31 May 2022

Cell Health Module
Automation Integration Guide

Find us on the World Wide Web at:
www.beckmancoulter.com

Preface

Introduction

The Cell Health Module analyzer is connected to the Shepherd Silas module to allow sample processing without human intervention.

The Cell Health Module running in automation mode provides results consistent with standalone Vi-CELL BLU instruments with the convenience of online sampling for cell concentration and viability.

Automation Instructions for Cell Health Module

This document will go over the process to use the Cell Health Module with external automation software. Cell Health Module uses *Open Platform Communications Unified Architecture* (OPC UA) as the machine to machine communication implementation. If your automation software can function as an OPC UA Client, you can already connect to and communicate with the Cell Health Module. If your automation software will not function as an OPC UA Client, you can use the Cell Health Module .Net DLL to communicate with Cell Health Module using .Net languages like C#, for example.

High Level Software Requirements for Automation

Automation system will be able to exercise the following with the reader:

- Request the health status
- Request operational metrics (i.e. amount of reagent left, tube tray count, available cell types, etc.)
- Request if the reader is idle and ready to be locked for automation
- Request an Automation Mode Lock on the reader to guarantee exclusive operational access
- Release the Automation Mode Lock
- Send sample configuration information to the reader
- Send basic operation commands like Start, Stop, Abort, Pause, etc.
- Request sample result(s) from the reader
- Read notification blocks provided by the reader (i.e. Notification Code, Notification Severity (Warning or Fatal), Notification Description). Dispositioning these is at the discretion of the Automation System

Reader system (Cell Health Module):

- Respond to the requests noted above
- Should look after its own error handling
- Should accept an id for samples from the automation system
- Reader should provide the ability to download to the automation system the raw images used in the reading
- Reader should provide the ability to download unencrypted object data file (eBinary)
  The data will not be able to be imported back into the Reader. Exports can be resource intensive so unencrypted exports will be run when the Reader is not running samples

- When the reader is “Locked” the Human Actor cannot exercise any function that performs Create, Update, or Delete operations against Sample data or Configuration data. Nor will the operator be allowed to exercise resource intensive operations (i.e. exporting). Some examples of operations that would be allowed are running reports, reviewing historical data, etc.

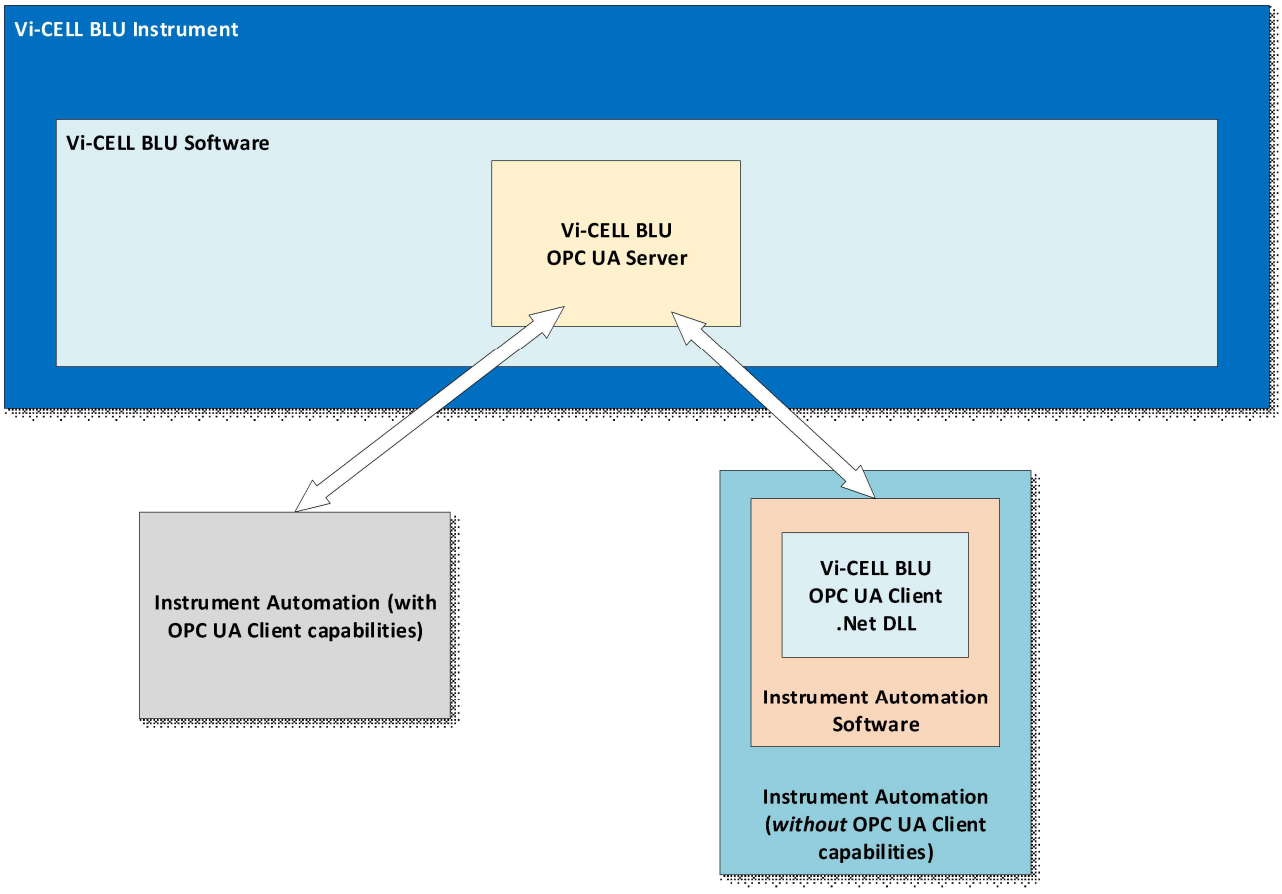
General:

- Interaction following a documented standard communication layer via Ethernet using OPC UA
- The OPC UA Server will be SOA oriented with a defined Service Layer, Operation Layer, and Data Layer
- Beckman Coulter will provide Technical Documentation and Sample Implementation (with source code) to Automation Software Engineers
- Beckman Coulter will provide a .Net client wrapper DLL if the Automation implementer desires to hide the OPC UA implementation from their code base
- When the Reader is locked the user interface splash screen will show that the system is locked for Automation. This visual should be obvious from a distance

System Diagram

There are 2 methods available for your automation software to communicate with the Cell Health Module:

1. Direct OPC UA communication using your OPC UA Client
2. Have your automation software reference the OPC UA Client .Net DLL and code your software to communicate using that library



API Functionality Overview

To aid in this document working for all customers with and without OPC UA Client capabilities, we will assume:

- You already have a working understanding of OPC UA communication (if your automation software already contains OPC UA Client functionality).
- You already have the Cell Health Module .Net DLL referenced in your automation software (if your automation software does not contain OPC UA Client functionality).

Restrictions

OPC UA requires an ethernet port to be open. This port is configurable via the Cell Health Module software, but your company's IT department may need to intervene to allow network communication on this port. You will need to have security enabled and use a username / password to connect to Cell Health Module using OPC UA. There will be a default username/password for access in this case.

NOTE: Keep in mind that this is an early document and that some of the information within is subject to change as product development continues.

Cell Health Module OPC UA Development Platform

Automation Instrument with OPC UA Client:

- OPC UA Clients can function on Microsoft Windows, Apple OSX, Android, or Linux
- OPC UA Clients can be written in a variety of programming languages including: C, C++, Java, .Net, JavaScript and Python

Automation Instrument with Cell Health Module .Net Library:

- The Cell Health Module .Net DLL was written in .Net Framework 4.8 and can be added as a reference to your automation software to directly access it if your software is also programmed in .Net
- The Cell Health Module .Net DLL can function on Microsoft Windows™ 10 v1607 and newer

Programmer Skills Required

If you are using the Cell Health Module .Net DLL, you do not need OPC UA programming skills. You must have to ability to refer to the DLL in your application and use its namespaces. You will also need to be able to call its methods and register .NET delegates for callback events.

If your automation software already has OPC UA Client capabilities, you will need to know how to connect to an OPC UA Server using user authentication and x509 certificates. You will also need to know how to communicate using OPC UA methods and handle its inputs/outputs.

Typical Workflow

1. Ensure the Cell Health Module is powered on, running and connected to the network
2. Ensure your instrument automation is powered on, running and connected to the network
3. Connect your instrument automation to Cell Health Module using user authentication by executing the Connect command
4. Request an automation lock to the Cell Health Module by executing the Automation Lock command with the "automation" username and password.

5. Using external sample cup, execute the start sample command with the details of the sample to be analyzed
6. The Pause, Stop, or Resume commands have no real effect in the Cell Health Module since samples are processed in the A-Cup which only supports one sample.
7. Retrieve currently running sample information during analysis if you wish.
8. Wait for Cell Health Module to send the sample complete event (or poll ViCellStatus and wait for Idle to be returned).
9. Send the Automation Unlock command to release the lock on the Cell Health Module.
10. You can request the sample(s) analysis export by executing the Retrieve Sample Export command.

API Library

Note: The following section is subject to change prior to public release by Beckman Coulter..

The Cell Health module has several variables available for monitoring via OPC UA and several methods that can be called:

Properties

- ViCellIdentifier - string
- SoftwareVersion – string
- FirmwareVersion - string
- CurrentStatus ViCellStatus - enum
- CurrentLockState - LockState - enum
- ReagentUsesRemaining – uint32
- WasteTubeRemainingCapacityRemaining – uint32
- DiskSpaceAvailable uint32 – in MB
- CellTypes –List<CellType>
- QualityControls –List<QualityControl>
- CurrentRunningSampleName - string
- CurrentSamplePosition – SamplePosition
- LastSampleResult – SampleResult

Methods:

- Connect – Establish a connection to a Cell Health Module
    - Inputs
        - Username – string
        - Password – string
        - IpAddr – string
        - Port – uint32
        - discoverTimeout – uint32
        - cnxTimeout – uint32

  - Outputs – none
  - Return VcbResult – enum
- Disconnect
  - Inputs - none
  - Outputs – none
  - Return VcbResult - enum

- RequestLock – Request control of the Cell Health Module
  - Inputs - none
  - Outputs
    - LockState - the current state after request is completed
  - Return VcbResult - enum

- ReleaseLock
  - Inputs - none
  - Outputs - none
  - Return VcbResult - enum

- StartSample - Start for single sample (sample cup)
  - Inputs
    - SampleCfg
  - Outputs - none
  - Returns VcbResult
  - Callbacks called
    - On*SampleComplete* when the sample processing has completed, stopped, or cancelled.
    - On*SampleSetComplete* when the sample processing has completed, stopped, or cancelled.

- StartSampleSet - Start for multiple samples (96 well plate)
  - Inputs
    - SampleSetConfig
  - Outputs - none
  - Returns VcbResult
  - Callbacks called
    - On*SampleComplete* when the sample processing has completed, stopped, or cancelled.
    - On*SampleSetComplete* when the sample processing has completed, stopped, or cancelled.

- Pause
  - Inputs - none
  - Outputs - none
  - Return VcbResult

- Resume
  - Inputs - none
  - Outputs - none
  - Return VcbResult

- Stop
  - Inputs - none
  - Outputs - none
  - Return VcbResult

- EjectStage
  - Inputs - none
  - Outputs - none
  - Return VcbResult

- GetSampleResults
  - Inputs
    - Filter type - eFilterItem
    - From Date - DateTime
    - To Date - DateTime
    - Username - string (string.Empty for all users)
    - Search string for sample name or sample set name (depends on Filter Type) - string
    - Search string for sample tag - string
    - Cell type or quality control name - string
  - Outputs
    - List of SampleResult
  - Return VcbResult

- RetrieveSampleExport
  - Inputs
    - Samples – List<Uuid>
    - Filename – string- filename to save data to
  - Outputs - none
  - Callbacks called
    - *OnSampleExportComplete* – called when the sample export has completed

- DeleteSampleResults
  - Inputs
    - List of GUIDs (SampleRecordDomain uuids)
    - The automated instrument can make calls to GetSampleResults to get the uuids needed for the delete method
  - Outputs - none
  - Return VcbResult
  - Callbacks called
    - On*DeleteSampleResultsComplete* once the instrument has completed the delete operation

- CreateCellType
  - Inputs

        - CellType
    - Outputs none
    - Return VcbResult

- DeleteCellType
    - Inputs
        - CellTypeName - string
    - Outputs - none
    - Return VcbResult

- CreateQualityControl
    - Inputs
        - QualityControl
    - Outputs - none
    - Return VcbResult

- ImportConfig
    - Inputs
        - Filename – string – file to read and send to Cell Health Module
    - Outputs - none
    - Return VcbResult

- ExportConfig
    - Inputs
        - Filename – string – filename to save config to
    - Outputs - none
    - Return VcbResult

- DeleteCampaignData
    - Inputs - none
    - Outputs - none
    - Return VcbResult

- CleanFluidics
    - Inputs - none
    - Outputs - none
    - Return VcbResult
    - Callbacks called
        - *OnCleanFluidicsStatus* – called when the status of the operation changes.

- CancelCleanFluidics
    - Inputs - none
    - Outputs - none
    - Return VcbResult

- GetReagentVolume
    - Inputs

        - Fluid type - CellHealthFluidType
    - Outputs
        - Current reagent volume – int32
    - Return VcbResult

- SetReagentVolume
    - Inputs
        - Fluid type – CellHealthFluidType
        - Volume (uL) to set specified reagent to – int32
    - Outputs - none
    - Return VcbResult

- AddReagentVolume
    - Inputs
        - Fluid type – CellHealthFluidType
        - Volume (uL) to add to specified reagent – int32
    - Outputs - none
    - Return VcbResult

- ShutdownOrReboot
    - Inputs
        - Operation to perform (shutdown or reboot) – ShutdownOrReboot enum
    - Outputs - none
    - Return VcbResult

- PrimeReagents
    - Inputs - none
    - Outputs - none
    - Return VcbResult
    - Callbacks called
        - *OnPrimeReagentsStatus* – called when the status of the operation changes.

- CancelPrimeReagents
    - Inputs - none
    - Outputs - none
    - Return VcbResult

- PurgeReagents
    - Inputs - none
    - Outputs - none
    - Return VcbResult
    - Callbacks called
        - *OnPurgeReagentsStatus* – called when the status of the operation changes.

- CancelPurgeReagents
    - Inputs - none
    - Outputs - none

    - Return VcbResult

- DecontaminateFlowcell
    - Inputs - none
    - Outputs - none
    - Return VcbResult
    - Callbacks called
        - *OnDecontaminateFlowccellStatus* – called when the status of the operation changes.

- CancelDecontaminateFlowcell
    - Inputs - none
    - Outputs - none
    - Return VcbResult

- StartLogDataExport
    - Inputs
        - Output filename path - string
        - Start date/time for export - DateTime
        - End date/time for export - DateTime
    - Outputs
        - CSV containing the Audit, Error and Sample log data is written to the specified directory.
    - Return VcbResultExport

Callback functions

- OnDisconnect
  - Triggered when disconnected from automation target.
  - Parameters
    - None

- OnReconnect
  - Triggered when reconnected to automation target.
  - Parameters
    - None

- OnLockStateUpdate
  - Triggered when the Cell Health Module lock state is updated.
  - Parameters:
    - Lock state: LockState enum

- OnSystemStatusUpdate
  - Triggered when an system status update occurs .
  - Parameters
    - Status – ViCellStatus enum

- OnSampleStatusUpdate
  - Triggered when a system status update occurs .
  - Parameters
    - Status – SampleStatusData

- OnSampleComplete
  - Triggered when a sample has completed analysis.
  - Parameters
    - Result data for the sample that has completed (with UUID) – SampleResult

- OnWorklistComplete
  - Triggered when all samples have completed analysis.
  - Parameters
    - UUIDs of all the samples processed – Array of UUIDs

- OnDeleteStatusUpdate
  - Triggered when there is a change in the sample deletion operation.
  - Parameters:
    - Status – DeleteSampleStatus

- OnExportStatusUpdate
  - Triggered when there is a change in the data export operation.
  - Parameters
    - Status – ExportStatusData

- OnExportComplete

  - Triggered when the export operation has completed.
  - Parameters
    - Status – ExportStatus enum
    - Output filename – string

- OnReagentRemainingUpdate
  - Triggered as samples are processed.
  - Parameters
    - Number of samples that may be processed based on the remaining reagents – uint32

- OnWasteTubeCapacityUpdate
  - Triggered as carousel samples are processed.
  - Parameters
    - Number of samples that may be processed before the Waste Tube Tray must be emptied – uint32

- OnViCellIdentifierUpdate
  - Triggered when the ViCell identification changes.
  - Parameters
    - Identifier – string

- OnCleanFluidicsStatus
  - Triggered when the status of the Clean Fluidics operation changes.
  - Parameters:
    - Status – CleanFluidicsStatus enum

- OnPrimeReagentsStatus
  - Triggered when the Prime Reagents state changes.
  - Parameters
    - Status – PrimeReagentsStatus enum

- OnDecontaminateFlowcellStatus
  - Triggered when the DecontaminateFlowcell state changes.
  - Parameters
    - Status –DecontaminateFlowcellStatus enum

Complex Types/Objects

- SampleConfig
  - Name – string
  - Position – SamplePosition – ignored for A Cup
  - Dilution – uint32
  - Tag – string
  - CellTypeName – string – used if set
  - QCName – string – used if CellTypeName is not set
  - SaveEveryNthImage – uint32
  - WashType – enum – ignored for A Cup
- SampleSetConfig
  - Name - string
  - Samples – List<SampleConfig>

- SampleResult
  - Properties:
    - Configuration - SampleConfig
    - SummaryResultUuid – GUID
    - Date – DateTime
    - CellCount – UInt32
    - ViableCellCount – UInt32
    - ViabilityPercent – double
    - AverageDiameter – double
    - AverageViableDiameter – double
    - AverageCircularity – double
    - AverageCellsPerImage – double
    - AverageBackgroundIntensity – double
    - BubbleCount – UInt32
    - ClusterCount – UInt32
- CellType
  - Properties:
    - Name – string
    - MinDiameter – double
    - MaxDiameter – double
    - NumImages – UInt32
    - Sharpness – double
    - MinCircularity – double
    - DecluserDegree – uint32
    - NumAspirationCycles – UInt32
    - ViableSpotBrightness – double
    - ViableSpotArea – double
    - NumMixingCycles – UInt32
    - ConcentrationAdjustmentFactor - double
- QualityControl
  - Properties:
    - Name – string
    - CellTypeName – string
    - AssayParameter - enum
    - LotNumber - string

        - AssayValue – double
        - AcceptanceLimits - uint
        - ExpirationDate - DateTime
        - Comments - string

- SamplePosition – only used for processing plates
    - Properties:
        - Row – char
        - Column – uint32

Enumerations

- VcbResult
  - Values:
    - Error - 0
    - Success
    - NoConnection
    - NotLocked

- ViCellStatus
  - Values:
    - Unknown - 0
    - Idle
    - Initializing
    - Cleaning
    - Running
    - Error
    - Warning
    - RequiresUserInteraction

- LockState
  - Values:
    - Unknown - 0
    - Locked
    - Unlocked

- AssayParameter
  - Values:
    - Concentration - 0
    - PopulationPercentage
    - Size

- WashType – only used for Sample Sets (plate processing)
  - Values:
    - Normal - 0
    - Fast

- CellHealthFluidType
  - Values:
    - Unknown - 0
    - Cleaner
    - Disinfectant
    - Buffer
    - TrypanBlue
    - Diluent

- ShutdownOrReboot
  - Values:
    - Shutdown - 0

    - Reboot

- CleanFluidicsStatus: the states actually used depends on the workflow script.
    - Values:
        - Idle - 0
        - FlushingCleaner
        - FlushingDisinfectant
        - FlushingBuffer
        - FlushingDiluent (???)
        - FlushingAir
        - Completed
        - Failed

- PrimeReagentsStatus: the states actually used depends on the workflow script.
    - Values:
        - Idle - 0
        - PrimeCleaner
        - PrimeDisinfectant
        - PrimeBufferSolution
        - PrimeTrypanBlue
        - PrimeDiluent
        - Completed
        - Failed

- PurgeReagentsStatus: the states actually used depends on the workflow script.
    - Values:
        - Idle - 0
        - PurgeCleaner
        - PurgeDisinfectant
        - PurgeBufferSolution
        - PurgeTrypanBlue
        - PurgeDiluent
        - Completed
        - Failed

- DecontaminateStatus: the states actually used depends on the workflow script.
    - Values:
        - Idle - 0
        - AspirateBleach
        - Dispensing1
        - DecontaminateDelay
        - Dispensing2
        - FlushingBuffer
        - FlushingAir
        - Completed
        - Failed
        - FindingTube (not used in CellHealth)
        - Complete
        - Cancelled

Certificates and Required Configuration

To connect to the Cell Health Module using the .NET API, you need to configure the OPC UA Security / Certificate validation of the .NET object. An example of how to configure this is provided below. Note that you must provide a callback function to validate the certificate. The callback may actually validate the certificate, or it can simply set the Accept flag to True. The callback function is called during the connection process.

C# Example code snippet:

_myBlu = new ViCellBLU();
SetOpcCertificate(_myBlu);
var result = _myBlu.Connect("username", "password", 192.168.1.1);

Note: username and password must be specified for a valid user.

Configure the .NET Object:
Note: The certificate validation callback function is set in this method.

```
// ****************************************************************
private static void SetOpcCertificate(ViCellBLU blu)
{
   blu.OpcAppConfig.SecurityConfiguration.ApplicationCertificate = new CertificateIdentifier
   {
      StoreType = @"Directory",
      StorePath = @"%CommonApplicationData%\ViCellBlu_dotNET\pki\own",
      SubjectName = "CN=Vi-Cell BLU Client, C=US, S=Colorado, O=Beckman Coulter, DC=" +
                    Dns.GetHostName()
   };
   blu.OpcAppConfig.SecurityConfiguration.TrustedIssuerCertificates = new CertificateTrustList
   {
   StoreType = @"Directory", StorePath = @"%CommonApplicationData%\ViCellBlu_dotNET\pki\issuers"
   };
   blu.OpcAppConfig.SecurityConfiguration.TrustedPeerCertificates = new CertificateTrustList
   {
   StoreType = @"Directory", StorePath = @"%CommonApplicationData%\ViCellBlu_dotNET\pki\trusted"
   };
   blu.OpcAppConfig.SecurityConfiguration.RejectedCertificateStore = new CertificateTrustList
   {
   StoreType = @"Directory", StorePath = @"%CommonApplicationData%\ViCellBlu_dotNET\pki\rejected"
   };
   blu.OpcAppConfig.SecurityConfiguration.UserIssuerCertificates = new CertificateTrustList
   {
   StoreType = @"Directory", StorePath = @"%CommonApplicationData%\ViCellBlu_dotNET\pki\issuerUser"
   };
   blu.OpcAppConfig.SecurityConfiguration.TrustedUserCertificates = new CertificateTrustList
   {
   StoreType = @"Directory", StorePath = @"%CommonApplicationData%\ViCellBlu_dotNET\pki\trustedUser"
   };
   blu.OpcAppConfig.SecurityConfiguration.AddAppCertToTrustedStore = true;
   blu.OpcAppConfig.SecurityConfiguration.RejectSHA1SignedCertificates = false;
   blu.OpcAppConfig.SecurityConfiguration.RejectUnknownRevocationStatus = true;
   blu.OpcAppConfig.SecurityConfiguration.MinimumCertificateKeySize = 2048;
   blu.OpcAppConfig.SecurityConfiguration.SendCertificateChain = true;
   blu.OpcAppConfig.SecurityConfiguration.AutoAcceptUntrustedCertificates = true;
   blu.OpcAppConfig.CertificateValidator.CertificateValidation +=
                                                   CertificateValidator_CertificateValidation;
}
```

Provide a certificate validation function:

```
// ****************************************************************
private static void CertificateValidator_CertificateValidation(CertificateValidator validator,
                                                                CertificateValidationEventArgs e)
{
   if (e.Error.StatusCode != StatusCodes.BadCertificateUntrusted)
   {
      return;
   }
   // Always accepts the certificate
   e.Accept = true;
   return;
}
```