BioTek®

# *LHC Runner User Application*

Interface Specification

**Document Number: 5336208-SP**

**Revision G**

# TABLE OF CONTENTS

# 1. Scope

This document describes the interface between a third-party developer's software application and the BioTek Instrument's LHC_Runner Software.

## 1.1 DOCUMENT HISTORY

| REV | DESCRIPTION OF CHANGES | ECO | DATE |
|---|---|---|---|
| A | Release To Production | 51456 | 10/07/10 |
| B | Added Sections 5.16 and 5.17 and updated return values<br>Added Sections 3.4, 5.18, and 5.19. Updated Sections 4.3 and 4.3 | 52029 | 03/15/12 |
| C | Updated Section 3.1 and 3.2 to add information about the LHC_Caller.<br>Updated Section 5.10. LHC_GetProtocolStatus return values<br>Updated Section 3.1 and 3.2 to add information about ELx405 ActiveX | 52348 | 06/19/12 |
| D | Added section 3.5, 3.6<br>Added section 3.7 for Verify Technology level sensor<br>Added section 5.20 - 5.23 | 527 | 02/21/13 |
| E | Updated section 6.1 Added MultiFloFX to Product Type list.<br>Updated section 6.4 Added detail on interpreting RunStatus values.<br>Updated section 3.4 and 5.8 Clarified that the LHC_OverrideValidation feature is not supported by the ELx405 and the MicroFlo instruments.<br>Updated Section 5.1 LHC_SetProductType [this should no longer be used, it is replaced by the method LHC_SetProductName].<br>Added Section 5.24 LHC_SetProductName [This replaces LHC_SetProductType] | 2217 | 06/18/13 |
| F | Section 3.1 Removed requirement to comment out LHCInterface_ELx405.inf if ELx405 is not used.<br>Section 3.4 Updated new functionality for over-riding validation.<br>Section 3.5.2 Updated Threading Models information.<br>Section 4.3 Added LHC_ValidateProtocol to sequence<br>Section 5.18 Updated description for the LHC_OverrideValidation command.<br>Section 5.25 Added description of LHC_GetWellCount command.<br>Section 5.26 Added description of LHC_ValidateProtocol command.<br>Section 5.27 Added description of LHC_SetRunnerThreading command.<br>Section 5.25 Update *Description* to indicate limited support. | 3609 | 12/23/15 |
| G | Added support for the 50 TS washer:<br>Added section 3.8 Support for the 50 TS washer<br>Added section 5.28 LHC_SetFirstStrip () method description<br>Added section 5.28 LHC_SetNumberOfStrips () method description<br>Section 6.5 Added "50 TS" string to list | 4494 | 2/14/17 |

| | | | |
|---|---|---|---|
| | Section 5.8 LHC_LoadProtocolFromFlash<br>Added a note about loading Maintenance protocols for the 50 TS washer. | | |

# 2. Introduction

BioTek's Liquid Handling Control Runner Software (LHC) contains a new component for third-party developers to more easily execute standard LHC protocol files. The LHC protocols can be loaded from an attached instrument or loaded from files. The *Operating Scenarios* section of this document describes sequence requirements so that proper initialization takes place and that information is provided as needed. For third party integrators, it is important note that the requirements that are needed to run BioTek's "Caller" sample application apply to your application as well. Be sure to read section 3 to learn how to integrate the LHC_Runner into your application and how to manage the special software requirements for the ELx405 washer.

# 3. Installation Issues

An example application has been written by BioTek that calls all of the methods that the LHC_Runner provides. The LHC_Runner is installed as part of the LHC installation. The example application, LHC_Caller.exe, is not part of the LHC install and needs to be obtained separately (can be downloaded) from BioTek. The LHC_Caller comes with source code so a developer can view, debug, and modify the test application. The LHC_Caller.exe is also provided so it can be run directly for testing purposes.

## 3.1. Running the LHC_Caller.exe

Copy the LHC_Caller.exe file to the folder where the LHC was installed. For example, the LHC version 2.12 would be installed by default to *C:\Program Files\BioTek\Liquid Handling Control 2.12*. The LHC_Caller.exe is located in the Run Environment folder of the downloaded example. If the version of the LHC_Caller that you have was built with a reference to a version of the BTILHCRunner.dll that is different than what you have installed, then you will need to recompile the LHC_Caller app with a reference to your version of the BTILHCRunner.dll.

Next, the ELx405 product needs to be considered (this does not apply to the 405 TS/LS product line). If the user would like to run the ELx405 product, then the BTIAutowasher.ocx component needs to be registered. To register the BTIAutowasher.ocx component run the regsvr32 command. For example with Windows XP select *Start* and then *Run*. Type "regsvr32" and then the path and name of the BTIAutowasher.ocx file. An example would be "*regsvr32 C:\Program Files\BioTek\Liquid Handling Control 2.12\ELx405\BTIAutowasher.ocx*". In Windows 7, the *Run* app is in *Start*, *All Programs*, *Accessories* folder.

## 3.2. Working with the LHC_Caller source code

The LHC_Caller source code can be placed anywhere on the developers PC. Copy the entire folder, LHC Caller Test App, to the desired location. Double click on the solution, LHC Caller Test.sln, to view or debug the project in Visual Studio. The LHC_Caller needs a connection back to the LHC installation to run. Follow the instructions below to configure the project.

In the LHC_Runner Test project remove and add the BTILHCRunner reference. When adding the reference, add the BTILHCRunner from the LHC folder (ex: *C:\Program Files\BioTek\Liquid Handling Control 2.12*).

The ELx405 product component is an Active X and needs to be registered. Read section 3.1 above for directions on how to register the ELx405 component (BTIAutowasher.ocx) or disable it if the user is not running the ELx405 product.

The project has been configured to use the Run Environment folder as the working directory. Open the LHC_Installation_Folder.inf in the Run Environment folder and set the path in the file to the folder where the LHC was installed.

The project is provided only as an example of different models of operation using single and multi-threaded architectures. It is not meant to be an example of good programming practices. Many good programming practices have been ignored for the purpose of keeping this sample application small and easier to read. You should apply your own company's programming standards. You should always evaluate values that are returned by method calls provided in the LHCBTIRunner.dll.

## 3.3. LHC Software Registration

Starting with version 2.1, the LHC software contains an inclusive install for the LHC_Runner software and Interface Software for all supported instruments. When installing the software, a serial number dialog will appear and you must enter the serial number that comes with the LHC package. This will give you a number of “demo days” where the software will run with full functionality. Each time the software is run, a registration dialog will be displayed. This will continue to be displayed until the software is registered. At any point during the demo period, you can register the software with BioTek and the registration dialog will stop appearing. If the demo period expires and you have not yet registered the software, the software will limit its functionality until it is registered.

## 3.4. Optimizing Run Time

Starting with version 2.12 of the LHC_Runner software, two methods have been added to help speed up processing and maximize throughput.

Each time the **LHC_RunProtocol**() method is called, the LHC_Runner performs a pre-run validation sequence. It will test the selected communication port, determine the current configuration of the specified instrument, and validate the selected protocol to ensure it will run on the instrument. This can take a few second depending on the instrument and the protocol. If the protocol will be run multiple times once it is loaded and validation is not needed each time it is run, then time can be saved by skipping this pre-run validation. There is a method called **LHC_OverrideValidation**() that will omit validation when the **LHC_RunProtocol**() method is called. However, validation *must be performed once* before a protocol can be run. There is a new method called **LHC_ValidateProtocol**() that will perform validation of the loaded protocol. See the description for **LHC_OverrideValidation**() in section 5.18 and the description for **LHC_ValidateProtocol**() in section 5.26.

The integrator should follow one of these 3 basic sequences to validate and run protocols:

1. Perform validation before each time the protocol is run. This is accomplished by loading your protocol and then using only the **LHC_RunProtocol**() method and not using the specialized **LHC_OverrideValidation** and **LHC_ValidateProtocol**() methods.
2. Load your protocol, run it once (**LHC_RunProtocol**() will perform the initial validation), and then turn validation off using **LHC_OverrideValidation(1)** for all subsequent runs of this protocol.
3. Load your protocol, validate it using the **LHC_ValidateProtocol(true)** method, turn validation off using **LHC_OverrideValidation(1**) method, and then run the protocol as many times as needed.

*Important*: Each time you load a protocol, all the validation information is removed and you must validate that protocol in one of the 3 sequences described above before running it.
(*Overriding validation is not supported by the ELx405 or the MicroFlo instruments).*

The second new method helps to eliminate wait times that are sometimes experienced by the vacuum pump. When a protocol run is complete, the vacuum pump is turned off and in some cases, depending on the type of vacuum pump and vacuum bottle size, it cannot be started again until a vacuum dissipate time duration is complete. If you are running protocols back-to-back, this unwanted delay can be bypassed by leaving the vacuum pump on when the protocol is done. There is now a method called **LHC_LeaveVacuumPumpOn**() that gives you control over when the vacuum pump turns off. See the description for this method in section 5.19

## 3.5. Integrator .NET Applications

This section offers some specifics to using the LHC_Runner software if you are calling it from a .NET application created with Visual Studio.

### 3.5.1.Referencing the LHC_Runner.dll

Your Visual Studio project needs to have a reference to the LHC_Runner.dll. But first, you must determine where you want the LHC_Runner.dll to reside. You can reference it from the folder where the

LHC was installed, or you can move it into your project folder. At run time, the LHC_Runner.dll looks in the folder from where it is running for a file named *LHC_Installation_Folder.inf*. This is an XML file that specifies where the LHC was installed so the LHC_Runner.dll can find all the supporting files needed to interface with the instruments. If you move the LHC_Runner.dll to a folder other than where it was installed with the LHC software, then you will also need to copy the *LHC_Installation_Folder.inf* file into the folder with the LHC_Runner.dll and be sure its contents "points to" the correct folder for the version of LHC you are using.

### 3.5.2.Threading models

Integrators implement many different approaches to calling the LHC_Runner.dll. Some use a single threaded model with a timer tick for polling, and others use multi-threaded model with their UI running independently from the LHC_Runner.dll. The LHC_Caller sample application provided by BioTek offers a few different models for using the LHC_Runner.dll. Please see section 3.2 for other information about the LHC_Caller sample application and it how to use it. Ultimately, you need to find the best approach for integrating the LHC_Runner functionality into your application.

A new method has been added so that the LHC_Runner can run in threads it creates. Section 5.27 describes how LHC_Runner commands that require interfacing with the instrument can be run in their own thread so they become independent of the caller application.

### 3.5.3.Controlling Selected Instruments

In the LHC installation folder, you will find several files that start with *LHCInterface_* and have an *.INF* file extension. These files are how the LHC and the LHC_Runner find what instrument drivers are installed. If you are not supporting a particular instrument, you can rename its file to have an extension of *.INF_OMIT.* For instance, rename *LHCInterface_ELx405.INF* to *LHCInterface_ELx405.INF_OMIT* if your application does not support the ELx405. The ELx405 is a special case because its driver is an ActiveX component. See section 3.1 for special instructions on registering this ActiveX if you support the ELx405.

### 3.5.4.Evaluate Return Values

A very important aspect of interfacing with this LHC_Runner is the values that are returned by the method calls. Be sure you evaluate the return values of every method to determine if an error has occurred. Since errors can be detected by the LHC_Runner, the instrument drivers, or the instrument's on-board basecode, methods can return different types of errors. Refer to the specific type of return values for each method. Ignoring errors and making calls out of sequence can cause unpredictable behavior.

### 3.5.5.Upgrading to a newer version of the LHC

If you upgrade to a newer version of the LHC, it is best to recompile your application with the LHC_Runner version that gets installed with the new LHC. It is very important that the *LHC_Installation_Folder.inf* file contains the name of the correct LHC Installation folder you wish to use. Using a new LHC_Runner but a previous version of the LHC could cause unpredictable behavior.

## 3.6. COM Visible Example

For developers who are not using .Net there is a folder called "COM Visible Example". This folder contains a .Net C# project that creates a Component Object Module (COM) visible component that calls the LHC Runner. This project is in a folder called "LHC Runner Wrapper". The folder also contains an MFC C++ example Caller that works with the COM visible component to access the LHC Runner. This project is in a folder called "LHCCallerMfc using wrapper".

The "LHC Runner Wrapper" folder contains the BTILHCRunnerWrapper project that builds and registers the BTILHCRunnerWrapper.dll. The BTILHCRunnerWrapper.dll is the entry point for calling the LHC Runner functions for non .Net applications. This project was developed using Visual Studio 2008. Before building the project the BTILHCRunner reference in the project needs to be removed and then added into the project again. Remove the reference and then add it again by locating the BTILHCRunner.dll in the folder where the LHC was installed on the PC (c:\Program Files\BioTek\Liquid Handling Control x.xx).

Next display the Project Properties screen and select the Build tab. Change the Output Path to the same directory that contains the BTILHCRunner.dll. Currently the project Output Path is the LHCCallerMfc example folder. This path should remain as it is if the developer is going to run the LHCCallerMfc example in debug mode. Once the BTILHCRunnerWrapper is built on the PC it will be registered on the PC and can be accessed like any other COM object.

NOTICE: The BTILHCRunnerWrapper project was created for the 2.12.4 version of the LHC Runner. If the developer is running with an older or newer version of the LHC Runner then the interface description and the functions that wrap the LHC Runner may need to change. The interface and the wrapper functions are in ClassLHCRunnerWrapper.cs.

The “LHCCallerMfc using wrapper” folder contains a sample MFC C++ caller application that uses the BTILHCRunnerWrapper.dll to access the LHC Runner functions. This sample caller can test communications, run a self test, and run and monitor the status of a protocol. This project was developed using Visual Studio 2008. Before building this project copy the BTILHCRunnerWrapper.tlb file from the output area of the BTILHCRunnerWrapper project into the “LHCCallerMfc using wrapper” folder. The project can then be built and run. The two projects are currently set up so the LHCCallerMfcWrapper.exe can be run out of the debug folder of the LHCCallerMfc project. The debug folder would contain the BTILHCRunner.dll and BTILHCRunnerWrapper.dll. The LHC_Installation_Folder.inf file in the debug folder needs to point to the directory where the LHC was installed on the PC.

## 3.7. Verify Manifold test (405TS/LS option)

### 3.7.1. Preparing for a Verify Manifold test

If your 405 TS/LS is equipped with the Ultrasonic level sensor (using BioTek’s *Verify Technology*), the Verify Manifold test can evaluate the washer manifold for clogged or impaired aspirate and dispense tubes. Four new methods were added to support this feature. See section 4.6 of the Operating Scenarios for the sequencing of these methods. See sections 5.20 – 5.23 for the functional description of each method. See the LHC Help system for a description of the requirements, details, and outcomes of the test.

### 3.7.2. Using the LHC to create the Settings files

The Verify Manifold test uses a *ManifoldTestSettings.inf* file to guide how parts of the test are performed. This file is created the first time the Verify Manifold test is run by the LHC software. A copy of this file is saved for each instrument that has the Verify Manifold feature. This settings file stores the user’s choice for several parameters:

- The buffer to use if the instrument has valves for multi-buffer connections
- Whether to perform a manifold prime before the start of the test
- The tolerance value to use a cut-off for the dispense %CV
- The Windows folder where test Reports are stored
- LHC display parameters

To set these values, run the Verify Manifold test from the LHC software where you will have access to each parameter. The saved parameters will be used when the LHC_Runner executes the Verify Manifold test. You must generate a *ManifoldTestSettings.inf* for each instrument even if the parameters are the same.

## 3.8. Support for the 50 TS washer

BioTek is adding LHC Runner support for its 50 TS washer. When run from its touch screen interface, this washer allows the user to specify the range of strips (columns of wells) in which the protocol will be run. This feature is being added to the LHC_Runner as well. This is accomplished by specifying the first strip and number of strips to process. Using an 8 channel manifold, the entire 96 well plate would be processed if you were to specify *first strip* = 1 and *number of strips* = 12. If you were to specify *first strip* = 6 and *number of strips* = 3, then the protocol would be run in strips 6, 7, and 8.

When a 50 TS protocol is loaded (using either the *LHC_LoadProtocolFromFile* method or the *LHC_LoadProtocolFromFlash* method ), by default, all strips are selected for that plate type. If you wish to change that selection, use the *LHC_SetFirstStrip* method as described in section 5.28 and the *LHC_SetNumberOfStrips* method as described in section 5.29. These methods should always be used together. Be sure that the number of strips to process starting at the specified first strip to process does not exceed the number of strip (column) in the plate.

If you are using the *LHC_SetProductName* method, for this 50 TS washer you must use “50 TS” with a space character between the 50 and the TS characters.

When using the LHC software to create protocols for the 50 TS washer, there are the typical washer commands to *Aspirate*, *Dispense*, *Wash*, etc. There is also a new command called *Prompt.* When encounter at runtime, the processing will stop and a prompt dialog will be displayed with the specified prompt message. No further processing takes place until the user responds. In an unattended automated system, this behavior may be undesirable. Therefore, be sure your protocols do not contain any unwanted *Prompt* commands.

# 4. Operating Scenarios

This section describes some scenarios when interfacing with the LHC software and the required sequence of method calls expected by the LHC Runner software. The developer's application must follow these sequences. Not all scenarios may be implemented.

## 4.1. Product Initialization

```
LHC_SetProductName();           // set product name: El406, ELx405, MultiFloFX, etc…
LHC_SetCommunications();        // set COM port to use or indicate direct USB
```

## 4.2. Communications Test

Perform *Product Initialization* sequence

```
LHC_TestCommunications()        // test that device is attached and responding
```

## 4.3. Running Protocols from .LHC File

```
LHC_LoadProtocolFromFile();     // file and pathname of LHC protocol

LHC_SetCommunications();        // set COM port to use or indicate direct USB

LHC_TestCommunications();       // confirm instrument attached is responding

LHC_SetFirstStrip ();           // optional call to set first strip to process (50 TS washer only)

LHC_SetNumberOfStrips ();       // optional call to set number of strips to process (50 TS washer only)

LHC_ValidateProtocol();         // optional call to force immediate validation

LHC_OverrideValidation();       // optional call to bypass validation

LHC_LeaveVacuumPumpOn();        // optional call to leave pump on when run is complete

LHC_RunProtocol();              // validates protocol to instrument and starts execution
While LHC Protocol not complete {
   LHC_GetProtocolStatus()      // return completion status of executing LHC protocol file

   Optional
      LHC_PauseProtocol()
      If paused
           LHC_ResumeProtocol() or
           LHC_AbortProtocol()
}
If error generated
   LHC_GetErrorString()
```

## 4.4. Running Protocols from Flash (on board instrument)

```
Perform Communication Test sequence
LHC_GetOnBoardProtocolCount()  // optional, but requires LHC_GetOnBoardProtocolName
LHC_GetOnBoardProtocolName()   // optional, but requires LHC_GetOnBoardProtocolCount

LHC_LoadProtocolFromFlash();   // name of protocol stored in instrument's flash memory

LHC_SetFirstStrip ();          // optional call to set first strip to process (50 TS washer only)

LHC_SetNumberOfStrips ();      // optional call to set number of strips to process (50 TS washer only)

LHC_OverrideValidation();      // optional call to bypass validation

LHC_LeaveVacuumPumpOn();       // optional call to leave pump on when run is complete

LHC_RunProtocol();             // validates protocol to instrument and starts execution
While LHC Protocol not complete {
   LHC_GetProtocolStatus()     // return completion status of executing LHC protocol file

   Optional
      LHC_PauseProtocol()
      If paused
           LHC_ResumeProtocol() or
           LHC_AbortProtocol()
}
If error generated
   LHC_GetErrorString()
```

## 4.5. Miscellaneous Methods

```
Perform Product Initialization sequence
LHC_GetProductName()
LHC_GetProductSerialNumber()
LHC_PerformSelfCheck()
LHC_GetLastErrorCode()
LHC_OverrideValidation();
LHC_LeaveVacuumPumpOn();
```

## 4.6. Perform Verify Manifold test

```
LHC_SetProductName();          // set product name: El406, ELx405, MultiFloFX, etc…
LHC_SetCommunications();       // set COM port to use or indicate direct USB
LHC_TestCommunications();      // confirm instrument attached is responding

LHC_RunVerifyManifoldTest ();  // Start the Verify Manifold test
While Manifold Test not complete {
   LHC_GetVerifyManifoldTestStatus()      // return status of executing test
}
When complete
   LHC_GetVerifyManifoldTestResults ()    // causes data to be save and return test result
   Optional
      LHC_CreateVerifyManifoldTestReport ()       // creates a text-formatted report for the test
```

# 5. LHC Runner Interface Methods

These methods detail the LHC_Runner *method* calls used in the operating scenarios above.

## 5.1. LHC_SetProductType < < < *NO LONGER USED* > > >

### 5.1.1. Format

Int16 LHC_SetProductType (Int16 nProductType)

### 5.1.2. Description

Sets the LHC instrument type based on an enumeration passed in.

### 5.1.3. Passed Parameters

| Type | Name | Values |
|---|---|---|
| Int16 | nProductType | ProductType |

### 5.1.4. Return Value

| Type | Name | Values |
|---|---|---|
| Int16 | nReturnCode | Runner_ReturnCode |

### 5.1.5. Comments

*[NOTE: This should no longer be used. You should use the "LHC_SetProductName" method]*

If the nReturnCode is not eOK, then make a call to LHC_GetErrorString (nReturnCode) to get a string that describes the error.
See *Operating Scenarios* section 4.1 for related method calls.

## 5.2. LHC_SetCommunications

### 5.2.1. Format

Int16 LHC_SetCommunications (String strSetting )

### 5.2.2. Description

Sets the instrument's COM port assignment or USB identifier based on a string passed in. The available COM and USB values can be viewed by clicking the Port link in the LHC software and dropping down the Port list on the Communications dialog.

### 5.2.3. Passed Parameters

| Type | Name | Values |
|---|---|---|
| String | strSetting | String of format "COMn" where n is a port number or<br>"USB inst sn: x" where inst is the instrument type and x is the serial number. |

### 5.2.4. Return Value

| Type | Name | Values |
|---|---|---|
| Int16 | nReturnCode | Runner_ReturnCode |

### 5.2.5. Comments

If the nReturnCode is not eOK, then make a call to LHC_GetErrorString (nReturnCode) to get a string that describes the error.
See *Operating Scenarios* section 4.1 for related method calls.

## 5.3. LHC_TestCommunications

### 5.3.1. Format

Int16 LHC_TestCommunications ( )

### 5.3.2. Description

Tests the communication between the PC and the instrument.

### 5.3.3. Passed Parameters

None.

### 5.3.4. Return Value

| Type | Name | Values |
|---|---|---|
| Int16 | nReturnCode | Runner_ReturnCode or IS_ReturnCode |

### 5.3.5. Comments

If the nReturnCode is not eOK, then make a call to LHC_GetErrorString (nReturnCode) to get a string that describes the error.
See *Operating Scenarios* sections 4.2 related method calls.

## 5.4. LHC_GetProductName

### 5.4.1. Format

Int16 LHC_GetProductName (ref string strProductName)

### 5.4.2. Description

Returns a string containing the name of the Instrument. (This is from the Interface Software's Assembly Name)

### 5.4.3. Passed Parameters

| Type | Name | Values |
|---|---|---|
| String | strProductName | The string to contain the Product Name |

### 5.4.4. Return Value

| Type | Name | Values |
|---|---|---|
| Int16 | nReturnCode | Runner_ReturnCode |

### 5.4.5. Comments

See *Operating Scenarios* section 4.5 for related method calls.

## 5.5. LHC_GetProductSerialNumber

### 5.5.1. Format

Int16 LHC_GetProductSerialNumber (ref string strProductSN)

### 5.5.2. Description

Returns a string containing the attached instrument's serial number. This function is not supported for the ELx405 instrument.

### 5.5.3. Passed Parameters

| Type | Name | Values |
|---|---|---|
| String | strProductSN | The string to contain the serial number |

### 5.5.4. Return Value

| Type | Name | Values |
|---|---|---|
| Int16 | nReturnCode | Runner_ReturnCode |

### 5.5.5. Comments

See *Operating Scenarios* section 4.5 for related method calls.
This function is not supported for the Elx405 instrument.

## 5.6. LHC_PerformSelfCheck

### 5.6.1. Format

Int16 LHC_PerformSelfCheck ( )

### 5.6.2. Description

Instrument attached will perform a system test and return a status. This function is not supported for the Elx405 instrument.

### 5.6.3. Passed Parameters

None.

### 5.6.4. Return Value

| Type | Name | Values |
|---|---|---|
| Int16 | nReturnCode | Runner_ReturnCode or IS_ReturnCode |

### 5.6.5. Comments

This function is not supported for the Elx405 instrument.
If the nReturnCode is not eOK, then make a call to LHC_GetErrorString (nReturnCode) to get a string that describes the error.
See *Operating Scenarios* sections 4.5 for related method calls.

## 5.7. LHC_LoadProtocolFromFile

### 5.7.1. Format

Int16 LHC_LoadProtocolFromFile (string strPathname)

### 5.7.2. Description

Loads an LHC protocol file.

### 5.7.3. Passed Parameters

| Type | Name | Values |
|---|---|---|
| String | strPathname | The string that defines path and file name of the LHC protocol to load. |

### 5.7.4. Return Value

| Type | Name | Values |
|---|---|---|
| Int16 | nReturnCode | Runner_ReturnCode |

### 5.7.5. Comments

See *Operating Scenarios* section 4.3 for related method calls.

## 5.8. LHC_LoadProtocolFromFlash

### 5.8.1. Format

Int16 LHC_LoadProtocolFromFlash (string strProtocol)

### 5.8.2. Description

Loads the specified protocol file from the instrument's flash memory. (see note below)

### 5.8.3. Passed Parameters

| Type | Name | Values |
|---|---|---|
| String | strProtocol | The name of the on-board protocol stored in flash memory. (see note below) |

### 5.8.4. Return Value

| Type | Name | Values |
|---|---|---|
| Int16 | nReturnCode | Runner_ReturnCode |

### 5.8.5. Comments

See *Operating Scenarios* section 4.4 for related method calls.

**NOTE**: With a 50 TS washer, if you wish to load and run one of the instruments on-board Maintenance protocols, you must add "@@" to the front of the filename. For example, if you want to load the "Daily_Clean" maintenance protocol, you must request "@@Daily_Clean" as the filename. The "@@" characters is how the washer differentiates between normal protocols and maintenance protocols when they are transferred to the washer from a PC running the LHC software.

## 5.9. LHC_RunProtocol

### 5.9.1. Format

Int16 LHC_RunProtocol ()

### 5.9.2. Description

Validate and begin executing the currently loaded LHC protocol.

### 5.9.3. Passed Parameters

None.

### 5.9.4. Return Value

| Type | Name | Values |
|---|---|---|
| Int16 | nRunStatus | RunStatus |

### 5.9.5. Comments

This command validates and starts the execution of the LHC protocol. To determine when the LHC protocol is done, the LHC_GetProtocolStatus command must be issued repeatedly (polled). See the LHC_GetProtocolStatus command description for information about threads and multi-thread models. See Section 6.4 on interpreting the RunStatus.
If an error occurs during this call,
See *Operating Scenarios* section 4.3 and 4.4 for related method calls.

## 5.10. LHC_GetProtocolStatus

### 5.10.1. Format

Int16 LHC_GetProtocolStatus ()

### 5.10.2. Description

Get the run status of the currently executing LHC protocol.

### 5.10.3. Passed Parameters

None.

### 5.10.4. Return Value

| Type | Name | Values |
|---|---|---|
| Int16 | nRunStatus | RunStatus or IS_ReturnCode. |

### 5.10.5. Comments

The LHC_GetProtocolStatus command must be issued repeatedly (polled) until the run completes. In a single thread model, it is best to perform this by using a Timer Tick event handler and not a Thread.Sleep command. Thread.Sleep commands should only be used if the protocol is run in its own thread and a proper multi-thread model is used. See the latest LHC_Caller sample application for example of multi-threaded models.
If the RunStatus indicates an error, call LHC_GetLastErrorCode () to get the specific error code that occurred and then pass the error to LHC_GetErrorString (nReturnCode) to get a string that describes the error. See Section 6.4 on interpreting the RunStatus.
See *Operating Scenarios* section 4.3 and 4.4 for related method calls.

## 5.11. LHC_PauseProtocol

### 5.11.1. Format

Int16 LHC_PauseProtocol ( )

### 5.11.2. Description

Pauses the currently executing LHC protocol.

### 5.11.3. Passed Parameters

None.

### 5.11.4. Return Value

| Type | Name | Values |
|---|---|---|
| Int16 | nReturnCode | Runner_ReturnCode |

### 5.11.5. Comments

See *Operating Scenarios* section 4.3 and 4.4 for related method calls. Once the instrument pauses, the run can be told to Resume (LHC_ResumeProtocol) or Abort (LHC_AbortProtocol).

## 5.12. LHC_ResumeProtocol

### 5.12.1. Format

Int16 LHC_ResumeProtocol ( )

### 5.12.2. Description

Resumes processing the currently paused LHC protocol.

### 5.12.3. Passed Parameters

None.

### 5.12.4. Return Value

| Type | Name | Values |
|---|---|---|
| Int16 | nReturnCode | Runner_ReturnCode |

### 5.12.5. Comments

The executing protocol must to paused (LHC_PauseProtocol) before this method can be called. See *Operating Scenarios* section 4.3 and 4.4 for related method calls.

## 5.13. LHC_AbortProtocol

### 5.13.1. Format

Int16 LHC_AbortProtocol ()

### 5.13.2. Description

Aborts the currently executing LHC protocol that is paused.

### 5.13.3. Passed Parameters

None.

### 5.13.4. Return Value

| Type | Name | Values |
|---|---|---|
| Int16 | nReturnCode | Runner_ReturnCode |

### 5.13.5. Comments

The executing protocol must to paused (LHC_PauseProtocol) before this method can be called. See *Operating Scenarios* section 4.3 and 4.4 for related method calls.

## 5.14. LHC_GetErrorString

### 5.14.1. Format

string LHC_GetErrorString(Int16 nLastReturnCode)

### 5.14.2. Description

Returns the string describing the specified error code.

### 5.14.3. Passed Parameters

| Type | Name | Values |
|---|---|---|
| Int16 | nLastReturnCode | Runner_ReturnCode or IS_ReturnCode. |

### 5.14.4. Return Value

| Type | Name | Values |
|---|---|---|
| String | strDefinition | An error text string. |

### 5.14.5. Comments

See *Operating Scenarios* section 4.3 and 4.4 for related method calls.

## 5.15. LHC_GetLastErrorCode

### 5.15.1. Format

Int16 LHC_GetLastErrorCode ()

### 5.15.2. Description

Returns the value of the last error that was encountered by the IS (Interface Software) component..

### 5.15.3. Passed Parameters

None.

### 5.15.4. Return Value

| Type | Name | Values |
|---|---|---|
| Int16 | nLastErrorCode | IS_ReturnCode. |

### 5.15.5. Comments

Once the Error Code is returned, a call to LHC_GetErrorString () will provide a descriptive string of the error.
See *Operating Scenarios* section 4.5 for related method calls.

## 5.16. LHC_GetOnBoardProtocolCount

### 5.16.1. Format

Int16 LHC_GetOnBoardProtocolCount (ref Int16 nCount)

### 5.16.2. Description

Returns the number of stored on-board protocols. This is an optional method that is used with LHC_GetOnBoardProtocolName when the names of the on-board protocols are not known.

### 5.16.3. Passed Parameters

| Type | Name | Values |
|---|---|---|
| Int16 | nCount | 0-n (passed by reference) |

### 5.16.4. Return Value

| Type | Name | Values |
|---|---|---|
| Int16 | nLastErrorCode | IS_ReturnCode. |

### 5.16.5. Comments

Once the Error Code is returned, a call to LHC_GetErrorString () will provide a descriptive string of the error.
See *Operating Scenarios* section 4.5 for related method calls.

## 5.17. LHC_GetOnBoardProtocolName

### 5.17.1. Format

String LHC_GetOnBoardProtocolName (Int16 nIndex)

### 5.17.2. Description

Returns the name of the requested protocol. This is an optional command and can only be used if LHC_GetOnBoardProtocolCount is called first to get the number of on-board protocols. This method is called multiple times, each time using a unique index representing each protocol in the protocol count.

### 5.17.3. Passed Parameters

| Type | Name | Values |
|---|---|---|
| Int16 | nIndex | 0-based index of on-board protocol |

### 5.17.4. Return Value

| Type | Name | Values |
|---|---|---|
| String | nProtocolName | String indicating name of requested on-board protocol.<br>(Empty string is returned if an error occurs) |

### 5.17.5. Comments

See *Operating Scenarios* section 4.5 for related method calls.

## 5.18. LHC_OverrideValidation

### 5.18.1. Format

Int16 LHC_OverrideValidation (Int16 nOverride)

### 5.18.2. Description

Each time the LHC_RunProtocol () method is called, the LHC_Runner by default performs a pre-run validation sequence. It will test the selected communication port, determine the current configuration of the specified instrument, and validate the selected protocol to ensure it will run on the instrument. If the integrator is running the loaded protocol more than once, then time (up to several seconds) can be saved by skipping this pre-run validation sequence after the first time it is validated and run. If the protocol validation is skipped for a protocol that was not designed for this instrument, then the processing outcome is uncertain. For more details on overriding validation, see section 3.4 Optimizing Run Time. *This feature is not supported by the ELx405 or the MicroFlo instruments.*

### 5.18.3. Passed Parameters

| Type | Name | Values |
|---|---|---|
| Int16 | nOverride | 0 (perform the default validation sequence)<br>1 (omit the validation sequence) |

### 5.18.4. Return Value

| Type | Name | Values |
|---|---|---|
| Int16 | nReturnCode | Runner_ReturnCode |

### 5.18.5. Comments

If the nReturnCode is not eOK, then make a call to LHC_GetErrorString (nReturnCode) to get a string that describes the error.
See *Operating Scenarios* section 4.3 and 4.4 for related method calls.

## 5.19. LHC_LeaveVacuumPumpOn

### 5.19.1. Format

Int16 LHC_LeaveVacuumPumpOn (Int16 nVacPumpOn)

### 5.19.2. Description

When a protocol run is complete, by default the vacuum pump is turned off. In some cases, depending on the type of vacuum pump and vacuum bottle size, it cannot be started again until a vacuum dissipate time duration is complete. If you are running protocols back-to-back, this unwanted delay can be bypassed by leaving the vacuum pump on when the protocol is done. If a protocol is complete and the vacuum pump is currently on (left on by a previous LHC_LeaveVacuumPumpOn(1) call), then making a call to LHC_LeaveVacuumPumpOn(0) will immediately turn the pump off.

### 5.19.3. Passed Parameters

| Type | Name | Values |
|---|---|---|
| Int16 | nVacPumpOn | 0 (turn vacuum pump off when protocol run is finished)<br>1 (leave pump on when protocol run is finished) |

### 5.19.4. Return Value

| Type | Name | Values |
|---|---|---|
| Int16 | nReturnCode | IS_ReturnCode |

### 5.19.5. Comments

If the nReturnCode is not eOK, then make a call to LHC_GetErrorString (nReturnCode) to get a string that describes the error.
See *Operating Scenarios* section 4.3 and 4.4 for related method calls.

## 5.20. LHC_RunVerifyManifoldTest

### 5.20.1. Format

Int16 LHC_RunVerifyManifoldTest ()

### 5.20.2. Description

Runs the Verify Manifold Test to check for clogged or impaired aspirate and dispense manifold tubes.

### 5.20.3. Passed Parameters

None.

### 5.20.4. Return Value

| Type | Name | Values |
|---|---|---|
| Int16 | nRunStatus | RunStatus |

### 5.20.5. Comments

This command starts the Verify Manifold Test. To determine when the test is done, the LHC_GetVerifyManifoldRunStatus command must be issued repeatedly (polled). See the LHC_GetProtocolStatus command description for information about threads and multi-thread models. When the status is no longer *Busy*, a call to LHC_GetVerifyManifoldTestResults must be made to write out the results toa raw data file and to retrieve the test result.
See Section 6.4 on interpreting the RunStatus.
See *Operating Scenarios* section 4.6 for related method calls.

## 5.21. LHC_GetVerifyManifoldRunStatus

### 5.21.1. Format

Int16 LHC_GetVerifyManifoldRunStatus ()

### 5.21.2. Description

Gets the status of the currently executing Verify Manifold test.

### 5.21.3. Passed Parameters

None.

### 5.21.4. Return Value

| Type | Name | Values |
|---|---|---|
| Int16 | nRunStatus | RunStatus or IS_ReturnCode. |

### 5.21.5. Comments

The LHC_GetVerifyManifoldRunStatus command must be issued repeatedly (polled) until the test completes. In a single thread model, it is best to perform this by using a Timer Tick event handler and not a Thread.Sleep command. Thread.Sleep commands should only be used if the protocol is run in its own thread and a proper multi-thread model is used. See the latest LHC_Caller sample application for example of multi-threaded models.
If the RunStatus indicates that the test is complete (either successfully or because of an error, then a call should be made to LHC_GetVerifyManifoldTestResults so that the test data is written to file.
See Section 6.4 on interpreting the RunStatus.
See *Operating Scenarios* section 4.6 for related method calls.

## 5.22. LHC_GetVerifyManifoldTestResults

### 5.22.1. Format

string LHC_GetVerifyManifoldTestResults ()

### 5.22.2. Description

Gets the test results of the Verify Manifold test just completed.

### 5.22.3. Passed Parameters

None.

### 5.22.4. Return Value

| Type | Name | Values |
|---|---|---|
| string | strResults | The string to contain the results of the Verify Manifold test |

### 5.22.5. Comments

When a call to LHC_GetVerifyManifoldTestResults is made, the test data is written to file and the test results in the form of a string is returned. The possibilities are:
 “PASSED”
 “FAILED_Step_x” where x is the step number that failed
 “ERROR xxxx” where xxxx is the error code number

If the results is not “PASSED”, the user may need to perform maintenance on the manifold as described in the LHC help system.
See *Operating Scenarios* section 4.6 for related method calls.

## 5.23. LHC_CreateVerifyManifoldTestReport

### 5.23.1. Format

string LHC_CreateVerifyManifoldTestReport ()

### 5.23.2. Description

Creates a text-formatted Report file for the Verify Manifold test just completed.

### 5.23.3. Passed Parameters

None.

### 5.23.4. Return Value

| Type | Name | Values |
|---|---|---|
| string | strReportName | A string that contains the name of the Report file. |

### 5.23.5. Comments

When a call to LHC_CreateVerifyManifoldTestReport is made, a text-formatted Report file is created and the full pathname of the Report is returned.
See *Operating Scenarios* section 4.6 for related method calls.

## 5.24. LHC_SetProductName

### 5.24.1. Format

Int16 LHC_SetProductName (String strProductName)

### 5.24.2. Description

[*This replaces the LHC_SetProductType method*]
Sets the LHC instrument type based on the Product Name string passed in.

### 5.24.3. Passed Parameters

| Type | Name | Values |
|---|---|---|
| String | strProductName | ProductName |

### 5.24.4. Return Value

| Type | Name | Values |
|---|---|---|
| Int16 | nReturnCode | Runner_ReturnCode |

### 5.24.5. Comments

If the nReturnCode is not eOK, then make a call to LHC_GetErrorString (nReturnCode) to get a string that describes the error.
See *Operating Scenarios* section 4.1 for related method calls.

## 5.25. LHC_GetWellCount

### 5.25.1. Format

Int16 LHC_GetWellCount ()

### 5.25.2. Description

This method will return the well count of the plate type specified in the loaded protocol. The protocol must be loaded prior to making this call.
(*This feature is not supported by the ELx405 or the MicroFlo instruments).*

### 5.25.3. Return Value

| Type | Name | Values |
|---|---|---|
| Int16 | nWellCount | 6, 12, 20, 24, 48, 96, 386, 1536, or 0 if the plate type is not determined |

### 5.25.4. Comments

The protocol must be loaded prior to making this call.

## 5.26. LHC_ValidateProtocol

### 5.26.1. Format

Int16 LHC_ValidateProtocol (bool bGetSettingsFirst)

### 5.26.2. Description

Each time the LHC_ValidateProtocol() method is called, the LHC_Runner performs a validation of the loaded protocol. It will test the selected communication port, determine the current configuration of the specified instrument, and validate the selected protocol to ensure it will run on the instrument. The *bGetSettingsFirst* parameter is used to retrieve the instrument's settings prior to performing validation. If this flag is set to false, validation is performed against the settings that are stored in the protocol file and does not insure that the protocol will run successfully on the instrument. It is advised to always have the *bGetSettingsFirst* parameter set to **true**. See how this command should be used when overriding validation by reading the description for the **LHC_OverrideValidation**() method in section 5.18. For more details on overriding validation, see section 3.4 Optimizing Run Time.
(*This feature is not supported by the ELx405 or the MicroFlo instruments).*

### 5.26.3. Passed Parameters

| Type | Name | Values |
|---|---|---|
| bool | bGetSettingsFirst | false (settings are **not** retrieved from the instrument)<br>true (settings **are** retrieved from the instrument) |

### 5.26.4. Return Value

| Type | Name | Values |
|---|---|---|
| Int16 | nReturnCode | Runner_ReturnCode |

### 5.26.5. Comments

The protocol must be loaded prior to making this call.
If the nReturnCode is not eOK, then make a call to LHC_GetErrorString (nReturnCode) to get a string that describes the error.
See *Operating Scenarios* section 4.3 and 4.4 for related method calls.

## 5.27. LHC_SetRunnerThreading

### 5.27.1. Format

Int16 LHC_SetRunnerThreading (Int16 nThreadingModel)

### 5.27.2. Description

The LHC_SetRunnerThreading () method allows the integrator to specify if the LHC_Runner performs its tasks: in the caller's thread or in its own thread. Having the LHC_Runner use its own thread may be beneficial in some circumstances depending on how the integrator's calling routine is written. By default, it uses the caller's thread. When a command is sent to the LHC_Runner that requires interfacing with the instrument, the LHC_Runner will evaluate the threading flag. If the flag is set, the LHC_Runner will create a thread and run the command in that thread. If the flag is not set, the LHC_Runner will simply run the command without a thread.

### 5.27.3. Passed Parameters

| Type | Name | Values |
|---|---|---|
| Int16 | nThreadingModel | 0 (the LHC_Runner does **not** run in its own thread)<br>1 (the LHC_Runner **does** run in its own thread) |

### 5.27.4. Return Value

| Type | Name | Values |
|---|---|---|
| Int16 | nReturnCode | Runner_ReturnCode |

### 5.27.5. Comments

If the nReturnCode is not eOK, then make a call to LHC_GetErrorString (nReturnCode) to get a string that describes the error.

## 5.28. LHC_SetFirstStrip (50 TS washer only)

### 5.28.1. Format

Int16 LHC_SetFirstStrip (byte byFirstStrip)

### 5.28.2. Description

This method is for the BioTek 50 TS washer only. See section 3.8 for details. An error will be returned if it is used for any other BioTek instrument.
The 50 TS allows the integrator to specify which strips (columns of wells) are to be processed by the protocol at runtime. When a protocol is loaded, by default all strips (columns) are selected for processing. The LHC_SetFirstStrip () method is used to set the first strip in a range to be processed. This command must be used with the LHC_SetNumberOfStrips() method.

### 5.28.3. Passed Parameters

| Type | Name | Values |
|---|---|---|
| byte | byFirstStrip | A valid strip number for the plate being used |

### 5.28.4. Return Value

| Type | Name | Values |
|---|---|---|
| Int16 | nReturnCode | Runner_ReturnCode or IS_ReturnCode. |

### 5.28.5. Comments

If the nReturnCode is not eOK, then make a call to LHC_GetErrorString (nReturnCode) to get a string that describes the error.
See *Operating Scenarios* section 4.3 and 4.4 for related method calls.

## 5.29. LHC_SetNumberOfStrips (50 TS washer only)

### 5.29.1. Format

Int16 LHC_SetNumberOfStrips (byte byNumberOfStrips)

### 5.29.2. Description

This method is for the BioTek 50 TS washer only. See section 3.8 for details. An error will be returned if it is used for any other BioTek instrument.
The 50 TS allows the integrator to specify which strips (columns of wells) are to be processed by the protocol at runtime. When a protocol is loaded, by default all strips (columns) are selected for processing. The LHC_SetNumberOfStrips () method is used to specify the number of strips to process starting with the First Strip specified by the LHC_SetFirstStrip method.
This command must be used with the LHC_SetFirstStrip () method.

### 5.29.3. Passed Parameters

| Type | Name | Values |
|---|---|---|
| byte | byNumberOfStrips | The number of strips to process |

### 5.29.4. Return Value

| Type | Name | Values |
|---|---|---|
| Int16 | nReturnCode | Runner_ReturnCode or IS_ReturnCode. |

### 5.29.5. Comments

If the nReturnCode is not eOK, then make a call to LHC_GetErrorString (nReturnCode) to get a string that describes the error.
See *Operating Scenarios* section 4.3 and 4.4 for related method calls.

# 6. Enumerations

## 6.1. ProductType < < < *NO LONGER USED* > > >

*[NOTE: These are no longer used. You should now use the "LHC_SetProductName" method]*

The current values for Product Type are:

| | |
|---|---|
| eUndefined | = 0 |
| eEL406 | = 1 |
| eELx405 | = 2 |
| eMicroFlo | = 3 |
| eMultiFlo | = 4 |
| e405TSLS | = 5 |
| eMultiFloFX | = 6 |

## 6.2. Runner_ReturnCode

These codes are generated by the LHC_Runner software component.
The current values for Runner_ReturnCode are:

| | |
|---|---|
| eError | = 0 |
| eOK | = 1 |
| eRegistration_Failure | = 2 |
| eInterface_Failure | = 3 |
| eInvalid_Product_Type | = 4 |
| eOpen_File_Error | = 5 |
| ePre_Run_Error | = 6 |

## 6.3. IS_ReturnCode

These codes are generated by the IS (Interface Software) components. The return codes for each IS component is not listed here. These codes do not overlap with the Runner_ReturnCode value with the exception that they both contain "eOK = 1". You can pass any return code to the LHC_GetErrorString() method to get a description of the error that occurred.

## 6.4. RunStatus

The current values for RunStatus are:

| | |
|---|---|
| eUninitialized | = 0 |
| eReady | = 1 |
| eNotReady | = 2 |
| eBusy | = 3 |
| eError | = 4 |
| eDone | = 5 |
| eIncomplete | = 6 |
| ePaused | = 7 |
| eStopRequested | = 8 |
| eStopping | = 9 |
| eNotRequired | = 10 |

The only LHC_Runner calls that return a **RunStatus** value are:

| | |
|---|---|
| **LHC_RunProtocol** | **LHC_GetProtocolStatus** |
| **LHC_RunVerifyManifoldTest** | **LHC_GetVerifyManifoldTestStatus** |

The following information is further detail on how to interpret a **RunStatus** value.

The **RunStatus** returned by a call to **LHC_RunProtocol** should be evaluated as follows:

| | | |
|---|---|---|
| *eBusy* | (= 3) | the protocol has started properly: start polling for status using a call to the **LHC_GetProtocolStatus** method. There is no benefit to polling more frequently than once every 500 msec. |
| (*any other value*) | | protocol has failed to start properly: do not start polling for status. |

The **RunStatus** returned by a call to **LHC_RunVerifyManifoldTest** should be evaluated as follows:

| | | |
|---|---|---|
| *eBusy* | (= 3) | the test has started properly: start polling for status using a call to the **LHC_GetVerifyManifoldTestStatus** method. There is no benefit to polling more frequently than once every 500 msec. |
| (*any other value*) | | the test has failed to start properly: do not start polling for status. |

The **RunStatus** returned by a call to **LHC_GetProtocolStatus** or **LHC_GetVerifyManifoldTestStatus** should be evaluated as follows:

| | | |
|---|---|---|
| *eUninitialized* | (= 0) | should never be encountered |
| *eReady* | (= 1) | the run completed successfully: stop polling for status |
| *eNotReady* | (= 2) | failed to run a new step: stop polling for status, the run has failed |
| *eBusy* | (= 3) | busy running current step: the run is still active, keep polling for further status |
| *eError* | (= 4) | this run has an error: stop polling for status, the run has failed. Call **LHC_GetLastErrorCode** to get the error code and then **LHC_GetErrorString** to get the error description |
| *eDone* | (= 5) | the current step is done, the next step will be started automatically: the run is still active, keep polling for further status |
| *eIncomplete* | (= 6) | (not used) |
| *ePaused* | (= 7) | the run is paused by user: the run is still active, keep polling for further status |
| *eStopRequested* | (= 8) | a *Stop* is requested by user: the run is still active, keep polling for further status |
| *eStopping* | (= 9) | the run is stopping per request: the run is still active, keep polling for further status |
| *eNotRequired* | (= 10) | (not used) |

## 6.5. ProductName

The current string values for Product Name are:

“EL406”
“405 TS/LS”
“ELx405”
“MultiFlo”
“MultiFloFX”
“MicroFlo Select”
“50 TS”
(others will be used as new instruments are released)