Compatibility

The API is supported only with CFX Manager Version 3.1.1621 or higher. The source file CFXManagerUtilities.cs includes utilities to obtain and compare the version of the currently installed CFX Manager to this minimum version, as demonstrated by the Example_Application project.

It is important to note that previous versions of CFX Manager may publish previous versions of the API, and that it is possible under certain circumstances to connect your client to an unsupported version of the CFX Manager API. For that reason, it is important that your client application verifies programmatically that the installed version of CFX Manager meets the minimum version requirements.

1.3.3 SUMMARY

The CFX Manager API Examples Kit is a Microsoft Visual Studio 2010 Solution written in C#, and requires .Net Framework 4.0.

It includes source code libraries implementing an API client, a middleware wrapper presenting a simplified API interface to consumer applications, and two fully functional demonstration applications.

The example applications demonstrate how to utilize the API's status polling architecture to simultaneously perform instrument control operations and status monitoring.

The Examples Kit also includes CFX Manager related utilities that demonstrate how to operate CFX Manager in Server Mode, and how to obtain the Version of CFX Manager installed on the host system.

It is important to programmatically verify that a compatible version of CFX Manager is installed since previous versions of CFX Manager may publish incompatible API services that are nonetheless connectible to clients using the current API specifications.

CHAPTER 2 TECHNICAL SPECIFICATIONS

This chapter provides information on the following topics:

- WCF Service Specifications
- Supported Schema Elements

2.1 WCF SERVICE SPECIFICATIONS

The CFX Manager API WCF Service has endpoint address:

http://localhost:8003/BioRad.PCR.CommandManager/SOCFXCommandService

The service description (WSDL) can be obtained while CFX Manager is running from:

http://localhost:8003/BioRad.PCR.CommandManager/SOCFXCommandService?singleWsdl

The following important details related to the published service description must be noted:

- The service includes a call-back contract of type *ISOCFXCommandServiceCallback* corresponding to the call-back event OnServiceIsClosing. This call-back event is reserved for Bio-Rad internal use, and is not supported by the API. However, a stub OnServiceIsClosing instance context must be defined in the client implementation in order to support the service contract. See the Examples Kit source file CFXManagerClient.cs for an example of how to do this.

- The only operation defined by the service contract is *XmlCommand*. This operation accepts and returns serialized Message and Blocks documents as defined in the schema *CfxComSchema.xsd* (refer to Section 1.2), which is located in the SupportFiles folder in the CFX Manager Installation directory.

- The auto-generated ServiceModel configuration will need to be modified to meet certain requirements of the service. In particular, the binding configuration maxReceivedMessageSize property, and the readerQuotas configuration maxStringContentLength and maxBytesPerRead properties will need to be explicitly set to non-default values as specified in the file *app.config*, located in the Example_Application project. That file also contains the recommended binding configuration timeout properties for clients employing the status polling model described in the previous chapter. For clients not employing status polling, the time-out properties explicitly set in that file may need to be modified to satisfy the client's specific operational needs.

2.2 SUPPORTED SCHEMA ELEMENTS

The following tables specify all supported API operations and other supported schema elements. Operations and other elements that exist in the schema but are not listed in these tables are not supported and should not be used by client applications.

2.2.1 TABLE OF SUPPORTED OPERATION ELEMENTS

Supported operations are the schema elements that may be used as the operational element in API Message requests. This table lists and describes all supported operations.

Supported Operations			
Schema Operation Element	**Description**	**Operation Element Parameters**	**Notes**
RegisterServiceType	Register the client with the API Service	None	The API allows only one client to be registered at a time. This operation must be performed upon connecting to the API service in order to obtain a registration ID, which is then required as the MessageType RegistrationID parameter in all subsequent operation requests.
UnRegisterServiceType	Unregister the client with the API Service	None	New client sessions may be not initiated until a registered client unregisters.
QueryInstrumentBlocksType	Request the status of all connected CFX Systems	None	A typical client application will call this at regular intervals to maintain a running real-time snapshot of the status of all connected instruments.
OpenLidType	Open the motorized lid of the targeted CFX System	Base serial number of target CFX System	Note that as soon as this operation is initiated the status of the CFX System will become *Lid Open*. To track the actual position of the lid, monitor the MotorizedLidPosition field of the InstrumentBlockType element.
CloseLidType	Close the motorized lid of the targeted CFX System	Base serial number of target CFX System	Note that as soon as this operation is initiated the status of the CFX System will become *Idle*. To track the actual position of the lid, monitor the MotorizedLidPosition field of the InstrumentBlockType element.

Supported Operations			
Schema Operation Element	**Description**	**Operation Element Parameters**	**Notes**
FlashLedType	Cause the LED indicator on a CFX System lid to blink on and off for approximately 10 seconds	Base serial number of target CFX System	
RunProtocolType	Start a protocol run on the targeted CFX System	Refer to the Table *RunProtocolType Sub-Elements (Parameters)* immediately following this table	Starts a protocol on the targeted CFX System based on a CFX Manager Plate and protocol file pair, on a LIMS file, or on a Bio-Rad PrimePCR file. Allows specification of output data and report file names and locations, and an optional email recipients list.
StopRunType	Stop (Cancel) a running protocol	Base serial number of target CFX System	Note that when the run is stopped, all normal post-run processing will still be performed, including generation, analysis, and emailing (if configured) of the partial data.
PauseRunType	Pause a running protocol	Base serial number of target CFX System	For more information, refer to CFX Manager and CFX System documentation.
ResumeRunType	Resume a paused protocol	Base serial number of target CFX System	For more information, refer to CFX Manager and CFX System documentation.
GenerateReportType	Generate a report from a specified data file and report template.	• Input Data File • Report Template File • Output File Name	Input is a CFX Manager generated data file (.pcrd). If the report template is the empty string, the default CFX Manager report template will be used. If the Output File is the empty string, the data file name will be used. Otherwise, the given fully qualified file name will be used. For more information, refer to CFX Manager and CFX System documentation.
ShutDownType	Shutdown Server Mode instance of CFX Manager	None	Must be used to shut down Server Mode instances of CFX Manager started by client applications. This command gracefully closes multiple processes associated with the CFX Manager instance. Failure to issue this command prior to exiting a client application that has started CFX Manager in server mode will prevent any further use of CFX Manager on the host computer until it is rebooted.

Table of RunProtocolType Sub-Elements

The RunProtocolType operation element specified in the previous table is used to initiate PCR runs. It is the most widely used and also most complex of the operation elements. The following table specifies the meaning and usage of the RunProtocolType sub-elements.

RunProtocolType Sub-Elements (Parameters)		
Sub-element	**Description**	**Notes**
SerialNumber	Base serial number of target CFX System	
ProtocolFile	A CFX Manager Protocol file (.pcrd), Bio-Rad supported LIMS file (.plrn), or Bio-Rad PrimePCR (.csv) file	Fully qualified file name.
PlateFile	A CFX Manager Plate file (.pltd)	Fully qualified file name, or the empty string if the specified ProtocolFile is not a CFX Manager Protocol file (.pcrd).
Note	Text to appear in the output data file and any generated report files as “Notes”	
RunID	Text to appear in the output data file and any generated report files as “ID”	Typically populated with the plate barcode.
DataFile	Name of output data file	Fully qualified file name, including “.pcrd” extension, or the empty string. If empty, the output file settings from CFX Manager will be used.
LockInstrumentPanel	If true, hide the pause/resume and skip buttons on CFX Systems with touch-screen controls.	
GenerateReportEndOfRun	If true, a report will be automatically generated at the end of the run	
GenerateReportType	Unimplemented report type option. Should always be set to ReportTypes.pdf	This is an unimplemented option. The generated report type will always be PDF.
GenerateReportTemplateFile	An optional report template file to be used	If the empty string, the CFX Manager default report template will be used.

RunProtocolType Sub-Elements (Parameters)		
Sub-element	**Description**	**Notes**
GenerateReportOutputFile	Name of report file	Fully qualified file name, including ".pdf" extension, or the empty string. If empty, the specified DataFile name will be used.
EmailAddresses	Comma separated list of email addresses to which generated output and report files will be sent at the completion of the run	Email settings must be configured and tested in CFX Manager for this option to work.

2.2.1 TABLE OF OTHER SUPPORTED SCHEMA ELEMENTS

Other supported schema elements include the request and response elements discussed in Chapter 1, as well as structures and enumerations that are used as sub-element values in the request, response, and operation elements. The following table lists and describes all other supported schema elements.

Other Supported Schema Elements			
Schema Element	**Description**	**Sub-Elements**	**Notes**
ErrorType	Representation of error information	• DateTime • ErrorCode • ErrorDescriptionCurrentCulture • ErrorDescriptionInvariantCulture	Used by the API both to report server side errors and to forward device error reports. In the case of device errors, the error code will be the reported CFX System error code. The invariant culture description will be the English error description. The current culture description will be the error description in the localized language.
FirmwareVersionsType	Report of the various firmware versions associated with a CFX System	• PXA270 • FX2 • HC12 • LID	Reported as part of the InstrumentBlockType status element. Primarily intended for internal Bio-Rad use. However, client applications may wish to track this information for technical support reference.
InstrumentBlocksType	API Response	Refer to Section 1.2	Structure returned in response to API requests, as described in Section 2.1.
InstrumentBlockType	CFX System status record	Refer to Section 1.2	CFX System status record, an array of which is returned as part of API responses, as described in section 2.1.

Other Supported Schema Elements			
Schema Element	**Description**	**Sub-Elements**	**Notes**
MessageType	API Request	Refer to Section 1.2	Structure used to make API requests, as described in Section 2.1
SerialNumbersType	Report of the various serial numbers associated with a CFX System	• Base • Block • Shuttle • OpticalHead	Reported as part of the InstrumentBlockType status element. Primarily intended for internal Bio-Rad use. However, client applications may wish to track this information for technical support reference. Note that the Base serial number, which is used as a parameter for all of the Instrument control operations, is also reported separately as a sub-element of the InstrumentBlockType.
TemperatureUnits	Enumeration used to report temperature units type	(enumerated values) • Celsius	Temperatures are always reported as Celsius.
TemperatureUnitsType	Used to report CFX System lid and block temperatures	• Temperature • Units	CFX System lid and block temperatures are reported as part of the InstrumentBlockType status element using this structure. The Units element is a TemperatureUnits enumeration (see above) whose value will always be Celsius. Therefore the Units element may be ignored for all practical purposes.
VolumeUnits	Enumeration used to report sample volume units type	(enumerated values) • Microliter	Volumes are always reported as microliters.
VolumeUnitsType	Used to report CFX System sample volume	• Volume • Units	Sample Volume is reported as part of the InstrumentBlockType status element using this structure. The Units element is a VolumeUnits enumeration (see above) whose value will always be microliters. Therefore the Units element may be ignored for all practical purposes.

Other Supported Schema Elements			
Schema Element	**Description**	**Sub-Elements**	**Notes**
BlockLetterType	Enumeration used to report the block letter on dual block CFX Systems	(enumerated values) • A • B	The block letter is reported as part of the InstrumentBlockType status element for dual block CFX Systems using this structure.
StatusType	Enumeration corresponding to all possible CFX System operational status states.	Refer to Section 1.2 for a description of all sub-elements.	The operational status is reported as the status sub-element of the InstrumentBlockType element.

APPENDIX A FREQUENTLY ASKED QUESTIONS

This section provides answers to frequently asked questions about working with the CFX Manager API and the API Examples Kit.

Q: Can I access the API from a remote network location?

A: Yes. By changing the endpoint address property in your client's service model configuration from localhost to the IP of the CFX Manager host, you can establish a client connection from anywhere that has network access to the CFX Manager host.

Q: I'm not working in a .Net language. Can I still access the API?

A: Yes. There are WCF interoperability libraries and utilities for virtually all major programming languages. The API service specifically uses a secure interoperable binding for that reason. And since all CFX Manager API operations are specified by an XML schema, clients can be developed using standard web services technologies.

Q: I'm developing an integrated system that will programmatically control multiple CFX Systems, but won't have access to multiple systems for testing during development. Does the API have any simulation features?

A: Yes. The CFX Manager installation includes a shortcut called "CFX Manager (Simulation)," which executes CFX Manager in Simulated mode, allowing you to configure an arbitrary number of simulated CFX Systems that the API can be tested against. To test against configured simulated systems in Server Mode, add the flag "-simulation" to the CFX Manager Startup arguments in method StartCFXManagerAsServer, located in the Examples Kit source file CFXManagerUtilities.cs.

Q: I executed a CFX System operation from my client, and a device error occurred due to a problem that I subsequently identified and corrected. But now when I try to re-execute the operation, I continue to get the same error. Why is this occurring?

A: When a CFX System device error occurs, the CFX System can enter an error state from which it will continue to report the error or related errors until its power is recycled. In general, client applications should respond to device errors in part by notifying users that the power of the affected CFX System needs to be recycled.

Q: When I stop debugging my client application to make changes, then start debugging again, the client will longer connect to the API service. What's happening?

A: When a registered client closes without first unregistering with the API service, no further client connections will be accepted until CFX Manager is restarted. Be sure that your client includes the logic to unregister before closing. Then exit the application rather than halting execution from your IDE when you want to stop debugging.

Q: CFX Manager is running, but my client can't establish a connection with the API Service. How can I troubleshoot this?

A: There are two possible causes for this: issues with the service, and issues with the client. As a first step, perform the sanity check of pointing your web browser to the service URL. If you are hosting the client locally, this will be:

http://localhost:8003/BioRad.PCR.CommandManager/SOCFXCommandService

The following page should appear:

SOCFXCommandService Service

You have created a service.

To test this service, you will need to create a client and use it to call the service. You can do this using the svcutil.exe tool from the command line with the following syntax:

```
svcutil.exe http://localhost:8003/BioRad.PCR.CommandManager/SOCFXCommandService?wsdl
```

You can also access the service description as a single file:

```
http://localhost:8003/BioRad.PCR.CommandManager/SOCFXCommandService?singleWsdl
```

This will generate a configuration file and a code file that contains the client class. Add the two files to your client application and use the generated client class to call the Service. For example:

C#

```
class Test
{
    static void Main()
    {
        SOCFXCommandServiceClient client = new SOCFXCommandServiceClient();

        // Use the 'client' variable to call operations on the service.

        // Always close the client.
        client.Close();
    }
}
```

Visual Basic

```
Class Test
    Shared Sub Main()
        Dim client As SOCFXCommandServiceClient = New SOCFXCommandServiceClient()
        ' Use the 'client' variable to call operations on the service.

        ' Always close the client.
        client.Close()
    End Sub
End Class
```

If this page appears, then the service is accessible. If instead of this page you get a message to the effect that the page can't be displayed, then the service is not accessible.

If the Service is accessible:

- Your client may have previously registered with the API, and then not unregistered prior to closing. Restart CFX Manager. If CFX Manager is running in Server Mode and you cannot shut it down programmatically using the API shutdown operation, use task manager to end the following tasks: BioRadC1000Server.exe, BioRadCFXManger.exe, and BioRadMiniOpticonDiscovery.exe

- If you have restarted CFX Manager but your client still cannot connect to the service, then the issue may be with the client itself. Try debugging the client step by step through the connection and registration process. Review the client code in the Examples Kit for reference

- If you are still unable to resolve the issue, as a final sanity check reboot the host computer, start CFX Manager, and attempt to use one of the demonstration applications from the Examples Kit. If the service appears accessible but the example application is unable to connect the service, contact Bio-Rad Technical support

If the Service is not accessible:

- There may be a process running on the host computer competing for the service port. Some applications, including Skype and certain remote network access applications, may bind to the API service port and prevent the service from starting when CFX Manager is executed. The CFX Manager service must have exclusive access to port 8003 on the host system in order to operate. Try closing all unrelated applications and processes. If the service remains inaccessible, try rebooting the host system. If the problem persists, contact Bio-Rad Technical Support.

CFX Manager™ API Reference Guide

Revision: 1.3
Date: November 21, 2014

Printed in the United States of America

Bio-Rad
Laboratories, Inc.

Life Science
Group

Web site *www.bio-rad.com* ***USA*** *800 424 6723* ***Australia*** *61 2 9914 2800* ***Austria*** *01 877 89 01* ***Belgium*** *09 385 55 11* ***Brazil*** *55 11 3065 7550*
Canada *905 364 3435* ***China*** *86 21 6169 8500* ***Czech Republic*** *420 241 430 532* ***Denmark*** *44 52 10 00* ***Finland*** *09 804 22 00*
France *01 47 95 69 65* ***Germany*** *089 31 884 0* ***Greece*** *30 210 9532 220* ***Hong Kong*** *852 2789 3300* ***Hungary*** *36 1 459 6100* ***India*** *91 124 4029300*
Israel *03 963 6050* ***Italy*** *39 02 216091* ***Japan*** *81 3 6361 7000* ***Korea*** *82 2 3473 4460* ***Mexico*** *52 555 488 7670* ***The Netherlands*** *0318 540666*
New Zealand *64 9 415 2280* ***Norway*** *23 38 41 30* ***Poland*** *48 22 331 99 99* ***Portugal*** *351 21 472 7700* ***Russia*** *7 495 721 14 04*
Singapore *65 6415 3188* ***South Africa*** *27 861 246 723* ***Spain*** *34 91 590 5200* ***Sweden*** *08 555 12700* ***Switzerland*** *026 674 55 05*
Taiwan *886 2 2578 7189* ***Thailand*** *1800 88 22 88* ***United Kingdom*** *020 8328 2000*

Bulletin 6676 Rev A US/EG *14-1955 1114 Sig 1213*