CFX Maestro Software API Reference Guide

For use with all CFX Real-Time PCR Systems

CFX Maestro Software

API Reference Guide

For Use with All CFX Real-Time PCR Systems

BIO-RAD

Bio-Rad Technical Support

The Bio-Rad Technical Support department in the U.S. is open Monday through Friday, 5:00 AM to 5:00 PM, Pacific Time.

Phone: 1-800-424-6723, option 2

Email: Support@bio-rad.com (U.S./Canada Only)

Notice

Table of Contents

Table of Contents

Chapter 1 Introduction

This guide explains the Bio-Rad CFX Maestro Software Application Programming Interface (API). It provides an overview of the API and gives its technical specifications. It also provides an overview of the associatedCFX Maestro Software Source Code Examples Kit (API Examples Kit) and includes answers to frequently asked questions about the API.

Important: The CFX Maestro API is based on the CFX Manager API. All examples in the CFX Manager API kit are applicable to the CFX Maestro API.

This guide is intended for software developers who plan to integrate the capabilities ofCFX Maestro Software 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 Maestro Software (known in this guide as CFX Maestro.

Introduction to the Bio-Rad CFX Maestro Software API

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

Supported API operations include:

- 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 Maestro 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 Maestro. 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 Maestro 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 Maestro 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.

Overview of the API Schema

The CFX Maestro API schema is provided as part of the CFX Maestro Software installation. It is located in the CFX Maestro installation folder (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. Fig. 1 illustrates one complete transaction between a client and the API service:

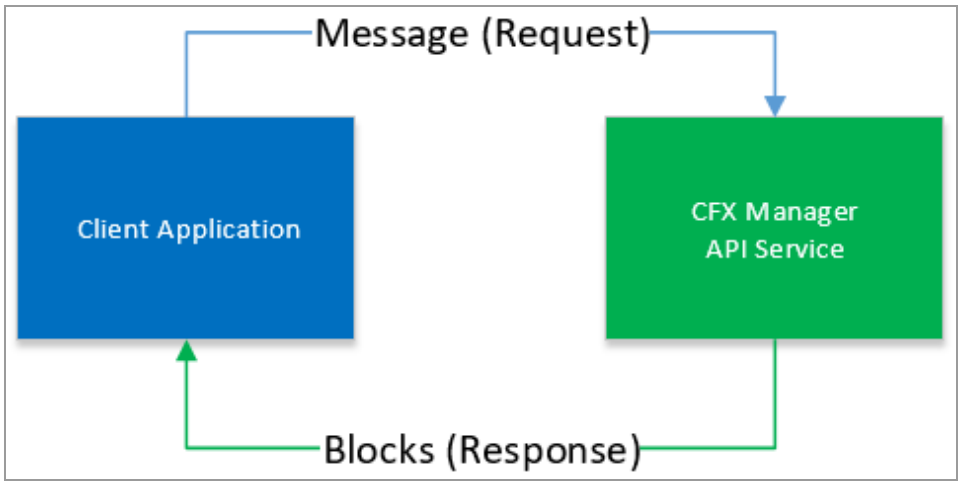


Fig. 1: Complete CFX Maestro client/API service transaction.

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. Therefore, 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.

Requests

API service requests are constructed from the MessageType element, which consists of two subelements:

- **RegistrationID** — A unique registration ID associated with the client connection. Obtained by the Register Service request.
- **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. Fig. 2 illustrates an Open Lid request:

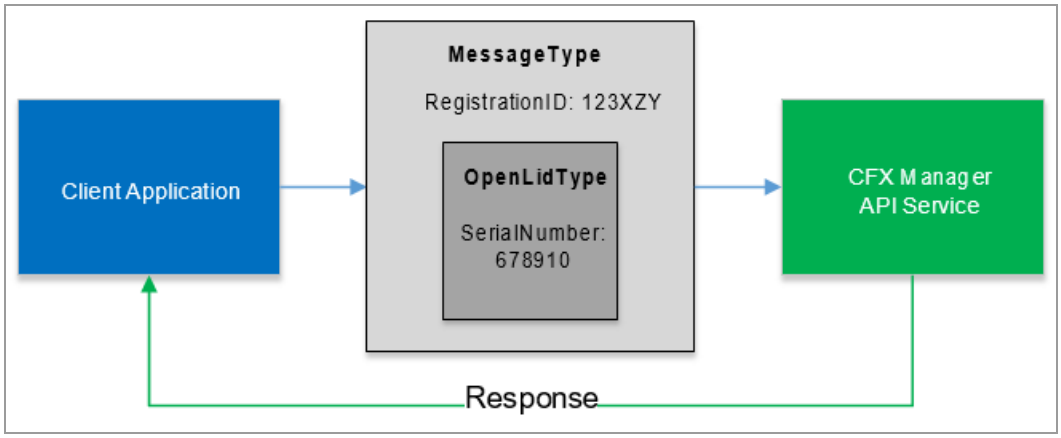


Fig. 2: Open Lid request.

Note: 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.

Responses

API service responses are constructed from the `InstrumentBlocksType` element, which consists of five subelements:

- **`CFXManagerVersion`** — The software version of the CFX Maestro instance hosting the service.
- **`User`** — The current CFX Maestro username, indicating which user preferences are in force.
- **`RegistrationID`** — The unique registration ID associated with the client connection.
- **`ErrorArray`** — An array of `ErrorType` elements (elements defined by the schema to convey error information)

- **`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 system-level failure.

Fig. 3 on page 9 illustrates an `InstrumentBlocksType` response.

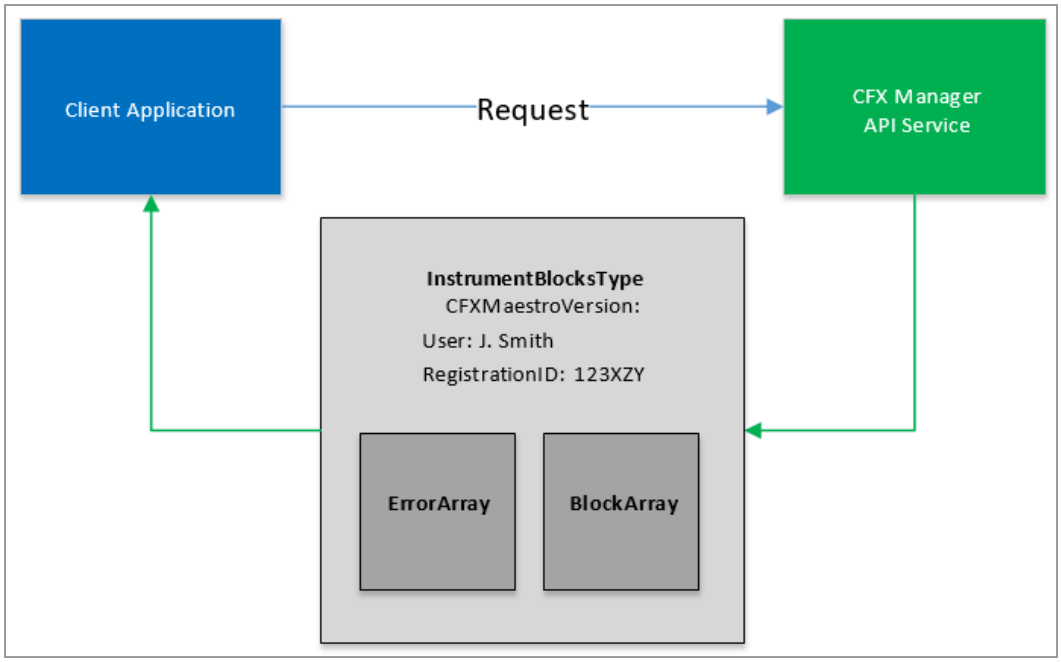


Fig. 3: InstrumentBlocksType response.

The following sections explain 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 subelements:

- **`DateTime`** — A DateTime value used to associate the error with an instance in time.
- **`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.
- **`ErrorDescriptionCurrentCulture`** — A string containing a human-readable description of the error in the language of the currently localized culture.
- **`ErrorDescriptionInvariantCulture`** — A string containing a human-readable description of the error in English.

The API uses the `ErrorType` element 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 subelements 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:

- **`ErrorArray`** — Zero or more `ErrorType` elements corresponding to any currently reported instrument errors.
- **`SerialNumber`** — The serial number of the CFX system base. This is the unique system identifier used as a parameter for instrument control requests.
- **`Status`** — The operational status of the instrument. Possible values are defined in an enumeration found in the schema.
- **`EstimatedRemainingRunTime`** — When the instrument is running a protocol, this will be the estimated time to completion of the protocol.
- **`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.

Fig. 4 on page 11 illustrates a complete `OpenLid` transaction in terms of schema elements, assuming no server errors and no instrument errors occurred during the operation.

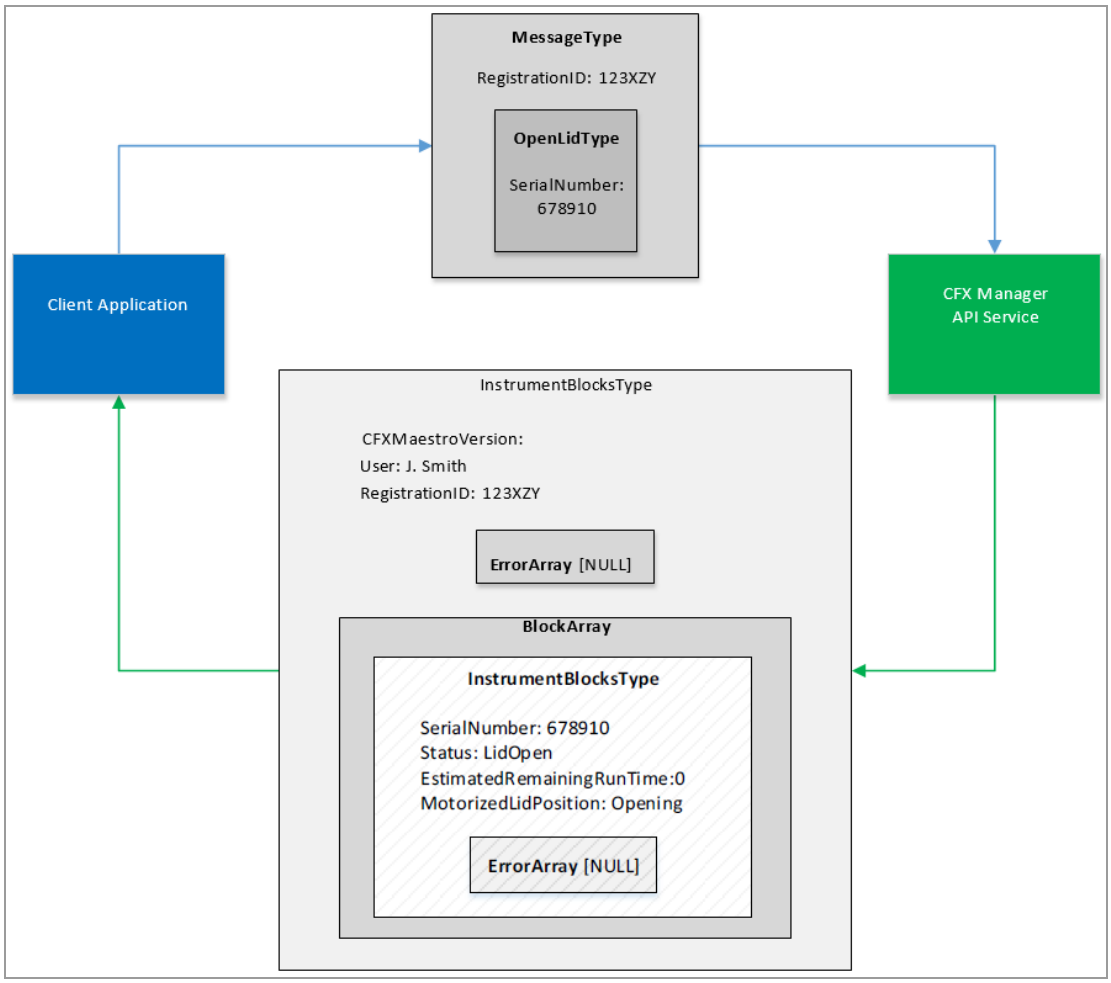


Fig. 4: Complete OpenLid transaction.

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, Technical Specifications 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.

Summary

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

Service requests are defined in the schema by `Message` documents, which contain a unique registration ID and a subelement 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 Maestro 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, Technical Specifications. Any content not specified in Chapter 2 is unsupported and should not be referenced or instantiated by client applications.

Overview of the API Examples Kit

The CFX Manager 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 Maestro service.

Note: All CFX Manager source code examples are compatible with the CFX Maestro software.

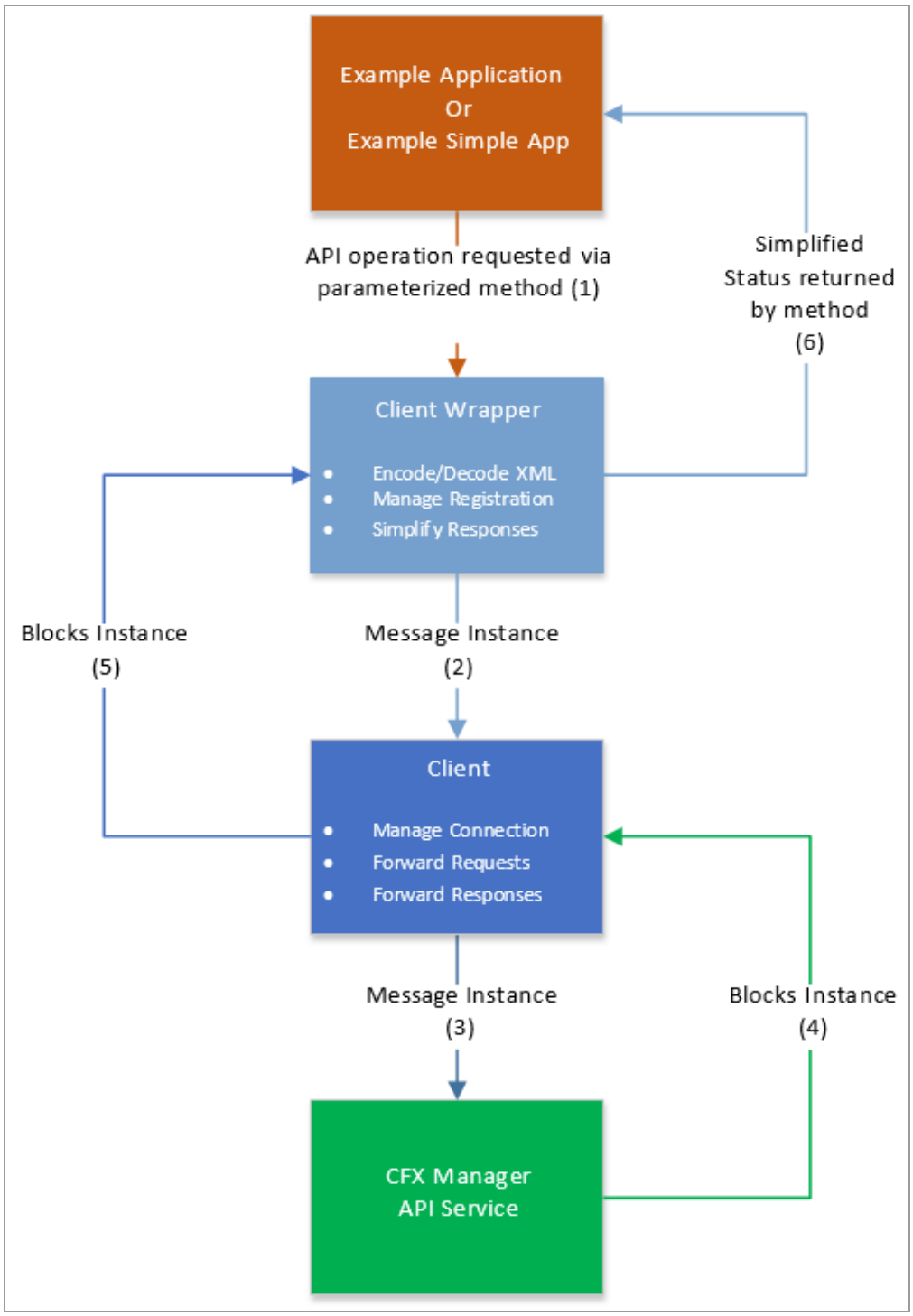
The API Examples Kit is a Visual Studio 2010 solution written in C#. It is compatible with the current version of Visual Studio. Even if you will not be working in C#, the Examples Kit will prove useful by clarifying some of the more subtle aspects of creating and managing CFX Maestro API clients, and 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.

The CFX Manager Examples Kit Solution

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

1. **Example_Client**— Implements a CFX Maestro service client.
2. **Example_Client_Wrapper**— Performs the following:
   - Implements a middleware wrapper around the API service client.
   - Encapsulates and manages client registration and XML encoding and decoding.
   - Provides simplified customer-facing instrument status and error reporting structures.
   - Provides utilities to manage CFX Maestro 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 figure on page 14 illustrates the top-level architecture of the API Examples Kit Solution in terms of the schema structures discussed in the previous sections.


Example Application
Or
Example Simple App
API operation requested via
parameterized method (1)
Simplified
Status returned
by method
(6)
Client Wrapper
Encode/Decode XML
Manage Registration
Simplify Responses
Blocks Instance
(5)
Message Instance
(2)
Client
Manage Connection
Forward Requests
Forward Responses
Message Instance
(3)
Blocks Instance
(4)
CFX Manager
API Service

Note: 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 tables in this section summarize the source files that comprise the Examples Kit solution.

Table 1. 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). See Chapter 2, Technical Specifications for details of the requried 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.

Table 2. 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.

Table 2. Source Files for Project: Example_Client, continued

File	Origin	Description
SOCFXCommandService.cs	Generated by the Microsoft ServiceModel Metadata Utility Tool (Svcutil)	The service model code for the API service. This code must be generated .Net Framework in order to be compatible with the Examples Kit.

Table 3. 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 Maestro is installed on the local host, and to start CFX Maestro in server mode.

Table 4. 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.

Instrument Status Polling

The CFX Maestro API service is designed to support a status polling client application architecture. Instead of defining callback events or using third party frameworks to generate events, the API 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 any 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

Table 5 on page 19 describes the CFX system operational status states and identifies state transitions that client applications might 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.

Table 5. CFX System Operational Status States

Status Type 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 Maestro has completed post-run data analysis and report generation, which can take up to one minute to complete after the device has processed the data.
`Paused`	The instrument has been paused while running a PCRprotocol.	Running -> Paused indicates a running protocol has been paused. Paused -> Running indicates a paused protocol has been resumed.
`Running`	The instrument 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.

Table 5. CFX System Operational Status States, continued

Status Type Value	Description	Transitions of Interest
`Infinite Hold`	A protocol is holding the instrument 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 instrument 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 instrument is being calibrated.	For internal Bio-Rad use. Will not be encountered in API controlled contexts.

Table 5. CFX System Operational Status States, continued

Status Type Value	Description	Transitions of Interest
`Lid Open`	The instrument's 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 Maestro Software and CFX instrment documentation for additional information.
`Initializing`	The instrument 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.

Table 5. CFX System Operational Status States, continued

Status Type Value	Description	Transitions of Interest
`Waiting Manual Start`	Protocol and plate information have been loaded to the instrument from CFX Maestro, along with an instruction to not start the protocol until it is started manually from the instrument .	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 instrument 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 Maestro has completed post-run data analysis and report generation, which can take up to one minute to complete after the device has processed the data.
`Preserving`	An unrecoverable instrument 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 to 4°C. The CFX system will not be controllable until its power is recycled.

Table 5. CFX System Operational Status States, continued

Status Type Value	Description	Transitions of Interest
`Error`	An instrument 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 Maestro.	Can be treated analogously to the state Running. Operations such as pause, resume, and 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 Maestro at the end of the run.

CFX Maestro Software Management

The Examples Kit provides a collection of useful utility methods for performing various CFX Maestro related tasks, such as detecting installed versions, detecting running versions, checking version compatibility, and startingCFX Maestro 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 Maestro from API client applications. For additional information, refer to the source code and comments in `CFXManagerUtilities.cs`, and to the CFX Maestro Software User Guide.

Server Mode vs. User Mode

One of the unique features of the CFX Maestro API is that it can be accessed when CFX Maestro 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 Maestro as a stand-alone application. When accessing the API while CFX Maestro is in User Mode, API operations and user-initiated operations can 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 proto col 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 Maestro without unexpectedly interfering with each other's work, and it allows 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 Maestro 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 Maestro in Server Mode. When running in Server Mode, there are two important items to consider:

1. If CFX Maestro 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 Maestro 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 Maestro is executing, a User Mode instance of CFX Maestro cannot be started, and vice versa.

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

The **Example_Application** project supports both modes as follows:

- If a User Mode instance of CFX Maestro is already running when the application is started, it will establish a client connection to that instance.
- If CFX Maestro is not running when the application starts, it will start a Server Mode instance and establish a connection with that instance.

Version Compatibility

The API is supported only with CFX Maestro Software. The source file `CFXManagerUtilities.cs` includes utilities to obtain and verify the version of the currently installed CFX Maestro, as demonstrated by the **Example_Application** project.

Summary

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

Note: The CFX Maestro API is based on the CFX Manager API. All examples in the CFX Manager API kit are applicable to the CFX Maestro API.

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 Maestro in Server Mode, and how to obtain the version of CFX Maestro installed on the host system.

Chapter 2 Technical Specifications

This chapter explains the WCF Service specifications and the supported schema elements of the CFX Maestro API.

WCF Service Specifications

The CFX Maestro API WCF Service has endpoint address:

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

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

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

Notes: The following are important details related to the published service description:

- 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` (see Overview of the API Schema on page 7) located in the SupportFiles folder in the CFX Maestro 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.

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.

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.

Table 6. 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.
`UnRegisterServiceTyp`	Unregister the client with the API Service	None	New client sessions might not be initiated until a registered client is unregisterd.
`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.

Table 6. Supported Operations

Schema Operation Element	Description	Operation Element Parameters	Notes
`OpenLidType`	Open the motorized lid of the targeted CFX system	Base serial number of the 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 the 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.
FlashLedType	Cause the LED indicator on a CFX system lid to blink on and off for approximately 10 seconds	Base serial number of the target CFX system	

Table 6. Supported Operations

Schema Operation Element	Description	Operation Element Parameters	Notes
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 aCFX Maestro Software 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 the target CFX system	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 the target CFX system	For more information, refer to CFX Maestro Software and CFX system documentation.
ResumeRunType	Resume a paused protocol	Base serial number of the target CFX system	For more information, refer to CFX Maestro Software and CFX system documentation.

Table 6. Supported Operations

Schema Operation Element	Description	Operation Element Parameters	Notes
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 Maestro Software generated data file (.pcrd). If the report template is the empty string, the default CFX Maestro Software 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 theCFX Maestro Software and CFX system documentation.
ShutDownType	Shutdown Server Mode instance of CFX Maestro Software	None	Must be used to shut down Server Mode instances of CFX Maestro Software started by client applications. This command gracefully closes multiple processes associated with the CFX Maestro Software instance. Failure to issue this command prior to exiting a client application that has started CFX Maestro Software in server mode will prevent any further use of CFX Maestro Software on the host computer until it is rebooted.

Table of RunProtocolType Subelements

The `RunProtocolType` operation element specified in the previous table is used to initiate PCR runs. It is the most widely used and also the most complex of the operation elements. Table 7 on page 32 specifies the meaning and usage of the `RunProtocolType` subelements.

Table 7. RunProtocolType Sub-Elements (Parameters)

Subelement	Description	Notes
`SerialNumber`	Base serial number of the target CFX system	
`ProtocolFile`	A CFX Maestro protocol file (.pcrd), Bio-Rad supported LIMS file (.plrn), or Bio-Rad PrimePCR (.csv) file	Fully qualified file name
`PlateFile`	A CFX Maestro plate file (.pltd)	Fully qualified file name, or the empty string if the specified `ProtocolFile` is not a CFX Maestro 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 Maestro will be used.
`LockInstrumentPanel`	If True, hide the pause/resume and skip buttons on CFX systems with touch-screen controls	

Table 7. RunProtocolType Sub-Elements (Parameters), continued

Subelement	Description	Notes
`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 Maestro default report template will be used.
`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 Maestro for this option to work.

Table of Other Supported Schema Elements

Other supported schema elements include the request and response elements discussed in Chapter 1, Introduction, as well as structures and enumerations that are used as sub-element values in the request, response, and operation elements. Table 8 on page 34 lists and describes all other supported schema elements.

Table 8. Other Supported Schema Elements

Schema Element	Description	Subelements	Notes
ErrorType	Representation of error information	■ DateTime ■ ErrorCode ■ ErrorDiscriptionCurrentCulture ■ ErrorDiscriptionInvariantCulture	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 Overview of the API Schema on page 7	Structure returned in response to API requests, as described in WCF Service Specifications on page 27.
InstrumentBlockType	CFX system status record	Refer to Overview of the API Schema	CFX system status record, an array of which is returned as part of API responses, as described in WCF Service Specifications.

Table 8. Other Supported Schema Elements, continued

Schema Element	Description	Subelements	Notes
MessageType	API Request	Refer to Overview of the API Schema	Structure used to make API requests, as described in WCF Service Specifications.
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.

Table 8. Other Supported Schema Elements, continued

Schema Element	Description	Subelements	Notes
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 can be ignored for all practical purposes.

Appendix A Frequently Asked Questions

This appendix provides answers to frequently asked questions about working with the CFX Maestro API and the CFX Manager 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 Maestro host, you can establish a client connection from anywhere that has network access to the 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 Maestro 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 will not have access to multiple systems for testing during development. Does the API have any simulation features?

A: Yes. The CFX Maestro installation includes a shortcut called "CFX Maestro (Simulation)," which executes CFX Maestro Software in simulated mode, allowing you to configure an arbitrary number of simulated CFX Opus systems that the API can be tested against. To test against configured simulated systems in Server Mode, add the flag "-simulation" to the CFX Maestro 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 notify ng 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 Maestro is restarted. Ensure 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 Maestro is running, but my client can not establish a connection with the API Service. How can I troubleshoot this?

A: There are two possible causes for this:

- Issues with the service
- Issues with the client

As a first step, point your web browser to the service URL. If you are hosting the client locally, this is:

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

You see the following page:

If this page appears, then the service is accessible. If instead of this page you get a message to the effect that the page cannot be displayed, then the service is not accessible. If the Service is accessible:

- Your client might have previously registered with the API and then not unregistered before closing. Restart CFX Maestro. If the software is running in Server Mode and cannot be shut down programmatically using the API shutdown operation, use task manager to end the BioRad.
- If you have restarted CFX Maestro but your client still cannot connect to the service, the issue might be with the client itself. Debug 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 reboot the host computer, start CFX Maestro 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 might be a process running on the host computer competing for the service port. Some applications (for example, certain remote network access applications) can bind to the API service port and prevent the service from starting when CFX Maestro is executed. The service must have exclusive access to port 8003 on the host system in order to operate. Close all unrelated applications and processes. If the service remains inaccessible,reboot the host system. If the problem persists, contact Bio-Rad Technical Support.

Bio-Rad Laboratories, Inc.

Life Science Group

Website *bio-rad.com* ***USA*** *1 800 424 6723* ***Australia*** *61 2 9914 2800* ***Austria*** *00 800 00 24 67 23* ***Belgium*** *00 800 00 24 67 23* ***Brazil*** *4003 0399*
Canada *1 905 364 3435* ***China*** *86 21 6169 8500* ***Czech Republic*** *00 800 00 24 67 23* ***Denmark*** *00 800 00 24 67 23* ***Finland*** *00 800 00 24 67 23*
France *00 800 00 24 67 23* ***Germany*** *00 800 00 24 67 23* ***Hong Kong*** *852 2789 3300* ***Hungary*** *00 800 00 24 67 23* ***India*** *91 124 4029300*
Israel *0 3 9636050* ***Italy*** *00 800 00 24 67 23* ***Japan*** *81 3 6361 7000* ***Korea*** *82 2 3473 4460* ***Luxembourg*** *00 800 00 24 67 23*
Mexico *52 555 488 7670* ***The Netherlands*** *00 800 00 24 67 23* ***New Zealand*** *64 9 415 2280* ***Norway*** *00 800 00 24 67 23* ***Poland*** *00 800 00 24 67 23*
Portugal *00 800 00 24 67 23* ***Russian Federation*** *00 800 00 24 67 23* ***Singapore*** *65 6415 3188* ***South Africa*** *00 800 00 24 67 23*
Spain *00 800 00 24 67 23* ***Sweden*** *00 800 00 24 67 23* ***Switzerland*** *00 800 00 24 67 23* ***Taiwan*** *886 2 2578 7189* ***Thailand*** *66 2 651 8311*
United Arab Emirates *36 1 459 6150* ***United Kingdom*** *00 800 00 24 67 23*

10000132623 Ver A US/EG *20-0579 0820 Sig 0220*