TELEDYNE PRINCETON INSTRUMENTS
Everywhereyoulook™

Part of the Teledyne Imaging Group

LightField® Add-ins and Automation Programming Manual

4411-0135
Issue 11
March 18, 2019

www.princetoninstruments.com

Revision History (Sheet 1 of 2)

Issue	Date	List of Changes
Issue 11	March 18, 2019	Issue 11 of this document incorporates the following changes: • Updated Sample Code descriptions for: — Acquire Async Sample; — Acquire Sync Sample; • Added the following new sections to Section 7.8, IImageDataSet: — Section 7.8.3, Metadata Structure; — Section 7.8.4, Code Example; • Added the following new sections to Section 7.11, IExperiment: — Section 7.11.5, `Pulse` Structure; — Section 7.11.6, `RegionOfInterest` Structure; — Section 7.11.7, `Modulation` Structure; • Rebranded to Teledyne PI.
Issue 10	July 3, 2018	Issue 10 of this document incorporates the following changes: • Updated the following IFileManager Method in Table 7-11, IFileManager Supported Methods, on page 58: — **From:** `GetRecentlyAcquired()` — **To:** `GetRecentlyAcquiredFileNames()`
Issue 9	January 2, 2018	Issue 9 of this document incorporates the following changes: • Corrected minor typos; • Updated copyright year.
Issue 8	January 18, 2017	Issue 8 of this document incorporates the following changes: • Updated the copyright year; • Corrected typos.
Issue 7	January 8, 2016	Issue 7 of this document incorporates the following changes: • Updated the copyright year.
Issue 6	July 17, 2015	Issue 6 of this document incorporates the following changes: • Updated front and back cover design and title page graphic to reflect LightField 5 graphic design; • Added Revision History table to front matter.

Teledyne Princeton Instruments
3660 Quakerbridge Rd
Trenton, NJ 08619
TEL: 800-874-9789 / 609-587-9797
FAX: 609-587-1970

Revision History (Sheet 2 of 2)

Issue	Date	List of Changes
Version 5	April 4, 2015 January 1, 2014 Sept. 19, 2013	Version 5 of this document incorporates the following changes: • Converted the document from Word to FrameMaker and incorporated a new template and structure; • Added Chapter 3, Add-In Code Structure, which describes the parts of an add-in (C3 as well as Visual Basic); • Added Chapter 5, Create an Add-in Using a Template; • Corrected formatting issues with graphics; • Expanded information about add-in zones.
Version 4	March 14, 2013	Version 4 of this document incorporates the following changes: • Updated copyright page to 2013; • Global update to include ® mark for LightField; • Updated other graphics (showing new ? button). • Added, move, deleted properties and methods.
Version 3	April 23, 2012	Version 3 of this document incorporates the following changes: • Updated Manage Add-ins dialog; • Added Chapter 4 Automation Class; • Added Appendix A; • Added Export Sample to list; • ModulationTrackingPhase added to enum CalibrationCategory.
Version 2	December 6, 2011	Version 2 of this document incorporates the following changes: • Updated dll and .NET versions; • Changed Calibration sample to Spectroscopy Sample; • Removed ClipYAxis, YAxisClippingEnd, and YAxisClippingStart; • Updated Viewer Sample; • Added System Building Sample.
Version 1	July 26, 2011	This is the initial release of this document.

This page is intentionally blank.

Table of Contents

Chapter 1: About this Document 9
1.1 What is an Add-In Module? 9
1.2 Minimum System Requirements 9
1.3 Related Documents 9
1.4 Document Organization 10

Chapter 2: Introduction to LightField Add-Ins 11
2.1 Managing Add-ins 11
2.1.1 Activating/Deactivating Add-Ins 12
2.2 LightField Add-in Zones 12
2.2.1 Application Toolbar Zone 14
2.2.2 Data Toolbar Zone 15
2.2.3 Application Menu Zone 16
2.2.4 Experiment Settings Zone 17
2.2.4.1 Expander Objects 17
2.2.5 Experiment View Zone 19
2.3 Sample Add-ins 21

Chapter 3: Add-In Code Structure 23
3.1 C# Add-in Program File 23
3.2 Visual Basic Add-in File 24

Chapter 4: Developer Tools and Software 25
4.1 Install the Development Environment 25
4.1.1 Developer Tools Required when Creating an Add-in from a Template 26
4.1.2 Developer Tools Required when Creating an Add-in from Scratch 26
4.2 Install LightField 27

Chapter 5: Create an Add-in Using a Template 29
5.1 Create a Visual Studio Project File 29
5.2 Build and Test the New Add-in 31
5.3 Customize Add-in Functionality 33

Chapter 6: Create Add-ins from Scratch 35
6.1 Building and Add-in 35
6.2 Building Automation Modules 37
6.3 ILightField Experiment Settings.chm 38

Chapter 7: Programming Interface 39
7.1 Microsoft Managed Add-In Framework 39
7.2 Required (Core) Methods 39
7.2.1 Activate() 39
7.2.2 Deactivate() 40
7.3 Required Properties 41
7.3.1 UISupport 41

7.4 ILightFieldAddIn . . . 42
7.4.1 Supported Methods . . . 42
7.4.2 Supported Properties . . . 44
7.4.3 Code Example . . . 46
7.4.3.1 Code Description . . . 47
7.5 ILightFieldApplication . . . 50
7.5.1 Supported Properties . . . 50
7.6 IUserInteractionManager . . . 51
7.6.1 Supported Properties . . . 51
7.7 IImageData . . . 52
7.7.1 Supported Methods . . . 52
7.7.2 Supported Properties . . . 52
7.8 IImageDataSet . . . 53
7.8.1 Supported Methods . . . 53
7.8.2 Supported Properties . . . 53
7.8.3 Metadata Structure . . . 54
7.8.4 Code Example . . . 55
7.9 IDataManager . . . 56
7.9.1 Supported Methods . . . 56
7.9.2 Code Example . . . 56
7.10 IFileManager . . . 58
7.10.1 Supported Methods . . . 58
7.10.2 Supported Events . . . 60
7.10.3 Code Sample . . . 60
7.11 IExperiment . . . 61
7.11.1 Supported Methods . . . 61
7.11.2 Supported Properties . . . 64
7.11.3 Supported Events . . . 65
7.11.4 Code Example . . . 66
7.11.5 `Pulse` Structure . . . 67
7.11.5.1 Field Information . . . 67
7.11.5.2 Code Example . . . 67
7.11.6 `RegionOfInterest` Structure . . . 68
7.11.6.1 Field Information . . . 68
7.11.6.2 Code Example . . . 69
7.11.7 `Modulation` Structure . . . 69
7.11.7.1 Field Information . . . 69
7.12 IDevice . . . 70
7.12.1 Supported Properties . . . 70
7.13 ISettingRange . . . 70
7.13.1 Supported Properties . . . 70
7.14 IDisplay . . . 71
7.14.1 Supported Methods . . . 71
7.15 IDisplaySource . . . 73
7.15.1 Supported Methods . . . 73
7.16 IDisplayViewer . . . 73
7.16.1 Supported Methods . . . 73
7.16.2 Supported Properties . . . 75
7.16.3 Code Example . . . 80
7.16.3.1 Code Description . . . 81
7.17 IDisplayViewerControl . . . 82
7.17.1 Supported Properties . . . 82
7.18 IExportError . . . 82
7.18.1 Supported Properties . . . 82

7.19 IExportSelectionError . . . 83
7.19.1 Supported Properties . . . 83
7.20 IExportSettings . . . 84
7.20.1 Supported Methods . . . 84
7.20.2 Supported Properties . . . 84
7.21 IAviExportSettings . . . 88
7.21.1 Supported Properties . . . 88
7.22 ICsvExportSettings . . . 88
7.22.1 Supported Properties . . . 88
7.23 IFitsExportSettings . . . 89
7.23.1 Supported Properties . . . 89
7.24 ISpcExportSettings . . . 90
7.24.1 Supported Properties . . . 90
7.25 ITiffExportSettings . . . 90
7.25.1 Supported Properties . . . 90
7.26 Zone Support . . . 91
7.26.1 Application Toolbar Zone . . . 91
7.26.2 Data Toolbar Zone . . . 91
7.26.3 Experiment Settings Zone . . . 91
7.26.4 Experiment View Zone . . . 91
7.26.5 Menu Zone . . . 91

Chapter 8: Automation Class . . . 93
8.1 Properties Table . . . 93
8.2 Methods Table . . . 93
8.3 Events Table . . . 94

Appendix A: What Is New in LightField 4.0 . . . 95

List of Figures

Figure 2-1: Typical Add-In Manager Dialog . . . 11
Figure 2-2: Typical Graphical User Interface Objects . . . 13
Figure 2-3: Application Toolbar Zone . . . 14
Figure 2-5: Add-in in the Application Menu Zone . . . 16
Figure 2-6: Experiment Settings Zone . . . 17
Figure 2-7: Expander Object Example 1: High Speed Camera Add-In . . . 18
Figure 2-8: Expander Object Example 2: Regions of Interest Add-In . . . 19
Figure 2-9: Experiment View Zone . . . 19
Figure 4-1: Typical LightField Installation Dialog: Custom Setup . . . 27
Figure 5-3: Typical Project Add-in Source File: LightField Add-In Template (Partial View) . . . 30
Figure 5-4: Typical LightField *Manage Add-ins...* Menu Option . . . 32
Figure 5-5: Typical Manage Add-ins Dialog . . . 32
Figure 5-6: Typical LightField Menu: Add-ins Menu . . . 33
Figure 5-7: Output of LightField Add-In . . . 33
Figure 7-1: Plot Sample Add-in Listed in the Manage Add-in Dialog . . . 48
Figure 7-2: Plot Sample Add-in in the Application Toolbar . . . 48
Figure 7-3: Application Toolbar: View Layout . . . 72
Figure 7-4: Four Plots Generated by Code Example 7-8 . . . 81

List of Tables

Revision History 2
Table 2-1: Supported GUI Objects by LightField Zone 13
Table 7-1: ILightFieldAddIn Supported Methods 42
Table 7-2: ILightFieldAddIn Supported Properties 44
Table 7-3: ILightFieldApplication Supported Properties 50
Table 7-4: IUserInteractionManager Supported Properties 51
Table 7-5: IImageData Supported Methods 52
Table 7-6: IImage Data Supported Properties 52
Table 7-7: IImageDataSet Supported Methods 53
Table 7-8: IImageDataSet Supported Properties 53
Table 7-9: IImageDataSet Metadata Structure Fields 54
Table 7-10: IDataManager Supported Methods 56
Table 7-11: IFileManager Supported Methods 58
Table 7-12: IFileManager Supported Events 60
Table 7-13: IExperiment Supported Methods 61
Table 7-14: IExperiment Supported Properties 64
Table 7-15: IExperiment Supported Events 65
Table 7-16: `Pulse` Structure Field Information 67
Table 7-17: `RegionOfInterest` Structure Field Information 68
Table 7-18: `Modulation` Structure Field Information 69
Table 7-19: IDevice Supported Properties 70
Table 7-20: ISettingRange Supported Properties 70
Table 7-21: IDisplay Supported Methods 71
Table 7-22: IDisplaySource Supported Methods 73
Table 7-23: IDisplayViewer Supported Methods 73
Table 7-24: IDisplayViewer Supported Properties 75
Table 7-25: IDisplayViewerControl Supported Properties 82
Table 7-26: IExportError Supported Properties 82
Table 7-27: IExportSelectionError Properties 83
Table 7-28: IExportSettings Supported Methods 84
Table 7-29: IExportSettings Supported Properties 84
Table 7-30: IAviExportSettings Supported Properties 88
Table 7-31: ICsvExportSettings Supported Properties 88
Table 7-32: IFitsExportSettings Supported Properties 89
Table 7-33: ISpcExportSettings Supported Properties 90
Table 7-34: ITiffExportSettings Supported Properties 90
Table 8-1: Automation Class Properties 93
Table 8-2: Automation Class Methods 93
Table 8-3: Automation Class Events 94
Table A-1: List of Changes in LightField 4.0 95

Chapter 1: About this Document

This document provides programming information for users of Teledyne Princeton Instruments' LightField® data acquisition software who wish to expand its functionality with custom add-in modules.

1.1 What is an Add-In Module?

Add-ins are user-developed and compiled software modules that provide custom functionality to existing third-party software applications. By integrating these modules into the existing application framework, the added functionality is transparent to the end-user and executes seamlessly when called upon.

1.2 Minimum System Requirements

Creating custom LightField add-in modules requires the following applications be installed on the development computer:

- Teledyne Princeton Instruments' LightField® data acquisition software;
- LightField Software Developer Kit add-in;
  The SDK and all associated sample files are located in the following directory on the host/development computer:
  ```
  C:\Users\Public\Documents\Princeton Instruments\LightField\Add-in and Automation SDK
  ```
- Visual Studio® 2010, or Visual Studio 2010 Express;
- Microsoft® .Net 4.0 Framework.

Refer to Chapter 4, Developer Tools and Software, for additional information.

1.3 Related Documents

Refer to the following documents for additional information:

- LightField Users Manual, document number 4411-0125;
- PICam® Users Manual, document number 4411-0134.

1.4 Document Organization

This document includes the following chapters and appendices:

- Chapter 2, Introduction to LightField Add-Ins;
  Provides an overview of how add-ins are implemented within LightField.
- Chapter 3, Add-In Code Structure;
  Provides a high-level breakdown of the structure of add-in program files.
- Chapter 4, Developer Tools and Software;
  Provides information about the various developer tools and software that are required to write custom add-ins for LightField.
- Chapter 5, Create an Add-in Using a Template;
  Describes the process of customizing an existing add-in template.
- Chapter 6, Create Add-ins from Scratch;
  Describes the process of creating an add-in without the benefit of an existing template.
- Chapter 7, Programming Interface;
  Provides programming information necessary when developing an add-in.
- Chapter 8, Automation Class;
  Provides programming information about the Automation Class.
- Appendix A, What Is New in LightField 4.0.
  Provides an overview of changes made as part of LightField Release 4.0.

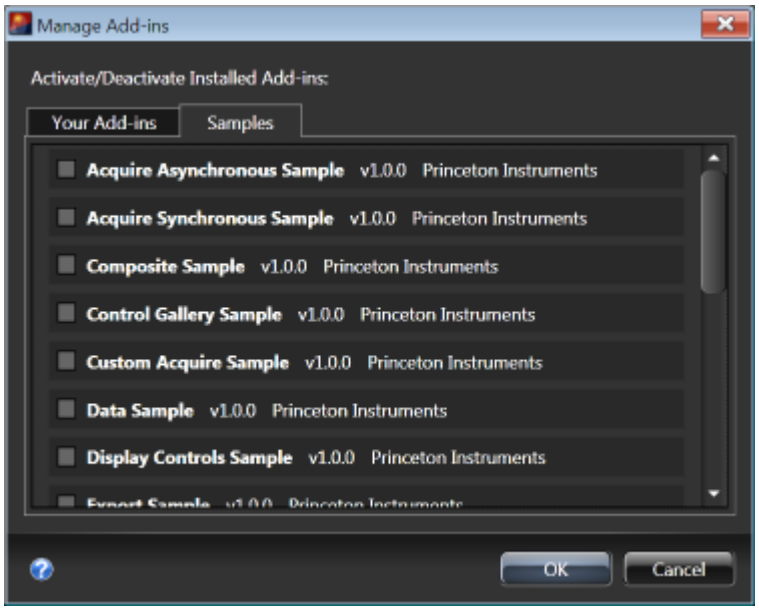
Chapter 2: Introduction to LightField Add-Ins

Microsoft's Managed Add-in Framework (MAF) allows users and third-party developers to extend LightField's capabilities. Add-ins have the potential to dynamically and dramatically change the look of the LightField application as well as extend the abilities and function of the core program.

2.1 Managing Add-ins

Individual add-in modules are activated/deactivated within a LightField session using the **Manage Add-Ins** dialog. Figure 2-1 illustrates a typical Manage Add-ins dialog.

Figure 2-1: Typical Add-In Manager Dialog



4411-0135_0001

As illustrated in Figure 2-1, add-in modules are displayed on one of two tabs:

- Your Add-Ins
  This is a list of add-ins created by users, third-parties, and/or Teledyne Princeton Instruments. Individual add-in modules are activated/deactivated based upon the currently logged-in user at the time LightField is launched.
- Samples
  This is a list of sample add-ins supplied with LightField. These add-ins typically perform trivial tasks and are provided for illustration only.

2.1.1 Activating/Deactivating Add-Ins

The activation status for each add-in is illustrated on the **Manage Add-ins** dialog as follows:

- An activated add-in displays a check in its corresponding box ☑.
- A deactivated add-in displays no check in its corresponding box ☐.

When launched, LightField checks for available Add-ins and populates each of the two tabs with an alphabetical list of supported Add-ins. All add-ins that are active for the logged-in user when LightField is launched are automatically included in the User Interface.

NOTE:

Add-ins can be activated/deactivated during an active LightField session.

Perform the following procedure to activate or deactivate add-in modules during a LightField session:

1. From the LightField menu bar, select **Application Menu ► Manage Add-ins...**. The **Manage Add-ins** dialog is displayed.
2. Left-click on the check box associated with the desired add-in to toggle its status.
3. Repeat step 2 to update each desired add-in.
4. Click **OK** to save the changes.

Each add-in will be immediately activated/deactivated within the current LightField session and the settings will be retained when LightField is closed. The next time LightField is launched on this computer by this user, the add-ins will be activated/ deactivated using these retained settings.

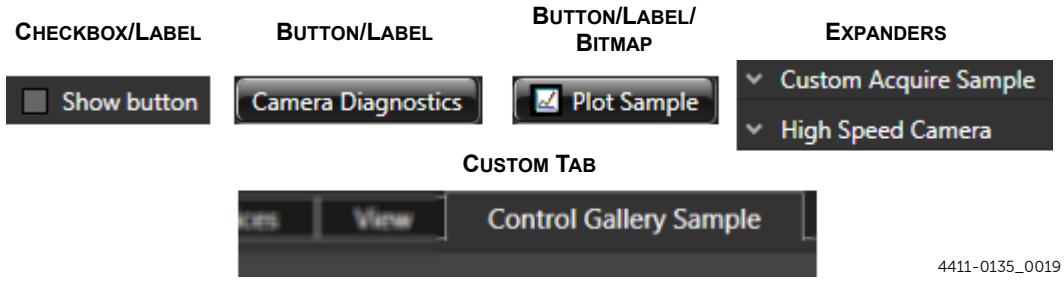
2.2 LightField Add-in Zones

In order for active add-ins to be useful, they must be accessible from within LightField. This is typically done via one or more Graphical User Interface (GUI) objects such as:

- A Checkbox with a custom text label;
  Toggles the use of the associated Add-in.
- A Button with a custom text label;
  Executes the associated Add-in routine when the button is depressed/clicked.
- A Button with a custom text label and bitmap;
  Executes the associated Add-in routine when the button is depressed/clicked.
- Custom Expanders with a custom text label;
  Provides an expanded area in which additional objects/controls may be used.
- Custom Tab with a custom text label.
  Provides the maximum flexibility and area in which additional objects/controls may be used.

Figure 2-2 illustrates examples of each GUI element.

Figure 2-2: Typical Graphical User Interface Objects



LightField supports the placement of GUI objects within the following areas, or zones:

- Application Toolbar Zone;
- Data Toolbar Zone;
- Application Menu Zone;
- Experiment Settings Zone;
- Experiment View Zone.

Depending on its scope and complexity, a single add-in may designate that:

- A single GUI object be placed in one zone; or
- Multiple GUI objects be placed in multiple zones.

The author of an add-in must determine how and where GUI objects will be placed.

It is important to understand that not all GUI objects are supported within all zones. Table 2-1 summarizes GUI object support for each zone

Table 2-1: Supported GUI Objects by LightField Zone

LightField Zone	Available GUI Objects				
	Checkbox	Button/Label	Button/Label/ Bitmap	Expander	Dedicated Tab
Application Toolbar	Yes	Yes	Yes	No	No
Data Toolbar	Yes	Yes	Yes	No	No
Application Menu	Yes	Yes	No	No	No
Experiment Settings[a]	No	No	No	Yes	No
Experiment View[b]	No	No	No	No	Yes

a. The Experiment Settings zone only supports Expander objects. However, individual Expanders support a wide variety of controls and objects. Refer to Section 2.2.4.1, Expander Objects, for complete information.
b. The Experiment View zone only supports Dedicated Tab objects. However, individual Tabs support a wide variety of controls and objects. Refer to Section 2.2.5, Experiment View Zone, for additional information.

Additional information about each LightField zone and their supported objects is provided in the following sections.

REFERENCES:

For detailed information about implementing supported objects within a zone, refer to Chapter 7, Programming Interface.

2.2.1 Application Toolbar Zone

The Application Toolbar zone is located along the top of the LightField application window directly above the Application Menu  . Objects for application-wide add-ins are displayed within this zone and are always available whenever an application-wide add-in is active.

NOTE:

The Application Toolbar zone is collapsed/hidden when no application-wide add-ins are activated.

The following objects may be displayed within the Application Toolbar zone:

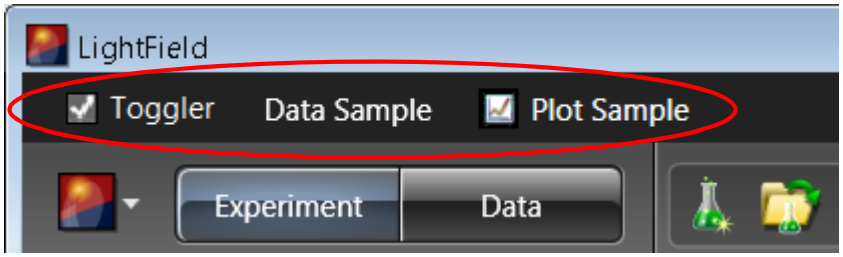
- Checkbox with custom label;
- Button with custom label;
- Button with custom label and bitmap.

NOTE:

A maximum of one object per add-in may be displayed within this zone. Additional objects for the same add-in may be displayed within other zones.

Figure 2-3 illustrates an Application Toolbar zone with objects for three (3) active add-ins.

Figure 2-3: Application Toolbar Zone



4411-0135_0002

2.2.2 Data Toolbar Zone

The Data Toolbar zone is located to the far right of the Data Workspace button 

and is visible only when the Data Workspace button has been clicked. Typically, this zone displays objects for add-ins that perform post-processing and/or data manipulation functions.

The following objects may be displayed within the Data Toolbar zone:

- Checkbox with custom label;
- Button with custom label;
- Button with custom label and bitmap.

NOTE:

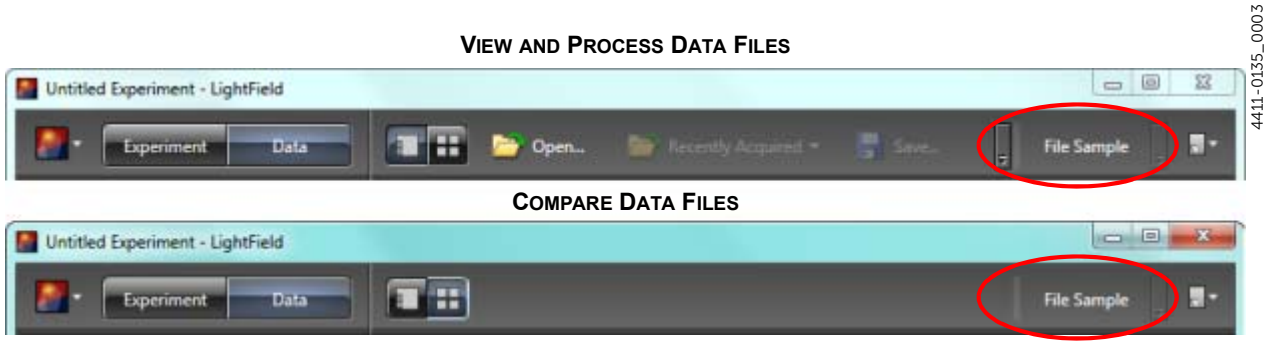
A maximum of one object per add-in may be displayed within this zone. Additional objects for the same add-in may be displayed within other zones.

The set of objects displayed varies depending on the view selected:

- View and Process Data Files  ;
- Compare Data Files  .

Figure 2-4 shows the Data Toolbar Zone for each of these views with a Button/Custom Label object displayed for a single add-in.

Figure 2-4: Typical Data Toolbar Zone



2.2.3 Application Menu Zone

The Applications Menu Zone is accessed from within the Application Menu  . Once an add-in is activated that designates an object for placement on the Application Menu, the **Application Menu ► Add-ins** option becomes active. If no appropriate add-ins are active, this menu option cannot be selected (i.e., it remains grayed out.)

The following objects may be displayed within the Application Menu zone:

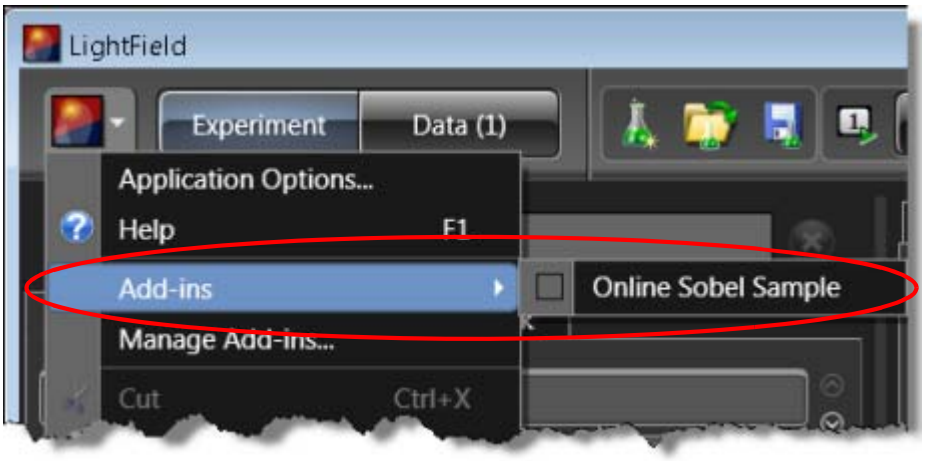
- Checkbox with custom label;
- Button with custom label.

NOTE:

A maximum of one object per add-in may be displayed within this zone. Additional objects for the same add-in may be displayed within other zones.

Figure 2-5 illustrates an active **Add-ins** menu with a single Checkbox add-in object.

Figure 2-5: Add-in in the Application Menu Zone



4411-0135_0004

2.2.4 Experiment Settings Zone

The Experiment Settings zone is part of the standard Expander Stack. Once an add-in is activated that designates an Expander object for this zone, a third tab, labeled **Add-ins**, is created and displayed to the right of the **Experiment Settings** and **Setting Dock** tabs. The **Add-ins** tab utilizes the same Expander Stack protocol as is used by the other two tabs.

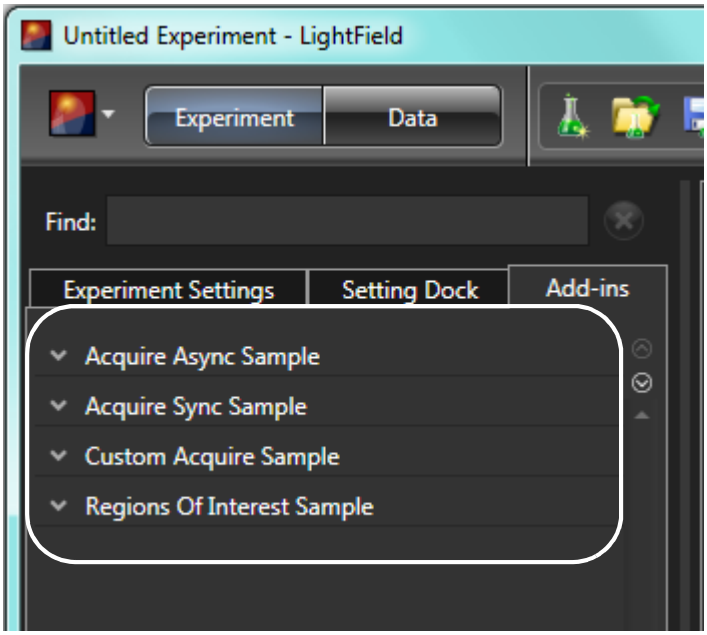
NOTE:

The only object that may be designated for use on the Experiment Settings zone is the Expander object.

A maximum of one Expander per add-in may be displayed within this zone. However, objects for the same add-in may be placed within this expander.

Figure 2-6 illustrates a typical Experiment Settings zone with Expanders for four add-ins.

Figure 2-6: Experiment Settings Zone



4411-0135_0005

2.2.4.1 Expander Objects

Expander objects are typically used by complex add-ins that require an increased level of user input/interaction beyond what is typically possible using a single checkbox and/or button object. Expanders are essentially data input forms.

For example, an add-in may require the configuration of several system parameters prior to its being used. By incorporating Expander objects, complex add-ins are able to incorporate any number and combination of objects ranging from a single Button to an entire custom User Interface.

Typical objects/controls that may be included on an Expander include:

- Custom text;
- Checkbox with custom label;
- Button with custom label;
- Button with custom label and bitmap;
- Custom pull-down menus;
- Text fields;
- Radio buttons;
- Hyperlinked text.

The only constraint is that the width of each control should be fairly narrow since a maximum object width is imposed by LightField.

Additionally, by default, all objects and controls inherit the standard LightField style. However, if desired, add-in objects can be designed with their own custom style(s).

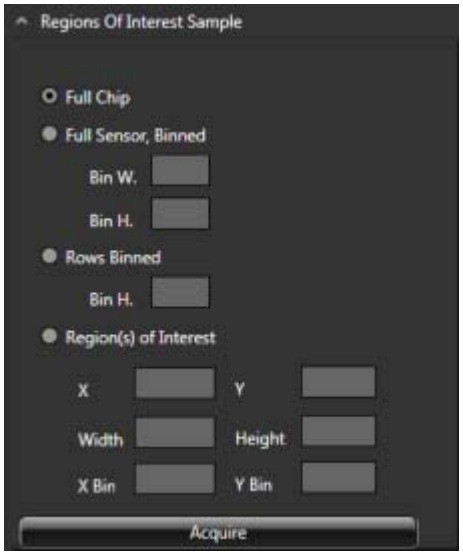
Figure 2-7 illustrates an Expander object for a custom High Speed Camera add-in.

Figure 2-7: Expander Object Example 1: High Speed Camera Add-In

4411-0135_0006

Figure 2-8 illustrates an Expander object for a custom Regions of Interest add-in.

Figure 2-8: Expander Object Example 2: Regions of Interest Add-In



4411-0135_0020

2.2.5 Experiment View Zone

The Experiment View Zone is part of LightField's Experiment Workspace. Once activated, an add-in designating the Experiment View zone for use creates a dedicated tab which is then displayed to the right of the standard **Devices** and **View** tabs. The add-in specifies the label text that is displayed on the tab.

NOTE:

The only object that may be designated for use on the Experiment View zone is the Tab object.

A maximum of one Tab per add-in may be displayed within this zone. However, objects for the same add-in may be placed within this tab.

The set of add-in tabs is displayed in alphabetical order, from left to right, based on the Add-ins' names as listed within the **Add-in Manager** dialog.

Figure 2-9 illustrates the location of a typical add-in tab.

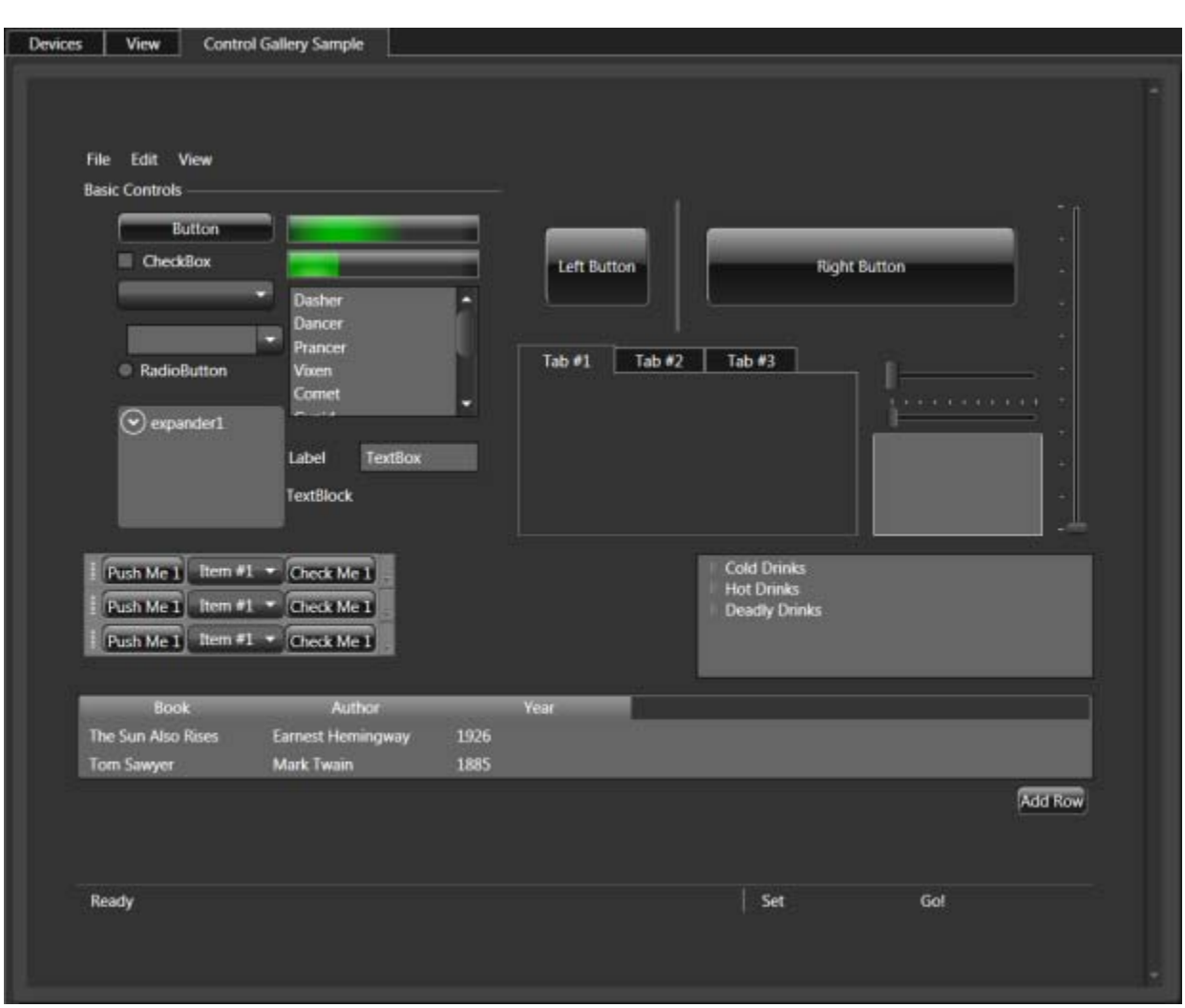
Figure 2-9: Experiment View Zone

4411-0135_0007

The Experiment View zone is ideal for complex add-ins requiring more space than is available within the Experiment Settings zone. By default, all objects and controls inherit the standard LightField style. However, if desired, add-in objects can be designed with their own custom style(s).

Figure 2-10 illustrates the **Control Gallery Sample** add-in which displays available objects and their default appearance.

Figure 2-10: Experiment View Zone: Control Gallery Add-in



2.3 Sample Add-ins

The following samples are supplied with LightField. [Term] indicates where the controls for the Add-in appear. Note that all samples have been built using Visual C#.

- Acquire Async Sample

  Illustrates:

  — Sample code for acquiring events and how to acquire data via the events;
  — Data gathering from each acquired image;
  — Naming and saving each acquired file through experiment settings;
  — How to wait for acquisition to complete (in an asynchronous deferred manner) before starting a new acquisition.

  [Experiment Settings]
- Acquire Sync Sample

  Illustrates:

  — Acquisition by calling the Capture function which waits for completion before starting the next acquisition;
  — Data gathering from each acquired image;
  — Naming each acquired file through experiment settings.

  [Experiment Settings]
- Composite Sample

  Shows how to dynamically change Add-ins components that are shown in the five LightField Add-in zones.

  [Experiment View]
- Control Gallery Sample

  Shows all of the Windows Presentation Foundation (WPF) components in a window to demonstrate how well the styles have been applied.

  [Experiment View]
- Custom Acquire Sample

  Demonstrates acquiring with a set exposure, file name and some other regularly used parameters.

  [Experiment Settings]
- Data Sample

  Generates data in memory and dumps it into data views for display.

  [Application Toolbar]
- Display Controls Sample

  Demonstrates most of the controls for adjustments on a view of data (i.e., source, zoom, type, cursor, etc.)

  [Experiment Settings]
- Export Sample

  Demonstrates exporting routines, objects and errors associated with performing exports.

  [Application Toolbar]
- File Sample

  Generates data in memory and stores the data in two SPE files in the user-designated location. One file contains multiple frames with a single region, and the other contains a single frame with multiple regions are displayed.

  [Data Toolbar]
- Metadata Sample

Demonstrates metadata tagging and how to access the timestamps on the data after acquisition on a frame by frame basis.

[Experiment Settings]

- Modal Dialog Sample

  Displays a modal dialog with a variety of WPF components.

  [Application Toolbar]
- Modeless Dialog Sample

  Displays a modeless dialog with a variety of WPF components.

  [Application Toolbar]
- Online Sobel Sample

  Hooks into the data path and performs edge detection on the buffers before the application displays or saves them. A camera must be active and the Add-in must be selected before acquisition begins.

  [Application Menu]
- Plot Sample

  Generates data in memory and plots to Data view. Also shows putting more than one source on a view.

  [Application Toolbar]
- Regions Of Interest Sample

  Demonstrates the modes of regions: full frame, binned, line sensor and custom.

  [Experiment Settings]
- Setting Snoop Sample

  Demonstrates an Add-in with no user interface that registers for setting changed events and logs them.
- Spectroscopy Sample

  Demonstrates how to read the existing calibration data out of an existing SPE file as well as how to create and display your own calibrated data files.

  [Application Toolbar]
- System Building Sample

  Demonstrates how to determine Available Devices as well as check the Devices in Use.

  [Application Toolbar]
- Viewer Sample

  Demonstrates opening a file through LightField and then putting it into a custom WPF view as well as using a LightField view.

  [Experiment View]

Chapter 3: Add-In Code Structure

This chapter provides general information about the overall structure of add-in code from a high-level viewpoint. Information about both C# and Visual Basic code is provided.

3.1 C# Add-in Program File

Code Example 3-1 illustrates a typical LightField add-in that has been written in C# with key blocks of code indicated.

Code Example 3-1: Typical C# Add-In Code

```
using System;
using System.Windows;
using System.Windows.Controls;
using System.AddIn;
using PrincetonInstruments.LightField.AddIns;

namespace CutAndPasteSample
{
    ////////////////////////////////////////////////////////////////////////////
    [AddIn("My Sample",
    Version = "1.0.0",
    Publisher = "Teledyne Princeton Instruments",
    Description = "This is my sample")]

    public class AddinMySample : AddInBase, ILightFieldAddIn
    {
        private Button button_;
        ////////////////////////////////////////////////////////////////////////
        public UISupport UISupport { get { return UISupport.ExperimentSetting; } }
        ////////////////////////////////////////////////////////////////////////
        public void Activate(ILightFieldApplication app)
        {
            // Capture Interface
            LightFieldApplication = app;
            // Simple button
            button_                   = new Button();
            button_.Content           = "Push Me!";
            ExperimentSettingElement = button_;
            // event hander
            button_.Click += new RoutedEventHandler(button__Click);
            // Base Initialize
            Initialize(button_.Dispatcher, "My Sample");
        }
        ////////////////////////////////////////////////////////////////////////
        void button__Click(object sender, RoutedEventArgs e)
        {
            MessageBox.Show("My Addin Message!");
        }
        ////////////////////////////////////////////////////////////////////////
        public void Deactivate() { }
        ////////////////////////////////////////////////////////////////////////
        public override string UIExperimentSettingTitle{get{return "My Sample";}}
    }
}
```

REFERENCE STATEMENTS

NAME SPACE

CLASS ATTRIBUTES

CLASS DERIVATIONS

MEMBER VARIABLES

METHODS

PROPERTIES

3.2 Visual Basic Add-in File

Code Example 3-2 illustrates a typical LightField add-in that has been written in Visual Basic with key blocks of code indicated.

Code Example 3-2: Typical Visual Basic Add-In Code

```
Imports System.AddIn
Imports System.Drawing
Imports System.Windows
Imports PrincetonInstruments.LightField.AddIns
' this class exposes the add-in to lightfield to simplify the implementation of
    the add-in itself
<AddIn(
    "MySample",
    Description:="Description Placeholder",
    Version:="1.0.0",
    Publisher:="Publisher Placeholder")>
Public Class MySampleAddInAdapter

    Implements ILightFieldAddIn

    Private impl_ As MySampleAddIn
    Public Sub Activate(ByVal lightField As ILightFieldApplication) Implements
        ILightFieldAddIn.Activate
        impl_ = New MySampleAddIn
        impl_.Activate(lightField)
    End Sub
    Public Sub Deactivate() Implements ILightFieldAddIn.Deactivate
        impl_.Deactivate()
    End Sub
        ' Common Code Block ...
End Class
' this class is the add-in
Public Class MySampleAddIn

    Inherits AddInBase
    ' this is called when lightfield loads this add-in
    Public Sub Activate(ByVal lightField As ILightFieldApplication)
        LightFieldApplication = lightField
    End Sub
    ' this is called when lightfield unloads this add-in
    Public Sub Deactivate()
    End Sub
    ' this tells lightfield to access this add-in via a menu
    Public ReadOnly Property UISupport As UISupport
        Get
            Return UISupport.Menu
        End Get
    End Property
    ' Code that supports the necessary functions based on the UISupport flags
        returned above
End Class
```

REFERENCE STATEMENTS

ADAPTER CLASS ADD-IN ATTRIBUTES

ADAPTER MUST IMPLEMENT ILIGHTFIELDADDIN

ADAPTER CONTAINS ADD-IN CLASS REFERENCE FOR FORWARDING

FORWARD TO ADD-IN CLASS

ADD-IN CLASS

ADD-IN CLASS MUST INHERIT FROM ADDINBASE

ADD-IN PROPERTIES

Chapter 4: Developer Tools and Software

This chapter provides information about the tools and software applications necessary for developing custom add-ins and automation modules.

Before any development can begin, the following developer tools and software applications must be installed:

- The appropriate Development Environment;

  The required tools depend on whether the add-in or automation tool will be created/developed from scratch, or developed using a template.
- LightField with SDK support.

4.1 Install the Development Environment

Subsequent chapters within this manual provide information about creating LightField add-ins and automation modules using two similar, yet very different, methods:

- Using a supplied template which is then customized

  This method is recommended for users who may not be comfortable with developing add-ins from the ground up. It is also recommended when an add-in must be created within a very short amount of time.

  Refer to Chapter 5, Create an Add-in Using a Template, for information about customizing an existing template.
- Creating an Add-in or Automation Module From Scratch.

  This method is recommended for users with prior add-in development and/or programming experience.

  Refer to Chapter 6, Create Add-ins from Scratch, for information about creating a new add-in without the aid of an existing template.

The set of developer tools required for each of these methods may, at first glance, appear to be the same. However, they are different, so be sure to refer to the appropriate set of requirements based on the approach that will be used:

- When starting with an existing template, refer to Section 4.1.1, Developer Tools Required when Creating an Add-in from a Template, for the list of developer tools required;
- When creating an add-in from scratch, refer to Section 4.1.2, Developer Tools Required when Creating an Add-in from Scratch, for the list of developer tools required.

NOTE:

The same set of development tools is required for both add-in development as well as automation module development.

4.1.1 Developer Tools Required when Creating an Add-in from a Template

NOTE:

For proper interaction and operation, the developer tools listed in this section must be completely installed prior to installing LightField.

When developing a LightField add-in based on an existing template, the following developer tools are required:

- **Microsoft Visual Studio 2010 Express**;
  Install one, or both, of the following free developer tools:
  - **Visual Basic 2010 Express**;
    Install this tool when developing add-ins using Visual Basic.
  - **Visual C# 2010 Express**.
    Install this tool when developing add-ins using Visual C#.

  These two developer tools are available for download at the following URL:

  http://www.microsoft.com/visualstudio/eng/downloads#d-2010-express

  If necessary, download the desired tools and install them per manufacturer instructions.
- **.Net Framework 4**.
  This is available for download at the following URL:

  http://www.microsoft.com/en-us/download/details.aspx?id=17851

  If necessary, download this and install it per manufacturer instructions.

4.1.2 Developer Tools Required when Creating an Add-in from Scratch

When creating a LightField add-in without the aid of a template, the following developer tools are required:

- **Microsoft Visual Studio 2010**;
- **.Net Framework 4**.
  This is available for download at the following URL:

  http://www.microsoft.com/en-us/download/details.aspx?id=17851

  If necessary, download this and install it per manufacturer instructions.

4.2 Install LightField

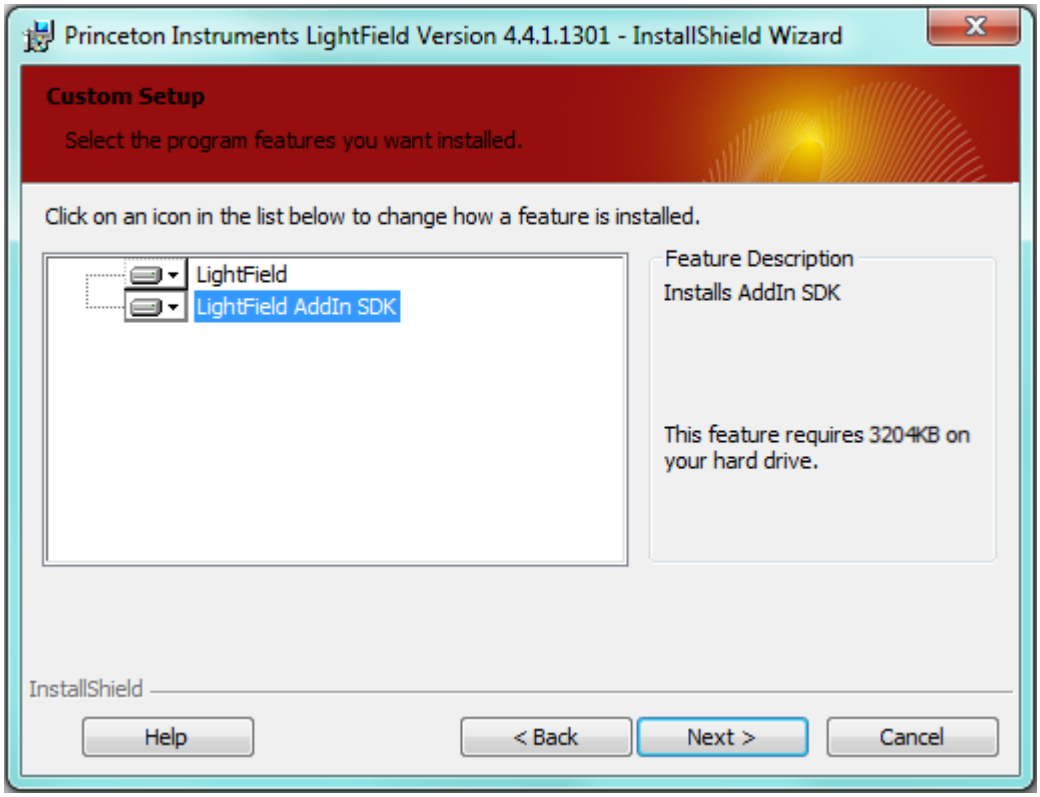
NOTE:

For proper interaction and operation when developing an add-in from a template, the developer tools listed in Section 4.1, Install the Development Environment, must be completely installed **prior** to installing LightField.

In addition to the required development tools, LightField must be installed with the option to include the AddIn SDK.

- When performing a complete LightField installation, the AddIn SDK option is included as part of the installation;
- When performing a Custom setup, the AddIn SDK option must be manually selected. See Figure 4-1.

Figure 4-1: Typical LightField Installation Dialog: Custom Setup



4411-0135_0013

Complete the installation as prompted by the Install Wizard.

This page is intentionally blank.

Chapter 5: Create an Add-in Using a Template

This chapter provides information about creating LightField add-in modules using pre-installed templates. This is particularly useful for new users or for those applications when an add-in is required to be developed in a very short period of time.

5.1 Create a Visual Studio Project File

Visual Studio Express (VS) includes a set of templates for a variety of standard applications which are then used to create new Project Files. When LightField is installed with the SDK option, a custom LightField template is added to this set of templates which can then be used when creating new LightField add-ins.

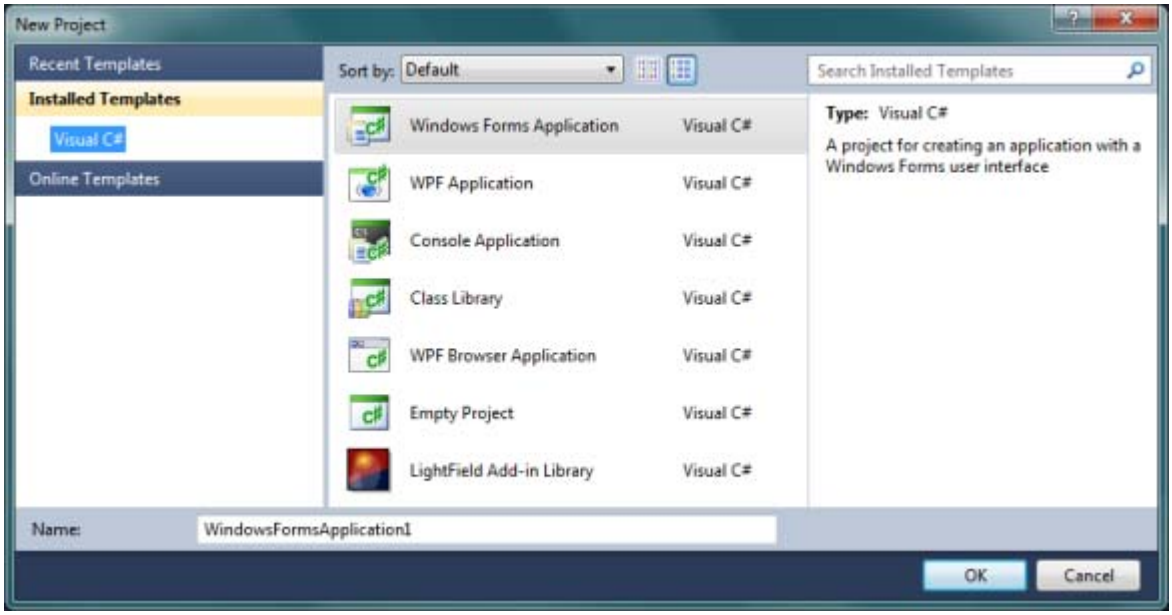
NOTE:

The LightField template included as part of the SDK creates a control that is located in the Add-in Menu zone. Refer to Section 2.2.3, Application Menu Zone, for more information.

Perform the following procedure to create a new VS project file for LightField:

1. Launch the desired Visual Studio 2010 Express tool based on the language being used:
   - Visual Basic 2010 Express, or
   - Visual C# 2010 Express.
2. From the VS pull-down menu, select **File ► New Project...**. See Figure 5-1.

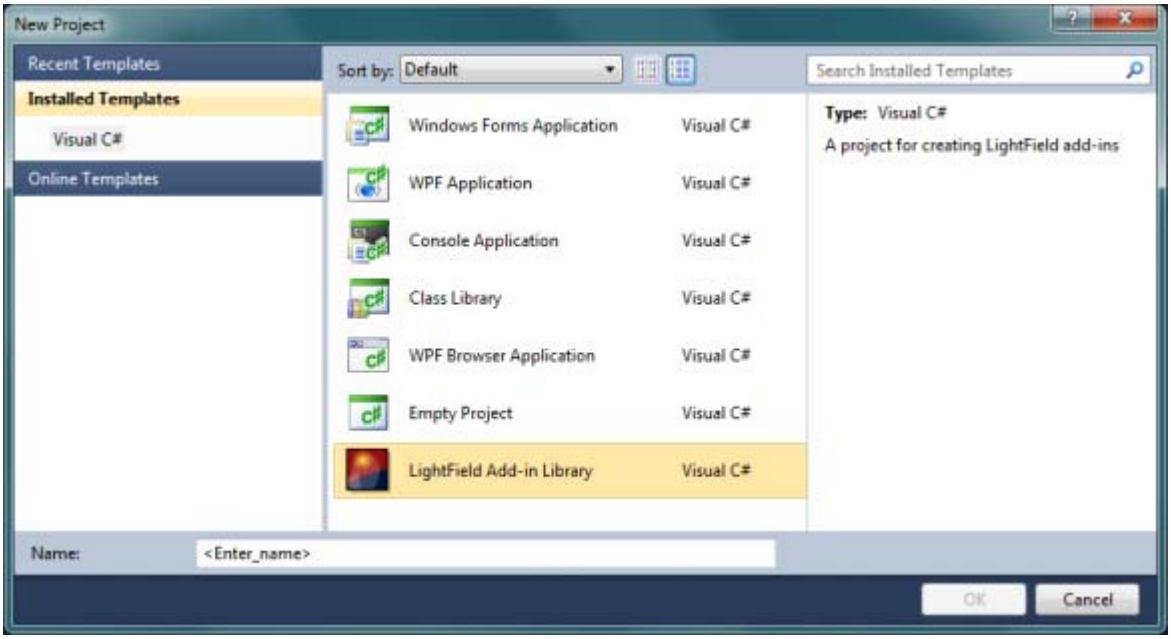
Figure 5-1: Typical Visual Studio New Project Dialog: Installed Visual C# Templates



4411-0135_0014

3. From within the list of **Installed Templates**, highlight the **LightField Add-in Library** template. See Figure 5-2.

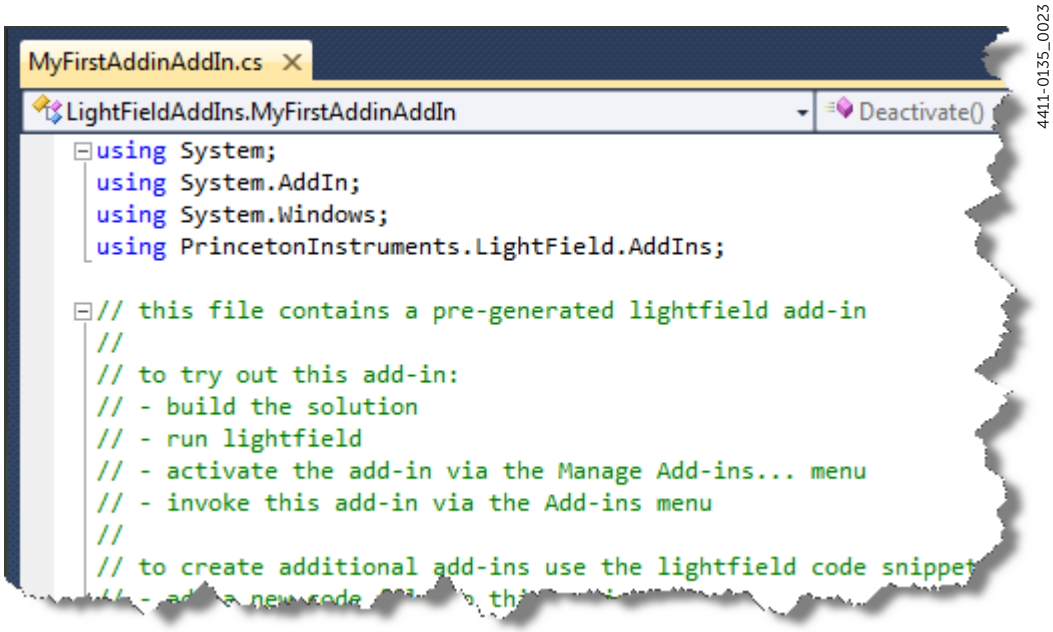
Figure 5-2: Visual Studio New Projects Dialog: LightField Add-in Library Template



4. Within the **Name:** field, enter the name of the new project (e.g., MyFirstAddin,) and click **OK**.

   Visual Studio will then create the new project based on the **LightField Add-in Library** template. When done, VS will open the add-in project for editing and customization. See Figure 5-3.

Figure 5-3: Typical Project Add-in Source File: LightField Add-In Template (Partial View)



5.2 Build and Test the New Add-in

Once the new add-in has been created and opened, before beginning any additional development work, it is a good idea to verify that the project can be compiled and run within LightField.

Perform the following procedure to build and test a new add-in project:

1. If desired, within Visual Studio, locate and customize the following information that is displayed within the LightField Manage Add-ins dialog:
   - The name of the add-in;
   - Description;
     This text is displayed when hovering the mouse over the name of the add-in.
   - Publisher Information.
     This text is displayed to the right of the add-in's name.

   Refer to Code Example 5-1.

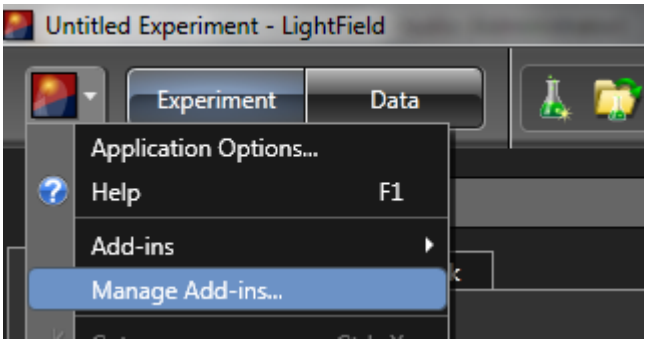
Code Example 5-1: Code Excerpt from Add-In Source File

```
…

namespace LightFieldAddIns
{
   // TODO: edit the add-in description and publisher information
   [AddIn(
      "MyFirstAddin",
      Description = "Description Placeholder",
      Version = "1.0.0",
      Publisher = "Publisher Placeholder")]
 …
```

2. If changes have been made, save the add-in source file using one of the following methods:
   - Select **File ► Save** `project_name.cs` from the VS menu bar;
   - Click the  icon.
3. From the Visual Studio menu bar select **Build ► Build Solution** to create the Add-in.
   Upon successful completion of the add-in's build, the message **Build Succeeded** will be displayed in the Visual Studio status bar.
4. Launch LightField.

5. From the **Application** menu, select **Manage Add-ins...**. See Figure 5-4.

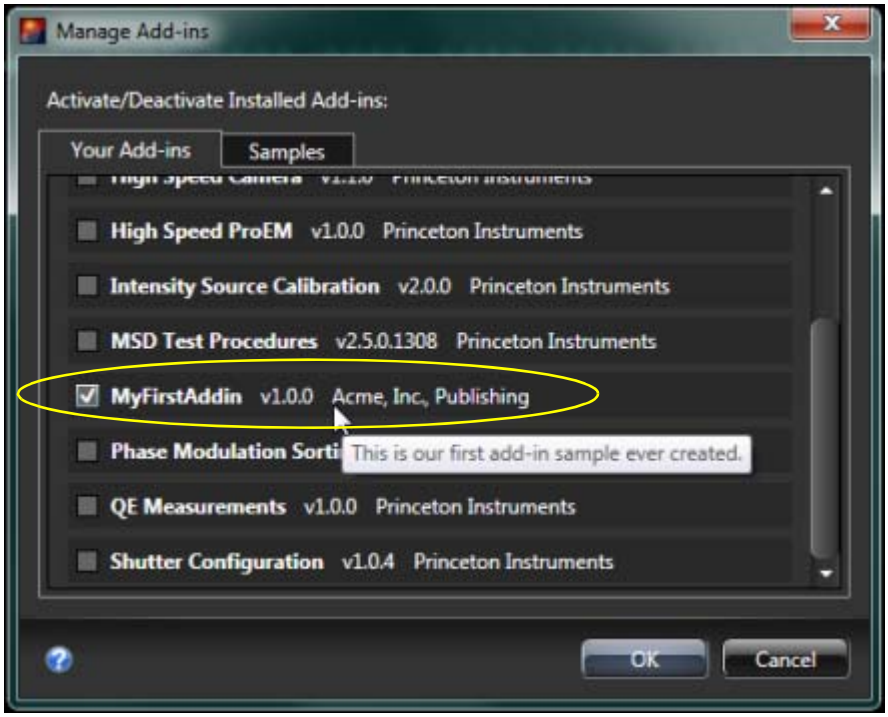
Figure 5-4: Typical LightField *Manage Add-ins...* Menu Option



4411-0135_0015

6. Within the **Your Add-ins** tab, scroll to locate the add-in just created, and check its box to activate it. See Figure 5-5.

Figure 5-5: Typical Manage Add-ins Dialog

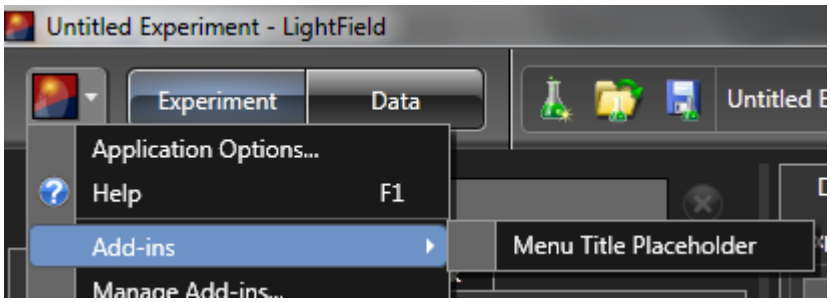


4411-0135_0016

7. Click **OK** to activate the add-in and dismiss the **Manage Add-ins...** dialog.

8. From the **Application** menu, select **Add-ins ► Menu Title Placeholder**.

Figure 5-6: Typical LightField Menu: Add-ins Menu



4411-0135_0017

Figure 5-7 illustrates the output of this test add-in. Click **OK** to dismiss the pop-up.

Figure 5-7: Output of LightField Add-In

4411-0135_0018

5.3 Customize Add-in Functionality

At this point, the basic functionality of the add-in has been verified. Additional customization of the add-in can now begin.

Sections of the source code tagged with a **TODO** comment should be customized by the add-in developer to provide required functionality and/or details.

REFERENCES:

Sample add-ins can be found in the following directory:

- `My Documents\Princeton Instruments\LightField\AddIn SDK\`

The following documentation is included and is titled:

- LightField LightField® Add-ins and Automation Programming Manual.pdf

This page is intentionally blank.

Chapter 6: Create Add-ins from Scratch

This chapter provides detailed information about building custom add-ins and automation modules without using pre-formatted templates.

CAUTION!

The procedures within this chapter are recommended for advanced users only.

LightField add-ins and automation modules are developed using the same developer interface. Custom code should be usable in both automation as well as Add-in contexts.

NOTE:

Verify that the required development tools and software applications have been installed per Section 4.1.2, Developer Tools Required when Creating an Add-in from Scratch.

6.1 Building and Add-in

Perform the following procedure to create/build a new LightField Add-in:

1. Create a new Class Library Project.
2. Include the following references in the project:
   - `System.AddIn;`
   - `LIGHTFIELD_ROOT\PrincetonInstruments.LightFieldAddInSupportServices.dll (Copy Local = true);`
   - `LIGHTFIELD_ROOT\AddInViews\PrincetonInstruments.LightFieldViewV4.dll (Copy local = false).`
   - `PresentationCore`
   - `PresentationFramework`
   - `WindowsBase`
   - `System.Xaml`
   - `System.Drawing`

NOTE:

`Copy local` settings can be changed by right-clicking on the reference to open its properties.

3. Derive a class from both `AddInBase` and `ILightFieldAddIn`.
   Configure the appropriate attributes.

```
[AddIn("MyAddinTitle",
Version = "1.0.0",
Publisher = "Teledyne Princeton Instruments")]
```

4. Determine in which zones the Add-in is to be displayed within LightField by returning the proper `UISupport` flags.

REFERENCES:

Refer to Chapter 2, Introduction to LightField Add-Ins, for detailed information about Add-in zones.

5. Add the necessary code for the zone(s) that are being supported.
   If all properties and methods for the selected zone(s) are not supported, the application will provide exceptions indicating which items are missing.
6. Create a new directory for the Add-in within the following directory:

   `LIGHTFIELD_ROOT\Addins\[your_name]`

   Copy or build the Add-in DLL into this location.
7. Launch LightField and navigate to the Add-in Manager via the **Application Menu** and the **Manage Add-ins...** selection.
   If everything was done correctly, the new Add-in should be listed.

NOTE:

Add-ins are listed alphabetically based on their name.

Once the Add-in is visible to LightField, custom code can be added to the Add-in to perform the desired task.

6.2 Building Automation Modules

Automation for LightField requires Visual Studio 2010 and the .Net 4.0 Framework. Interfaces for automation are identical to those in the Add-in interface.

NOTE:

All samples have been built using Visual C#.

Perform the following procedure to build a new automation module:

1. Create a new Class Library Project.
2. Include the following references in the project:
   - `System.AddIn`
   - `LIGHTFIELD_ROOT\PrincetonInstruments.LightFieldAddInSupportServices.dll (Copy Local = true)`
   - `LIGHTFIELD_ROOT\AddInViews\PrincetonInstruments.LightFieldViewV4.dll (Copy local = true)`
   - `LIGHTFIELD_ROOT\PrincetonInstruments.LightField.AutomationV4.dll (Copy Local = true)`
3. Create the automation object.

   The object reference should be held for the life of the application. Code Example 6-1 illustrates basic automation code which:
   - Creates an object with the `visible flag = true`;

     When `visible flag = false`, LightField will be run in invisible mode.
   - Illustrates that `ILightFieldApplication` can be obtained from an automation object. This is identical to the interface passed into the Add-ins.

REFERENCES:

Refer to Chapter 8, Automation Class, for additional information about automation class functionality.

Code Example 6-1: Basic Automation Code

```
public class MyApplication_
{
   // Backing for Automation_
   PrincetonInstruments.LightField.Automation.Automation automation_;

   //Backing For Application
   ILightFieldApplication application_;

   //Constructor
   MyApplication()
   {
      //Create a new automation object (visible = true)
      automation_ = new PrincetonInstruments.LightField.Automation.
         Automation(true);

      //Get the application with all of the usable interfaces on it
      application_ = automation_.LightFieldApplication;
   }
}
```

6.3 ILightField Experiment Settings.chm

For detailed information about setting names, units, etc., refer to the `LightField Experiment Settings.chm` help file located in the `Program Files\Princeton Instruments\LightField` directory. This is the same directory in which the `PrincetonInstruments.LightField.exe` file is stored.

Chapter 7: Programming Interface

This chapter provides programming information necessary to create custom LightField add-ins and automation modules.

7.1 Microsoft Managed Add-In Framework

LightField supports user-authored content via Microsoft's Add-in Framework. Using this framework, the author of an add-in includes a reference to the `LightFieldAddInView` module and then links to the appropriate parts of the user interface. Once this link is established, the author can then further customize the add-in.

Each add-in must be a class inherited from `ILightFieldAddIn` so it can be properly loaded by LightField. It must also be derived from `AddinBase` which is included in the `LightFieldAddinSupportServices` module.

7.2 Required (Core) Methods

Methods are blocks of codes in which executable statements are stored.

All LightField add-ins are required to include the following methods:

- `Activate();`
- `Deactivate()`.

This section provides programming information about these required methods.

7.2.1 `Activate()`

Description

`Activate()` is called by LightField when an add-in is activated within LightField's Add-in Manager.

This method provides the add-in with access to the LightField application and is where basic initialization should occur.

Access

`Activate()` must be publicly accessible.

Syntax

The syntax for `Activate()` is:

```
public void Activate(ILightFieldApplication app)
```

Input Parameters

Input parameters for `Activate()` are:

`app`: The LightField application.

Output Parameters

There are no output parameters associated with `Activate()`

7.2.2 `Deactivate()`

Description

`Deactivate()` is called when the add-in is deactivated using LightField's Add-in Manager.

It allows the add-in to clean up and release any resources before being deactivated.

Access

`Deactivate()` must be publicly accessible.

Syntax

The syntax for `Deactivate()` is:

```
public void Deactivate()
```

Input Parameters

There are no input parameters associated with `Deactivate()`.

Output Parameters

There are no output parameters associated with `Deactivate()`.

7.3 Required Properties

All LightField add-ins are required to include the following properties:

- `UISupport`.

This section provides programming information about these required properties.

7.3.1 `UISupport`

Description

`UISupport` is initially read by LightField when the add-in is loaded. This provides LightField with the following information:

- In which zone(s) the add-in is to be placed;
- Which GUI objects and/or views are to be created.

Access

`UISupport` must be publicly accessible.

Syntax

The syntax for `UISupport` is:

```
public UISupport UISupport
{
    get { return UISupport.ExperimentSetting; }
}
```

Enumeration Flags

`UISupport` uses enumeration flags to specify in which zone(s), if any, the add-in is to be placed.

The enumeration flag definition is:

```
[Flags]
public enum UISupport
{
    None = 0,
    Menu = 1,
    ApplicationToolBar = 2,
    DataToolBar = 4,
    ExperimentSetting = 8,
    ExperimentView = 16,
}
```

Each enumeration value can be combined one or more additional values with a logic OR so that an Add-in can be placed within all five zones if desired.

REFERENCES:

For additional information about zones, refer to Section 2.2, LightField Add-in Zones.

7.4 ILightFieldAddIn

ILightFieldAddIn is the interface between a custom Add-in and LightField. Using this interface an add-in specifies:

- Where it is to be placed within the LightField user interface;
- How it wants to look;
- What tool tips it provides;
- The handlers for the buttons, check boxes, and menu items.

In order for an add-in to support the five zones described in Section 2.2, LightField Add-in Zones, the UISupport enumeration flags must be used when the Add-in property UISupport is queried.

REFERENCES:

Refer to Section 7.3.1, UISupport, for additional information.

7.4.1 Supported Methods

Refer to Table 7-1 for information about methods supported by ILightFieldAddIn.

Table 7-1: ILightFieldAddIn Supported Methods (Sheet 1 of 2)

Return	Method	Parameters	Notes
void	Activate (ILightFieldApplicationhostObj)	ILightFieldApplication The ILightFieldApplication parameter is passed into the Add-in and is the top level interface which contains properties to retrieve all of the lower level interfaces and features provided by the host application.	Initializes the Add-in and gets the required interfaces from the hostObj. This is the only chance and mechanism available to either cache the individual interfaces you need or the entire ILghtFieldApplication object. ***NOTE:*** You will need it or a subset of it to do anything meaningful in the future lifetime of your Add-in.
void	Deactivate()	None	This is equivalent to a destructor or a disposal mechanism. It is basically telling you that the application is closing and you should clean up whatever objects, open files or memory buffers you possess that will not be cleaned up normally.

Table 7-1: ILightFieldAddIn Supported Methods (Sheet 2 of 2)

Return	Method	Parameters	Notes
`void`	`RequestHelp(int helpTopicID)`	`int` Describes the help topic id that is requesting help.	When the application is queried for help about a particular Add-in's **`UI`** Element, it will forward the call to the AddIn with the proper `helpTopicID` that was provided by the property.
`void`	`UIApplicationToolBarExecute()`	None	When the `ApplicationToolBar` Element that the Add-in is supporting is clicked from within the application, this method will be called. It is up to the author to implement any desired effect.
`void`	`UIDataToolBarExecute()`	None	When the `DataToolBar` Element that the Add-in is supporting is clicked from within the application, this method will be called. It is up to the author to implement any desired effect.
`void`	`UIMenuExecute()`	None	When the `Menu` Element that the Add-in is supporting is clicked from within the application, this method is called. It is up to the author to implement any desired effect.

7.4.2 Supported Properties

Refer to Table 7-2 for information about properties supported by `ILightFieldAddIn`.

Table 7-2: ILightFieldAddIn Supported Properties (Sheet 1 of 2)

Name	Type	Accessors	Notes
`UIApplicationToolBarBitmap`	`Bitmap`	get	The Add-in can return a Bitmap to be displayed as the button on the Application toolbar. Null = no Bitmap displayed.
`UIApplicationToolBarHelpTopicID`	`Nullable<int>`	get	Gets the help topic id for the `ApplicationToolBar` zone if it is supported and exists.
`UIApplicationToolBarIsChecked`	`Nullable<bool>`	get, set	If this is null, the toolbar item is a simple button. If it is not, this property is called when the user changes state on the checkbox in LightField. Otherwise, the set is called when the button is pressed.
`UIApplicationToolBarIsEnabled`	`bool`	get	Allows the author of the Add-in the ability to disable the button/ checkbox from being clicked. If this returns false, the button/checkbox item is grayed out.
`UIApplicationToolBarTitle`	`string`	get	Title shown in the application toolbar if the `ApplicationToolBar` bit is set in the `UISupport` property.
`UIApplicationToolBarToolTip`	`string`	get	Contextual tool tip for this Add-in when the user mouses over the button or checkbox assigned to this zone.
`UIDataToolBarBitmap`	`Bitmap`	get	The Add-in can return a Bitmap to be displayed as the button on the Data toolbar. Null = no Bitmap displayed.
`UIDataToolBarHelpTopicID`	`Nullable<int>`	get	Gets the help topic id for the `DataToolBar` zone if it is supported and exists.
`UIDataToolBarIsChecked`	`Nullable<bool>`	get, set	If this is null, the toolbar item is a simple button; if it is not, this property is called when the user changes state on the checkbox in LightField. Otherwise, the set is called when the button is pressed.
`UIDataToolBarIsEnabled`	`bool`	get	Gives the developer the ability to disable the button/checkbox from being clicked, if this returns false the button/checkbox item is grayed out.

Table 7-2: ILightFieldAddIn Supported Properties (Sheet 2 of 2)

Name	Type	Accessors	Notes
`UIDataToolBarTitle`	`string`	get	Title shown in the data toolbar if the `DataToolBar` bit is set in the `UISupport` property.
`UIDataToolBarToolTip`	`string`	get	Contextual tool tip for this Add-in when the user mouses over the button or checkbox assigned to this zone.
`UIExperimentSettingHelpTopicID`	`Nullable<int>`	get	Gets the help topic id for the experiment setting zone if it is supported and exists.
`UIExperimentSettingTitle`	`string`	get	Returns the title of the window for the experiment setting type. Only valid if the `ExperimentSetting(0x08)` bit is set.
`UIExperimentViewHelpTopicID`	`Nullable<int>`	get	Returns `UIExperimentViewHelpTopicID`, null if it is not used.
`UIExperimentViewTitle`	`string`	get	Returns title of the `UIExperimentView.` The title will be shown on the tab in the view section of LightField for this Add-in.
`UIMenuIsChecked`	`Nullable<bool>`	get, set	If this is null, the menu item is a simple button; if it is non-null, it is a checkbox and the state is true or false. This is required if you return the `Menu (0x01)` bit of the `UISupport` property.
`UIMenuIsEnabled`	`bool`	get	Gives the author of the Add-in the ability to disable the menu from being clicked: if this returns false, the menu item is grayed out.
`UIMenuTitle`	`string`	get	Title shown in the menu this field is required if you return the `Menu(0x01)` bit of the `UISupport` property.
`UISupport`	`UISupport`	get	Returns the support flags telling the application which zones the Add-in supports within the UI framework.

7.4.3 Code Example

Code Example 7-1 is a LightField Add-in for plotting data in graphical format.

Code Example 7-1: Plot Sample Add-in

```
[AddIn("Plot Sample",
Version = "1.0.0",
Publisher = "Teledyne Princeton Instruments",
Description="This sample Add-in demonstrates making data in memory and then
   plotting it as graphs. It also demonstrates adding multiple graph sources to
   the same window.
public class AddinPlotSample : AddinBase, ILightFieldAddIn
{
    bool? toolBarState_;

    ////////////////////////////////////////////////////////////////
    public UISupport UISupport {get {return UISupport.ApplicationToolBar;}}
    ////////////////////////////////////////////////////////////////
    public void Activate(IlightFieldApplication app)
    {
        // Capture Interface
        LightFieldApplication = app;
        toolBarState_ = null;
    }
    ////////////////////////////////////////////////////////////////
    public void Deactivate() { }
    ////////////////////////////////////////////////////////////////
    public override string UIApplicationToolBarTitle{get{return "Plot Sample";}}
    ////////////////////////////////////////////////////////////////
    public override bool? UIApplicationToolBarIsChecked
    {
        get {return toolBarState_;}
        set {toolBarState_ = value;}
    }
    ////////////////////////////////////////////////////////////////
    public override void UIApplicationToolBarExectue()
    {
        PlotSinAndCos();
    }
    ////////////////////////////////////////////////////////////////
    public override bool UIApplicationToolBarIsEnabled{get{return true;}}
    ////////////////////////////////////////////////////////////////
    public override string UIApplicationToolBarToolTip
    {
        get {return "Generate Cos & Sin Curves and display results in four
               windows.";}
    }
```

Code Continued on Next Page

Code Example 7-1: Plot Sample Add-in (continued)

```
40
41      ////////////////////////////////////////////////////////////
42      public override int? UIApplicationToolBarHelpTopicID{ get {return 1;}}
43      ////////////////////////////////////////////////////////////
44      public override Bitmap UIApplicationToolBarBitmap
45      {
46          get
47          {
48              Assembly executingAssembly =   Assembly.GetExecutingAssembly();
49              Stream resourceStream =        executingAssembly.GetManifestResource
                                               Stream ("LightFieldAddInSamples.
                                               Plot_Sample.PlotSample.bmp");
50              Bitmap image =                 new Bitmap(resourceStream);
51              return image;
52          }
53  }
```

7.4.3.1 Code Description

The coding structure illustrated in Code Example 7-1 is typical for all user-defined Add-ins.

NOTE:

In general, when a specific flag is used (e.g., the `ApplicationToolBar` flag for `UISupport`,) then all properties which include the flag should be supported. It is possible for an Add-in to support all of the areas and any that do that would also end up supporting all of the properties in the Table 7-2.

This section describes Code Example 7-1, line-by-line, to illustrate where everything goes and to define the interaction between the Add-in and the application.

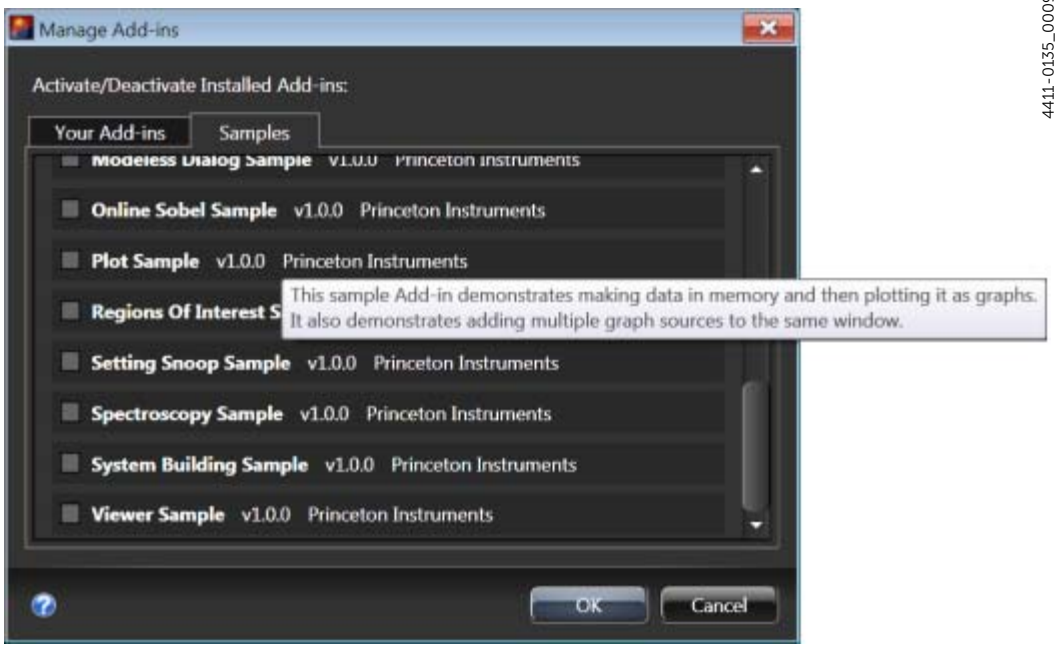
- Lines 1 through 4

  This section of code describes how the Add-in looks to the `AddInManager` in LightField.

  The Name, Version, Publisher and Description are all shown in LightField. This sample shows as Plot Sample, Version 1.0.0, with the Publisher's name in the Manage Add-ins dialog. The Description is shown as the tool tip

Figure 7-1 illustrates how this code is rendered on-screen.

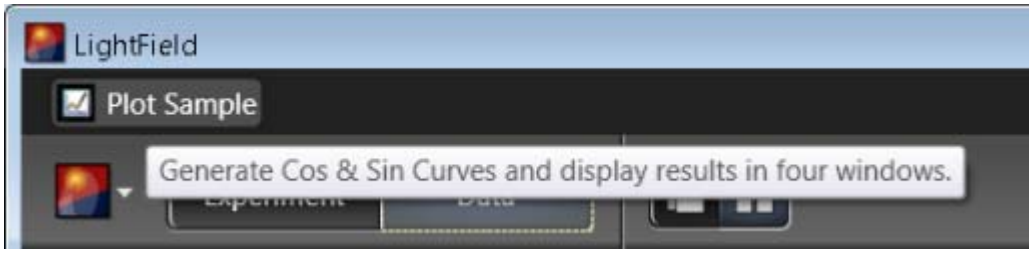
Figure 7-1: Plot Sample Add-in Listed in the Manage Add-in Dialog



4411-0135_0009

- Line 5
  Defines the class note as derived from `AddinBase` and `ILightFieldAddIn`. This is a requirement of all Add-ins in order to assure proper and complete functionality.
- Line 7
  Defines a nullable Boolean type used for the state of the toolbar.
- Line 10
  Supports the `UISupport` property and tells the application that this Add-in is supported on the UI Application toolbar. Figure 7-2 illustrates how this code is rendered on-screen as the Plot Sample button on the application toolbar.

Figure 7-2: Plot Sample Add-in in the Application Toolbar



4411-0135_0010

- Line 12
  Shows the activation call. The nullable Boolean is set to null and cache the `ILightFieldApplication` interface for later use.
- Line 19
  Shows the deactivate method which, in this case, does not need to do anything special.
- Line 21
  Shows the label that is to be displayed on the button.

- Lines 23 through 27

  `UIApplicationToolbarIsChecked` since the backing field is null in this case. When the application initially queries the Add-in, it will make the button a pressable button instead of a check box, which would be the case if the backing field here was initialized to true or false.

  If `toolBarState_` was initially set to true or false then this would be a check box button with the initial state set to match `toolBarState_`. When the state changed, the set property would be called. Since `toolBarState_` is initialized to null, this guarantees that the application will never call the set property.
- Line 29 through 32

  This is the execute command. This is called whenever a user pushes the **Plot Sample** button.

  NOTE:

  This command is not called when `toolBarState_` is either `True` or `False`. When `toolBarState_` is `True` or `False`, the `Set` would be used when the button is pushed rather than executing a command.
- Line 34

  This property determines if the button is enabled/disabled. In this sample, it is always enabled.
- Line 36 through 39

  This property is used to pick up the tool tip for the actual button as shown in Figure 7-2.
- Line 44 through 53

  This block of code creates a bitmap for the button and puts it in front of the text. Looking at the screen capture above you see a bitmap of a graph on the button. The base class will return a null by default and if this property is not supported, the button will be a normal button with no bitmap.

7.5 ILightFieldApplication

This is the application interface. From this, the Add-in can access interfaces to:

- DataManager;
- DisplayManager;
- Experiment;
- FileManager.

This is the primary interface for the application's exposed feature set. This interface is passed in to the Activate() method.

7.5.1 Supported Properties

Refer to Table 7-3 for information about properties supported by ILightFieldApplication.

Table 7-3: ILightFieldApplication Supported Properties

Name	Type	Accessor	Notes
StatusManager	IApplicationStatusManager	get	Returns the manager of the status for the main application. This controls the busy, slightly busy, and idle cursor.
DataManager	IDataManager	get	Returns the data manager whose only function is allowing the programmer to create image data sets from memory buffers.
DisplayManager	IDisplay	get	Returns the display manager interface which contains functions to get views of data and manipulate the views of the data.
Experiment	IExperiment	get	Returns the experiment interface, which is capable of changing current device and system settings as well as acquiring data and registering for system events.
FileManager	IFileManager	get	Returns the file manager which allows the end user to Open, Create, Read, Write and Copy Files.

7.6 IUserInteractionManager

This is the interface that controls the currently displayed cursor and user interaction during program flow in LightField. Using this interface enables add-in programmers to set the busy state if their add-ins are performing lengthy tasks. This interface also allows programmers to halt program flow until the user responds to a prompt.

7.6.1 Supported Properties

Refer to Table 7-4 for information about properties supported by `IUserInteractionManager`.

Table 7-4: IUserInteractionManager Supported Properties

Name	Type	Accessor	Notes

`ApplicationBusyStatus`	`ApplicationBusyStatus`	get, set	Gets or sets the application's cursor state to one of the following states defined in this enumeration:
			`enum ApplicationBusyStatus`
			`{`
			`NotBusy,`
			`SlightlyBusy,`
			`Busy`
			`}`

`SuppressUserInteraction`	`bool`	get, set	Setting this flag to true will choose the default action of all prompts invoked by the application. Setting it to false will mean program flow in the add-in will be halted until the prompt is manually removed by the user.

7.7 IImageData

This is the lowest level view of a slice of data. It represents a single region of interest and a single frame of data. This is the basic building block for the `FileManager` and `DataManager` objects. There is no constructor for `IImageData`. The only way to get an interface of this type is to pull it from the more complex `IImageDataSet` object. However, using this interface is the only way to modify the application's back end image data.

7.7.1 Supported Methods

Refer to Table 7-5 for information about methods supported by `IImageData`.

Table 7-5: IImageData Supported Methods

Return	Method	Parameters	Notes
`System.Array`	`GetData()`	None	Returns a one-dimensional array that holds Width * Height pixel elements with each element having a size determined by the `Format` property.
`void`	`SetData(System.Array array)`	`System.Array array` is a block of data to set to the object.	Copies the user's created array of the proper size back into the Application side object.

7.7.2 Supported Properties

Refer to Table 7-6 for information about properties supported by `IImageData`.

Table 7-6: IImage Data Supported Properties

Name	Type	Accessor	Notes

`Format`	`ImageDataFormat`	get	Returns the type of the data present in the blocks. The types supported are defined in the enumeration below.
			`public enum ImageDataFormat`
			`{`
			`MonochromeUnsigned16,`
			`MonochromeUnsigned32,`
			`MonochromeFloating32`
			`}`

`Height`	`int`	get	This is the height in pixel elements of this block of data.
`Width`	`int`	get	This is the width in pixel elements of this block of data.

7.8 IImageDataSet

This interface describes a set of data. It can be multi-frame and contain multiple regions of interest or even a LightField file object.

7.8.1 Supported Methods

Refer to Table 7-7 for information about methods supported by `IImageDataSet`.

Table 7-7: IImageDataSet Supported Methods

Return	Method	Parameters	Notes
`IImageData`	`GetColumn`	`int regionIndex, long frameIndex, int colIndex`	Gets a section of data from region and frame at the `colIndex` specified, this is a single column of data.
`IImageData`	`GetFrame`	`int regionIndex, long frameIndex`	Gets a section of data from the region and frame index specified. This is an X by Y section of data.
`IImageData`	`GetPixel`	`int regionIndex, long frameIndex, int rowIndex, int colIndex`	Gets a pixel from the region, frame, row and column specified.
`IImageData`	`GetRow`	`int regionIndex, long frameIndex, int rowIndex`	Gets a section of data from region and frame data at the specified `rowIndex`, this is a single row of data.
`Metadata`	`GetFrameMetadata`	`long frameIndex`	Gets the specified Metadata object for this frame of the current data set.

7.8.2 Supported Properties

Refer to Table 7-8 for information about properties supported by `IImageDataSet`.

Table 7-8: IImageDataSet Supported Properties (Sheet 1 of 2)

Name	Type	Accessor	Notes

`FileAccess`	`System.Nullable<System.IO.FileAccess>`	get	Returns the access of the image data set contained within. If the access does not support writing, then even if you get the underlying `IImageData`, you will not be able to change the contents of the data.
			`public enum FileAccess`
			`{`
			`  Read,`
			`  Write,`
			`  ReadWrite`
			`}`

`FilePath`	`string`	get	If the underlying data belongs to a file you can get the path of the file back from this interface via this property.

Table 7-8: IImageDataSet Supported Properties (Sheet 2 of 2)

Name	Type	Accessor	Notes
`Frames`	`long`	get	Returns the number of frames held by this image data set.
`Regions`	`RegionOfInterest[]`	get	Returns the regions of interest that this image data set contains.
`TimeStampOrigin`	`DateTimeOffset?`	get	Returns the initial time stamp on this ImageDataSet.

7.8.3 Metadata Structure

The `IImageDataSet`'s `GetFrameMetadata`() method returns a metadata structure.

The fields will be null if there is no metadata turned on at the time of acquisition; otherwise, data will be the appropriate type listed in Table 7-9.

Table 7-9: IImageDataSet Metadata Structure Fields

Field	Description
`TimeSpan? ExposureStarted`	Per frame time stamp of when the exposure was started.
`TimeSpan? ExposureEnded`	Per frame stamp of when the exposure was completed.
`long? FrameTrackingNumber`	Frame ID counts up per frame.
`double? GateTrackingWidth`	If intensified gated camera is used the width of gate per frame.
`double? GateTrackingDelay`	If an intensified camera the gating delay of the current frame.
`double? ModulationTrackingPhase`	If a modulating camera is used the phase of the modulation can be tracked per frame.

7.8.4 Code Example

Code Example 7-2 is sample code that illustrates how to acquire metadata structure information.

Code Example 7-2: IImageDataSet Code Example

```
1  //Turning on time stamping, prior to acquiring your data
2  experiment.SetValue(CameraSettings.AcquisitionTimeStampingStamps,
                       TimeStamps.ExposureEnded | TimeStamps.ExposureStarted);
3
4  // Turn On Frame Tracking
5  experiment.SetValue(CameraSettings.AcquisitionFrameTrackingEnabled, true);
6
7  // Turning On Gate Tracking
8  experiment.SetValue(CameraSettings.AcquisitionGateTrackingEnabled, true);
9
10 // Get the image dataset
11 IImageDataSet dataSet = fileManager.OpenFile(@"C:\Demo\Data.spe",
      FileAccess.Read);
12
13 / Get the meta data for each frame
14 for (int i = 0; i < dataSet.Frames; i++)
15 {
16     Metadata metaData = dataSet.GetFrameMetaData(i);
17
18     // Fields are null if the data does not contain them
19     if (metaData.ExposureEnded.HasValue && metaData.ExposureStarted.HasValue)
20     {
21         TimeSpan duration = metaData.ExposureEnded.Value -
                  metaData.ExposureStarted.Value;
22     }
23 }
```

7.9 IDataManager

This interface allows the author to create image data sets from memory buffers. There are two construction methods:

- One for a simple single region;
- One for creating a more complex data set with multiple regions.

7.9.1 Supported Methods

Refer to Table 7-10 for information about methods supported by `IDataManager`.

Table 7-10: IDataManager Supported Methods

Return	Method	Parameters	Notes
`IImageDataSet`	`CreateImageDataSet`	`System.Array Data, RegionOfInterest roi, ImageDataFormat Format`	Creates an `IImageDataSet` object for a single region and a given format. To access the data, the user must use one of the `IImageDataSet` methods and then use the data access methods within the `IImageData` interface to manipulate the data if desired after construction.
`IImageDataSet`	`CreateImageDataSet`	`IList<System.Array> arrays, RegionOfInterest[ ] roi, ImageDataFormat Format`	This performs the same task as the previous `CreateImageDataSet` method except it allows multiple regions and multiple buffers to be linked together in a construction call.

7.9.2 Code Example

Code Example 7-3 is sample code that:

- Creates an array of data;
- Gets the `IDataManager` interface;
- Creates and displays the newly constructed `IImageDataSet` object.

Code Example 7-3: Data Sample Add-In

```
1  ////////////////////////////////////////////////////////////////
2  // Make a linear ramp of data and display it.
3  ////////////////////////////////////////////////////////////////
4  private void GenerateRamps()
5  {
6      // Make a Frame 200x400 pixels
7      ushort[] frame1 = new ushort[200 * 400];
8      for (int pix = 0; pix < 200 * 400; pix++)
9          frame1[pix] = (ushort)(pix % 400);
10
11     // Get the addin file manager
12     var datamgr = LightFieldApplication.DataManager;
13     if (datamgr != null)
14     {
15         RegionOfInterest roi = new RegionOfInterest(0, 0, 400, 200, 1, 1);
16
17         IImageDataSet imageData = datamgr.CreateImageDataSet(frame1, roi,
                ImageDatFormat.MonochromeUnsigned16);
18         IDisplay display = LightFieldApplication.DisplayManager;
19         if (display != null)
20         {
21             // Select Data File Compare Mode & 3 Vertical Windows
22             display.ShowDisplay(DisplayLocation.DataComparisonWorkspace,
                       DisplayLayout.ThreeVertical);
23             IDisplayViewer view = null;
24
25             // Put the data in all three windows
26             for (int i = 0; i <= 2; i++)
27             {
28                 view = display.GetDisplay(DisplayLocation.DataComparisonWorkspace,
                           i);
29                 view.Display("Ramp", imageData);
30             }
31         }
32     }
33 }
```

7.10 IFileManager

This interface supports working with LightField data-type files. This interface can be used to handle any file access within an add-in.

7.10.1 Supported Methods

Refer to Table 7-11 for information about methods supported by `IFileManager`.

Table 7-11: IFileManager Supported Methods (Sheet 1 of 3)

Return	Method	Parameters	Notes
`void`	`CancelExport Asyn`	None	Cancels either of the `ExportAsync` methods and raises the completed event with a canceled flag.
`void`	`CloseFile`	`IImageDataSet file`	Closes the file represented by this `IImageDataSet`.
`IImageDataSet`	`CopyFile`	`IImageDataSet toBeCloned, string fileNameTo`	Makes a duplicate of a file and returns the duplicate. This makes a complete copy of the `IImageDataSet`. Modifying the copy does not affect the original data.
`IImageDataSet`	`CopyFile`	`string fileNameFrom, string fileNameTo`	Makes a duplicate of a file and returns the duplicate. This makes a complete copy of the `IImageDataSet`. Modifying the copy does not affect the original data.
`IExportSettings`	`CreateExportSettings`	`ExportFileType typeOfFile`	Returns the object that houses all of the settings for an export. This must be created with a particular type. The type is immutable: once set, the object cannot be changed. If a different export type is needed then another object of that type must be created. `public enum ExportFileType { Fits, Spc, Tiff, Csv, Avi }`
`IImageDataSet`	`CreateFile`	`string fileName, RegionOfInterest[] regions, long frames, ImageDataFormat format`	Creates an empty file. The data must be written frame by frame and region by region using the `IImageData` interface method `SetData`.

Table 7-11: IFileManager Supported Methods (Sheet 2 of 3)

Return	Method	Parameters	Notes
`IList<IExportError>`	`Export`	`IExportSettings exportSettings, IList<string> filesOrFolders`	Settings to use for the export and a list of files/directories to export. This will perform the export and also block until the export is completed.
`IList<IExportError>`	`Export`	`IExportSettings exportSettings, String file`	Settings to use for the export and a single file to export. This will perform the export and also block until the export is completed.
`void`	`ExportAsync`	`IExportSettings exportSettings, IList<string> filesOrFolders`	Settings to use for the export and a list of files/directories to export. This will start the export and return immediately. The export will signal a completed event.
`void`	`ExportAsync`	`IExportSettings exportSettings, String file`	Settings to use for the export and a single file to export. This will start the export and return immediately. The export will signal a completed event.
`IList<IImageDataSet>`	`GetDataWorkspaceFiles`	None	Returns a list of all the files that are currently open in the data workspace view.
`IList<IImageDataSet>`	`GetOpenFiles`	None	Returns a list of `IImageDataSets` that represent all of the files currently open in LightField's data viewer.
`IList<string>`	`GetRecentlyAcquiredFileNames`	None	Returns a list of strings that include the full file names of all recently acquired data. Basically, it is the list shown in the recently acquired pop-up window.
`string`	`GetXml`	`string filename`	Gets the XML document pertaining to the file pointed to by the string.
`string`	`GetXml`	`IImageDataSet fileData`	Gets the XML document pertaining to the file pointed to by the `IImageDataSet` object.
`IImageDataSet`	`OpenFile`	`string fileName, FileAccess access`	Opens a file with the specified access, where the filename is the full name including path. It returns an `IImageDataSet` which can be used to get the data in the file by region and frame number.

Table 7-11: IFileManager Supported Methods (Sheet 3 of 3)

Return	Method	Parameters	Notes
`bool`	`SaveFile`	`IImageDataSet dataSet, string fileName`	Saves the dataset to the filename specified.
`void`	`SetXml`	`IImageDataSet fileData, string xmlString`	Sets the XML on the file pointed to by the `IImageDataSet`. ***NOTE:*** You cannot do this on a file by its path. This is intentional and requires you to copy a file somewhere else before changing its XML data.

7.10.2 Supported Events

Refer to Table 7-12 for information about events supported by `IFileManager`.

Table 7-12: IFileManager Supported Events

Event Name	Raised Upon
`ExportCompleted`	The devices in the device tray changed, or someone plugged in or removed a device. In the arguments are a canceled flag and a list of `IExportErrors` interfaces which can be used to determine any problems with a batch export.

7.10.3 Code Sample

Code Example 7-4 creates a file, populates it with two frames of data, and closes the file.

Code Example 7-4: Sample Add-In: File Access

```
// Create a file with 2 frames and the proper width and height
// The file is initially created as empty and the user must load data into it.

RegionOfInterest roi = new RegionOfInterest(0, 0, w1[0], h1[0], 1, 1);
RegionOfInterest[] rois = { roi };
IImageDataSet TwoFrameOneRoi = filemgr.CreateFile(path + "\\TwoFrameOneRoi.spe",
       rois,
              2,// Frames
              ImageDataFormat.MonochromeUnsigned16);

// Put Data to frame 1
IImageData data1 = TwoFrameOneRoi.GetFrame(0, 0);
data1.SetData(frame1);

// Put Data to frame 2
IImageData data2 = TwoFrameOneRoi.GetFrame(0, 1);
data2.SetData(frame2);

// Finally close this file
filemgr.CloseFile(TwoFrameOneRoi);
```

7.11 IExperiment

This is the interface to the hardware devices in the system as well as some application settings. Via this interface you can do things like set up regions of interest, set exposure time, set the resultant file name, acquire data and much more.

7.11.1 Supported Methods

Refer to Table 7-13 for information about methods supported by `IExperiment`.

Table 7-13: IExperiment Supported Methods (Sheet 1 of 3)

Return	Method	Parameters	Notes
`void`	`Acquire`	None	Identical to pressing the Acquire button from within the LightField application.
`void`	`Add`	`IDevice device`	Adds a device to the current experiment that comes from the available device list.
`void`	`Clear`	None	Clears the Experiment of all Devices.
`IImageData Set`	`Capture`	`int frames`	Captures a number of frames.
`bool`	`Exists`	`string settingName`	This method will see if the `settingName` is actually a real setting. Some settings are contextual or hardware-dependent.
`void`	`FilterSettingChanged`	`IList<string> settings`	This method takes a list of setting names that can be monitored for changes. If any of the setting names in the list value change, it will notify anyone listening to the `SettingChanged` event with the name of the setting and the new value.
`IList <object>`	`GetCurrentCapabilities`	`string settingName`	Gets the settings current capabilities as a list of objects, the caller must know the type expected and then cast it for the selected setting
`ISettingRange`	`GetCurrentRange`	`string settingName`	Gets the setting range for the selected setting. This reflects the current range with all other settings taken into account.
`IList <object>`	`GetMaximumCapabilities`	`string settingName`	Gets the settings maximum capabilities as a list of objects, the caller must know the type expected and then cast it for the selected setting
`ISettingRange`	`GetMaximumRange`	`string settingName`	Gets the setting's maximum possible values for the setting selected. These values are the maximum extrema ever possible with the current system.

Table 7-13: IExperiment Supported Methods (Sheet 2 of 3)

Return	Method	Parameters	Notes
`IList<string>`	`GetSavedExperiments`	None	Returns the list of experiment names that are shown in the box when you bring up the dialog to restore experiments.
`object`	`GetValue`	`string settingName`	Gets a value by its setting name. Setting names are defined in the `LightField AddinSupportServices` module.
`bool`	`IsReadOnly`	`string setting`	This method is used to determine if a setting is read only or not.
`bool`	`IsRelevant`	`string settingName`	This method will determine if a setting's current value will have any impact on the system. If a setting is not relevant, then changing it will have no effect on current system.
`bool`	`IsValid`	`string settingName, object value`	This is a method used to check if a value is viable in the current system configuration. You can use this to pretest a value before setting it.
`bool`	`Load`	`string experimentName`	Loads a LightField experiment. It should be a name that is in the list of saved experiments. True will be returned if the load was successful; otherwise, False will be returned.
`void`	`Preview`	None	Identical to pressing the Preview (Run) button from within the application.
`void`	`Remove`	`IDevice device`	Removes a device from the current device list. This will remove the device from the ExperimentDevices property and put it back onto the AvailableDevices list.
`void`	`RestoreToDefault()`	None	Restores the experiment to its default state: the same state as if the user had just dragged the camera into the Experiment Devices area.
`void`	`Save`	None	Saves the current experiment as it is. This means if your Add-in has changed any settings, those changes will now be saved in the experiment file and restored upon restoration.
`void`	`SaveAs`	None	Saves the current experiment as a new name.
`void`	`SetBinnedSensor Region`	`int binnedX, int binnedY`	Sets the camera into full frame mode with binning factors passed in. Subsequent acquisitions will return full chip data with the applied binning passed in.

Table 7-13: IExperiment Supported Methods (Sheet 3 of 3)

Return	Method	Parameters	Notes
`void`	`SetCustomRegions`	`RegionOfInterest[] regions`	This sets the custom regions with the set of regions passed into it. It also forces the current region mode into custom; meaning if you acquire, then the expected data is your regions passed in.
`void`	`SetFullSensorRegion`	None	Sets the camera into full frame mode with no binning. Subsequent acquisitions will return full chip data.
`void`	`SetLineSensorRegion`	`int centerRowsBinned`	Sets the camera into a mode where it bins N center rows, dictated by the parameter passed in.
`void`	`SetValue`	`string settingName, object value`	Sets a value by its setting name. Setting names are defined in the `LightField AddinSupportServices` module. The architecture will try to convert the value to the proper value of the setting. For example if your value is a string "5.65" and the underlying setting is expecting a double, the conversion will be transparent to the programmer.
`void`	`Stop`	None	Performs the same as if the user presses the Stop button from within the application.
`void`	`TakeOneLook`	None	Takes a quick 1 frame of data. This is the same as clicking the one look button.

7.11.2 Supported Properties

Refer to Table 7-14 for information about properties supported by `IExperiment`.

Table 7-14: IExperiment Supported Properties

Name	Type	Accessors	Notes
`AvailableDevices`	`IList<IDevice>`	get	Returns a list of system devices shown in the device tray, these are the currently available for adding to the system.
`BinnedSensorRegion`	`RegionOfInterest`	get	Returns the binned mode region of interest.
`CustomRegions`	`RegionOfInterest[]`	get	Returns the current regions of interest stored as custom regions.
`ExperimentDevices`	`IList<IDevice>`	get	This returns a list of devices that are currently being used by the current experiment.
`FullSensorRegion`	`RegionOfInterest`	get	Returns the full sensor mode region of interest.
`IsReadyToRun`	`bool`	get	Returns the flag dictating if the system is in a state ready to acquire data. A value of true means that the acquisition functions can be called.
`IsRunning`	`bool`	get	If there is an acquisition running, then this will return true; otherwise, it will return false.
`IsUpdating`	`bool`	get	If the system is being built or manipulated in the User interface, returns true; otherwise, returns false.
`LineSensorRegion`	`RegionOfInterest`	get	Returns the line sensor mode region of interest.
`Name`	`string`	get	Returns the name of the experiment currently loaded in LightField.
`Progress`	`double`	get	If there is an acquisition running, it will return the current progress of the acquisition.
`SelectedRegions`	`RegionOfInterest[]`	get	Returns the current region(s) of interest that will be returned from the hardware if acquire was called next.
`SystemColumnCalibration`	`double[]`	get	Returns the current wavelength calibration from the Spectrometer/Camera Pair and the current center wavelength.
`SystemColumnCalibrationErrors`	`double[]`	get	If there is a current fixed calibration in place, this method will return the per pixel errors of that calibration.
`SystemIntensityCalibration`	`float[]`	get	If there is an intensity calibration in place, this will return the calibration coefficients that are being used by the system.

7.11.3 Supported Events

Refer to Table 7-15 for information about events supported by `IExperiment`.

Table 7-15: IExperiment Supported Events

Event Name	Raised Upon
`AvailableDevicesChanged`	The devices in the device tray changed or someone plugged in or removed a device.
`ExperimentCompleted`	When an acquisition is completed, this event will be raised.
`ExperimentDevicesChanged`	The experiment devices changed: someone dragged a camera, spectrometer, light source, or filter wheel into or out of the system.
`ExperimentStarted`	You begin an acquisition, either from pressing Start or from calling Preview/Acquire or Capture.
`ExperimentUpdated`	Experiment updating completed. The experiment was in flux but when this event is received it is finalized.
`ExperimentUpdating`	Experiment is being manipulated so Add-ins must be cautious when this event is received. They should not try to talk to system components until the ExperimentUpdated is received.
`ImageDataSetReceived`	Called whenever image data frame(s) are received. For example, in Preview mode each frame will signal an event that data was received.
`IsReadyToRunChanged`	Tells the Add-in author the current state (if they can run or not) in the arguments delivered with the event.
`SettingChanged`	Whenever a setting in the `filteredsettings` list value is changed, this event will be raised.

7.11.4 Code Example

Code Example 7-5 performs the following tasks:

- Gets an experiment object;
- Sets custom values (e.g., exposure, resultant file name;)
- Acquires data;
- Connects up and listens for the `ExperimentCompleted` event.

Code Example 7-5: Custom Acquire Sample Add-in

```
////////////////////////////////////////////////////////////////
// Override some typical settings and acquire an spe file with a
// specific name.
////////////////////////////////////////////////////////////////
private void control__Click(object sender, RoutedEventArgs e)
{
    // Get the experiment object
    IExperiment experiment = LightFieldApplication.Experiment;
    if (experiment != null)
    {
        // Integration to 100 millisecongs
        // Not All Systems Have an Exposure Setting
        if (experiment.Exists(CameraSettings.ShutterTimingExposureTime))
        experiment.SsetValue(CameraSettings.ShutterTimingExposureTime, 100.0);

        // Don't Attach Date/Time
        experiment.SetValue(ExperimentSettings.ExampleFileAttachDate, false);
        experiment.SetValue(ExperimentSettings.ExampleFileAttachTime, false);

        // Save file as Specific.Spe to the default directory
        experiment.SetValue(ExperimentSettings.ExampleFileBaseFileName,
               "Specific");

        // Connect the event handler
        acquireCompletedEventHandler_ = new EventHandler
               <ExperimentCompletedEventArgs>(exp_AcquisitionComplete);
        experiment.ExperimentCompleted += acquireCompletedEventHandler);

        // Begin the acquisition
        experiment.Acquire();
    }
}
////////////////////////////////////////////////////////////////
// Acquire Completed Handler
// This just fires a message saying that the data is acquired.
////////////////////////////////////////////////////////////////
void exp_AcquisitionComplete(object sender, ExperimentCompletedEventArgs e)
{
    ((IExperiment)sender).ExperimentComplete -= acquireCompletedEventHandler_;
    MessageBox.Show("Acquire Completed");
}
```

7.11.5 `Pulse` Structure

The `Pulse` structure is used with the experiment getvalue/setvalue methods and is only valid when used with the following `CameraSettingNames`:

- `GatingDifEndingGate`
- `GatingDifStartingGate`
- `GatingRepetitiveGate`
- `GatingSequentialEndingGate`
- `GatingSequentialStartingGate`
- `HardwareIOAuxOutputGate`

7.11.5.1 Field Information

Table 7-16 provides `Pulse` Structure field information.

Table 7-16: `Pulse` Structure Field Information

Field	Description
`double Width`	Width of the gated pulse, in nanoseconds.
`double Delay`	Delay of the gated pulse, in nanoseconds.

7.11.5.2 Code Example

Code Example 7-6 is sample code that illustrates how to acquire metadata structure information.

Code Example 7-6: `Pulse` Structure Code Example

```
1 // Create a pulse (width,delay) in nanoseconds
2 Pulse setPulse = new Pulse(500, 1000);
3
4 // Setting the pulse
5 experiment.SetValue(CameraSettings.GatingRepetitiveGate,
     setPulse);
6
7 // Getting the pulse
8 Pulse getPulse =
     (Pulse)experiment.GetValue(CameraSettings.GatingRepetitive
     Gate);
```

7.11.6 `RegionOfInterest` Structure

The `RegionOfInterest` structure encapsulates a single area of the sensor over which image data will be acquired. The following methods use this structure or an array of these structures:

```
Experiment {…

   // Get and set the custom regions of interest
   void SetCustomRegions(RegionOfInterest[] regions);
   RegionOfInterest[] CustomRegions { get; }

   // Returns the single region of interest for the different modes
   RegionOfInterest FullSensorRegion { get; }
   RegionOfInterest BinnedSensorRegion { get; }
   RegionOfInterest LineSensorRegion { get; }

   // Returns the current regions being used for acquires
   RegionOfInterest[] SelectedRegions { get; }
}

ImageDataSet {…

   RegionOfInterest[] Regions { get; }

}
```

7.11.6.1 Field Information

Table 7-17 provides `RegionOfInterest` Structure field information.

Table 7-17: `RegionOfInterest` Structure Field Information

Field	Description
`int X`	Origin of the x coordinate for the region
`int Y`	Origin of the y coordinate for the region
`int Width`	Width of the region in sensor pixels to acquire
`int Height`	Height of the region in sensor pixels to acquire
`int XBinning`	Binning in the X direction
`int YBinning`	Binning in the Y direction

7.11.6.2 Code Example

Code Example 7-7 is sample code that illustrates how to define a `RegionOfInterest` structure.

Code Example 7-7: `RegionOfInterest` Code Example

```
// Create 1 region
RegionOfInterest roi1
    = new RegionOfInterest(0, 0, 200, 200, 1, 1);

// Create a second region
RegionOfInterest roi2
    = new RegionOfInterest(400, 400, 300, 400, 1, 1);

// Create an array with both regions in it
RegionOfInterest[] rois = new RegionOfInterest[] { roi1, roi2 };

// Send the regions to the experiment
experiment.SetCustomRegions(rois);
```

7.11.7 `Modulation` Structure

The `Modulation` structure can be used with the experiment `setvalue/getvalue` function where the name of the setting name is:

`CameraSettings.IntensifierModulationCustomSequence`

The camera must support RF modulation in order to apply custom modulation to it.

The `setvalue/getvalue` methods take or return an array of these structures that define the entire custom modulation sequence.

7.11.7.1 Field Information

Table 7-18 provides `Modulation` Structure field information.

Table 7-18: `Modulation` Structure Field Information

Field	Description
`double Duration`	The duration of the modulated signal, in milliseconds.
`double Phase`	The phase of the modulated signal, in degrees.
`double Frequency`	The frequency of the modulated signal, in MHz.
`double OutputFrequency`	The frequency of the user's RF output, in MHz. The output frequency is only active if the modulation output signal is enabled.

7.12 IDevice

This interface is used with the Experiment interface in determining which devices are in the system or are available to be in the system.

7.12.1 Supported Properties

Refer to Table 7-19 for information about properties supported by `IDevice`.

Table 7-19: IDevice Supported Properties

Name	Type	Accessors	Notes
`Interface`	`string`	get, set	Gets or sets the string representation of the interface of the devices.
`Model`	`string`	get	Gets the model name/number of the device
`SerialNumber`	`string`	get	Gets the serial number of the device.

`Type`	`DeviceType`	get, set	Gets or sets the type of device.
			`public enum DeviceType`
			`{`
			`  Camera,`
			`  Spectrometer,`
			`  FilterWheel,`
			`  LightSource`
			`}`
			This is an enumeration as to what type of device this device is.

7.13 ISettingRange

This interface is used with the Experiment interface in determining which devices are in the system or available to be in the system.

7.13.1 Supported Properties

Refer to Table 7-20 for information about properties supported by `ISettingRange`.

Table 7-20: ISettingRange Supported Properties

Name	Type	Accessors	Notes
`Increment`	`double`	get	The increment value acceptable for this setting range which is associated to the setting name that was used in getting the range.
`Maximum`	`double`	get	The largest value acceptable for this setting range which is associated to the setting name that was used in getting the range.
`Minimum`	`double`	get	The smallest value acceptable for this setting range which is associated to the setting name that was used in getting the range.

7.14 IDisplay

The display interface is a very small interface that allows the user to access the more useful `IDisplayViewer` interface. This interface allows the Add-in author to set the current view mode from within the Add-in, especially useful in setting up comparison views.

7.14.1 Supported Methods

Refer to Table 7-21 for information about methods supported by `IDisplay`.

Table 7-21: IDisplay Supported Methods (Sheet 1 of 2)

Return	Method	Parameters	Notes
`IDisplaySource`	`Create`	`string fileName`	Opens a file into a view and overwrites anything currently in that view.
`IDisplaySource`	`Create`	`string sourceName, IImageDataSet imageDataSet`	Loads a data set with a source name and blow away anything else currently in that view.
`IDisplayViewer Control`	`CreateDisplayViewerControl`	`none`	Creates a new `DisplayViewerControl` so the programmer can use the LightField style data view from within the context of the Add-in
`IDisplayViewer`	`DisplayFileInDataWorkspace`	`String fileName`	Opens a file from disk, displays it in the data workspace view, and returns the viewer for the file.
`IDisplayViewer`	`GetDisplay`	`DisplayLocationlocation, int index`	From the selected location, gets the `IDisplayViewer` by its index. `public enum DisplayLocation { DataWorkspace, DataComparisonWorkspace, ExperimentWorkspace }`

Table 7-21: IDisplay Supported Methods (Sheet 2 of 2)

Return	Method	Parameters	Notes

`void`	`ShowDisplay`	`DisplayLocation location, DisplayLayout layout`	Sets the current display location and layout. Calling this will be the same as if you selected the display location and layout from within the application.
			`public enum DisplayLocation { DataWorkspace, DataComparisonWorkspace, ExperimentWorkspace }`
			The layout matches from left to right with the application toolbar shown in Figure 7-3
			`public enum DisplayLayout { One, TwoHorizontal, TwoVertical, ThreeTopFavored, ThreeVertical, FourEven, FourTopFavored, FourVertical, FourLeftFavored, FiveTopFavored, FiveLeftFavored }`

Figure 7-3: Application Toolbar: View Layout

0011

7.15 IDisplaySource

7.15.1 Supported Methods

Refer to Table 7-22 for information about methods supported by `IDisplaySource`.

Table 7-22: IDisplaySource Supported Methods

Name	Type	Accessors	Notes
`Dispose`	`void`	None	When you create a display source of your own using the create methods, it is the creator's responsibility to call the dispose on it and allow the application to properly clean up.

7.16 IDisplayViewer

This interface is the programmer's access to anything view-related. Everything you can do from within the application to alter the view of the data is possible to do via programming as well. There are a few exceptions, however (e.g., scroll position.)

7.16.1 Supported Methods

Refer to Table 7-23 for information about methods supported by `IDisplayViewer`.

Table 7-23: IDisplayViewer Supported Methods (Sheet 1 of 2)

Return	Method	Parameters	Notes
`IDisplaySource`	`Add`	`IDisplaySource newSource`	Adds a display source to this view.
`void`	`AutoScaleIntensity`	None	Autoscales the intensity of the image.
`void`	`Clear`	None	Clears everything in the view.
`void`	`CopyAsTextToClipboard`	None	Copies the image to the clipboard as text.
`IDisplaySource`	`Display`	`string fileName`	This is a helper function to load a file straight into this view and return `IDisplaySource` interface on it. This source is added to collection.
`IDisplaySource`	`Display`	`string sourceName, IImageDataSet imageDataSet`	This is a helper function to load a dataset with a given dataset name right into this view. This source is added to the collection.
`IDisplaySource`	`Display`	`IDisplaySource newSource`	Adds this source to the collection of sources.
`void`	`OptimizeContrast`	None	Tries to optimize the contrast of the image so that the most dynamic range is visible.

Table 7-23: IDisplayViewer Supported Methods (Sheet 2 of 2)

Return	Method	Parameters	Notes
`void`	`RemoveAt`	`int sourceIndex`	Removes a display source from this view at the specified index.
`IDisplaySource`	`ReplaceAt`	`IDisplaySource newSource, int sourceIndex`	Replaces a display source at the **`sourceIndex`** in this view with the one passed in.
`void`	`SetIntensityLevels`	`double blackLevel, double whiteLevel`	Effectively, just groups the two properties `IntensityBlackLevel` and `IntensityWhiteLevel` and sets them both in the same function call.
`void`	`SetXAxisVisibleRange`	`double start, double end`	Sets the x-axis visible range in start and end.
`void`	`SetYAxisVisibleRange`	`int index, double start, double end`	Index is source index and start and end are the limits you want to set the range to.
`void`	`ZoomIn`	None	Analogous to pressing the zoom in button.
`void`	`ZoomOut`	None	Analogous to pressing the zoom out button.
`void`	`ZoomToActualSize`	None	Zooms the data in this view to its size in pixel 1:1 with on-screen pixels.
`void`	`ZoomToBestFit`	None	Zooms the data in this view to its optimal fit.
`void`	`ZoomXAxisToBestFit`	None	Auto zooms along the X-axis for best fitting display.
`void`	`ZoomYAxisToBestFit`	None	Auto zooms along the Y-axis for best fitting display.

7.16.2 Supported Properties

Refer to Table 7-24 for information about properties supported by `IDisplayViewer`.

Table 7-24: IDisplayViewer Supported Properties (Sheet 1 of 5)

Name	Type	Accessors	Notes

`ActualDisplayType`	`DisplayType`	get	There are times when the actual display type could be different from the **`DisplayType`**. This value will be one of the following:
			`public enum DisplayType`
			`{`
			`Auto,`
			`Image,`
			`Graph`
			`}`

`AlwaysAutoScaleIntensity`	`bool`	get, set	Get/Set flag to always auto scale.

`CrossSections`	`DisplayCrossSections`	get, set	Get/Set the cross sections displayed on the image from the enumeration.
			`public enum DisplayCrossSections`
			`{`
			`None,`
			`XAxis,`
			`YAxis,`
			`XYAxes,`
			`ZAxis`
			`}`

`CursorIntensity`	`Nullable<double>`	get	Gets the intensity of the current cursor position.
`CursorPosition`	`Nullable<System.Windows.Point>`	get, set	Sets the current cursor position on this viewer.

`CursorStyle`	`DataCursorStyle`	get, set	Get/Set the current cursor style from the enumeration.
			`public enum DataCursorStyle`
			`{`
			`SmallCross,`
			`LargeCross`
			`}`

`DataPointsVisible`	`bool`	get, set	Flag to show individual points on the graph.
`DataSelection`	`System.Windows.Rect`	get, set	Get/Set the selection rectangle on this viewer as if the user had selected the red rectangle with the mouse.

Table 7-24: IDisplayViewer Supported Properties (Sheet 2 of 5)

Name	Type	Accessors	Notes

`DisplayLocation`	`DisplayLocation`	get	From within the viewer, gets the location in which it exists.
			`public enum DisplayLocation`
			`{`
			`DataWorkspace,`
			`DataComparisonWorkspace,`
			`ExperimentWorkspace`
			`}`

`DisplaySourceIndex`	`int`	get, set	Get/Set the currently selected display source index. Viewers can have up to five display sources but only if they are in `DisplayType.Graph`
`DisplaySources`	`IList<IDisplaySource>`	get	Gets the display source(s) contained in this viewer.

`DisplayType`	`DisplayType`	get, set	Get/Set the display type from the enumeration.
			public enum DisplayType
			`{`
			`Auto,`
			`Image,`
			`Graph`
			`}`

`FrameIndex`	`long`	get, set	Get/Set the frame index, for multi-frame images. This is equivalent to the frame slider on the bottom of the image/graph viewer.
`FrameIndexSelectionStart`	`System.Nullable<long>`	get, set	Allows you set a frame selection start point on a multi-frame image.
`FrameIndexSelectionEnd`	`System.Nullable<long>`	get, set	Allows you set a frame selection ending point on a multi-frame image.
`Frames`	`long`	get	Gets the current number of frames for the **`DisplaySourceIndex`** selected.

`GraphGridLines`	`GraphGridLines`	get, set	Enumeration that controls if any or which grid lines are shown on the plot's view.
			`public enum GraphGridLines`
			`{`
			`Both,`
			`XAxisOnly,`
			`YAxisOnly,`
			`None,`
			`}`

Table 7-24: IDisplayViewer Supported Properties (Sheet 3 of 5)

Name	Type	Accessors	Notes
GraphZoomMode	GraphZoomMode	get, set	Get/Set the enumeration on allowable zoom mode. `public enum GraphZoomMode { Normal, XAxisOnly, YAxisOnly }`
HasSource	bool	get	If this view has any associated image source than this property will be True.
IntensityBlackLevel	System.Nullable<double>	get, set	Get/Set current black level. Anything below this will be clipped by the display to black.
IntensityWhiteLevel	System.Nullable<double>	get, set	Get/Set current white level. Anything above this will be clipped by the display to white.
IsLiveSource	bool	get	If the source associated with this view is a live data stream, then this will return a True.
IsPlaybackRunning	bool	get, set	Get/Set the playback flag. This will play a multi-frame sequence in order.
LiveDisplaySource	IDisplaySource	get	Gets the current live display source
MaximumZoom	double	get	Gets the maximum zoom the display can support.
MinimumZoom	double	get	Gets the minimum zoom the display can support.
PeaksIndicated	PeaksIndicated	get, set	Get/Set the peak indicators for this viewer from the enumeration. `public enum PeaksIndicated { None, Sharp, Intermediate, Broad }`
PlaybackFrameRate	double	get, set	Get/Set the desired playback frame rate, for playing in slow motion or fast motion if desired.
PlotsStacked	bool	get, set	Get/Set Flag to show the plots either on the same axis (false) or stacked so that each plot gets its own axis (true). It only has any relevant behavior if the ActualDisplayType property is set to Graph.

Table 7-24: IDisplayViewer Supported Properties (Sheet 4 of 5)

Name	Type	Accessors	Notes
PlottingStyle	PlottingStyle	get, set	Get/Set the current plotting style for this viewer from the enumeration. `public enum PlottingStyle { None, Line, Point }`
PseudoColorPalette	PseudoColorPalette	get, set	Get/Set flag to determine if there is a pseudo color palette in use for over saturation. `public enum PseudoColorPalette { UnitializedEum, None, Exposure, BlueRed, Red, Yellow, Green, Cyan, Blue }`
RegionIndex	int	get, set	Get/Set the current region index. If a viewer has multiple regions, changing this with programming is the same as selecting the region in the drop down list on the viewer.
Regions	int	get	Gets the current number of regions this viewer has associated with it.
RepeatPlayback	bool	get, set	The repeat loop flag that works in tandem with the IsPlaybackRunning flag.
RowIndex	int	get, set	Get/Set the current row on the DisplaySourceIndex, RegionIndex selected. This is the same as picking the row from the drop down list.
Rows	int	get	Gets the current number of rows. This viewer has associated with its current RegionIndex.
ShowExposureEndedTimeStamp	bool	get, set	Get/Set flag that shows ending time stamp in this viewer.
ShowExposureStartedTimeStamp	bool	get, set	Get/Set flag that shows starting time stamp in this viewer.
ShowFrameTrackingNumber	bool	get, set	Get/Set flag that shows frame tracking in this viewer.

Table 7-24: IDisplayViewer Supported Properties (Sheet 5 of 5)

Name	Type	Accessors	Notes
ShowGateTrackingDelay	bool	get, set	Get/Set the flag that shows gate tracking delay metadata on the display.
ShowGateTrackingWidth	bool	get, set	Get/Set the flag that shows gate tracking width metadata on the display.
ShowModulationTrackingPhase	bool	get, set	Enables/disables if Modulation Phase tracking is shown in the current viewer.
ShowStampedExposureDuration	bool	get, set	Get/Set flag that shows exposure duration in this viewer.
ShowTimeStampsInAbsoluteTime	bool	get, set	Get/Set flag that determines how time stamping information is displayed.
TimeStampOffset	double	get, set	Get/Set the time stamp offset.
XAxisCalibrationDisabled	bool	get, set	Get/Set the flag that allows you to show in calibration space. If it is false, data is presented in pixels.

XAxisCalibration	CalibrationCategory	get, set	This gets/sets the type of calibration on the X Axis since there can be more than one type associated with data but only one is displayed at a time.
			`enum CalibrationCategory`
			`{`
			`None,`
			`Wavelength,`
			`ExposureStartedTimeStamp,`
			`ExposureEndedTimeStamp,`
			`GateTrackingDelay,`
			`GateTrackingWidth,`
			`FrameTrackingNumber`
			`ModulationTrackingPhase`
			`}`

ZAxisCrossSectionXAxisCalibrationDisabled	bool	get, set	Get/Set flag that allows calibrations in the Z Axis (Frames) dimension to be shown.

ZAxisCrossSectionXAxisCalibration	CalibrationCategory	get, set	This get/sets the type of calibration on the Z Axis since there can be more than one type associated with data but only one displayed at a time.
			`enum CalibrationCategory`
			`{`
			`None,`
			`Wavelength,`
			`ExposureStartedTimeStamp,`
			`ExposureEndedTimeStamp,`
			`GateTrackingDelay,`
			`GateTrackingWidth,`
			`FrameTrackingNumber`
			`}`

Zoom	double	get, set	Get/Set the zoom level of this viewer.

7.16.3 Code Example

Code Example 7-8 generates some curves and plots them in a viewer. This code is from the Plot Sample example.

Code Example 7-8: Plot Sample Add-in

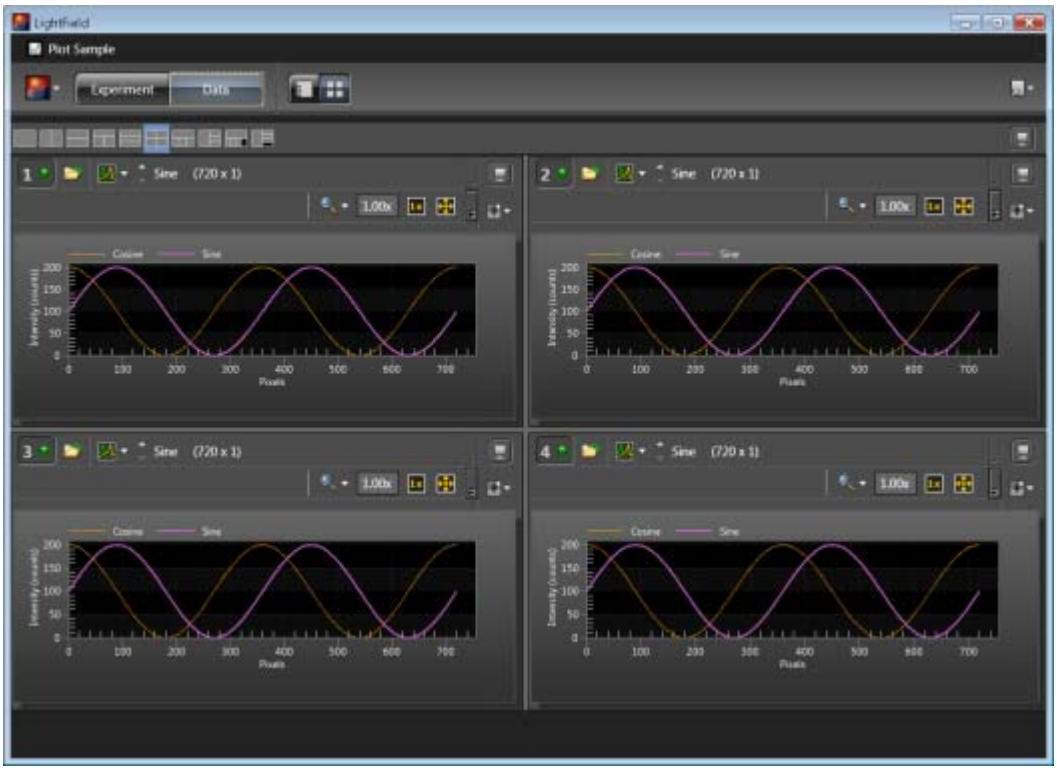
```
// Make two curves (720 = 2*PI so it is a full cycle)
ushort[] cosine = new ushort[720]
ushort[] sine = new ushort[720];

// Generate Curves (Amplitude 100)
for (int pix = 0; pix < 720; pix++)
{
    // Convert To Angle
    double angle = Math/PI * ((double)pix - 360) / 180.0;

    // Compute Points
    cosine[pix] = (ushort)((double)100 * (Math.Cos(angle) + (double)1));
    sin[pix] = (ushort)((double)100 * (Math.Sin(angle) + (double)1));
}

// Get the data manager
var datamgr = LightFieldApplication.DataManager;
if (datamgr != null)
{
    RegionOfInterest roi = new RegionOfInterest(0,0,720,1,1,1);
    // Create Blobs
    IImageDataSet cosData = datamgr.CreateImageDataSet(cosine, roi,
         ImageDataFormat.MonochromeUnsigned16);
    IImageDataSet sineData = datamgr.CreateImageDataSet(sine, roi,
         ImageDataFormat.MonochromeUnsigned16);

    // Get the Display Object
    IDisplay display = LightFieldApplication.DisplayManager;
    if (display != null)
    {
        // Select Data File Compare Mode & 4 Even Windows
        display.ShowDisplay(DisplayLocation.DataComparison
               Workspace, DisplayLayout.FourEven);
        IDisplayViewer view = null;

        // Put the data in all 4 windows
        for (int i = 0; i <3; i++)
        {
            view = display.GetDisplay(DisplayLocation.DataComparison
                    Workspace, i);
            view.Display("Cosine", cosData);
            IDisplaySource sinSource = view.Create("Sine",sineData);
            view.Add(sinSource);
        }
    }
}
```

7.16.3.1 Code Description

This section provides a line-by-line description of Code Example 7-8. Figure 7-4 illustrates the plots that result after executing this code in LightField.

- Lines 2 through 16
  Build up two arrays: one for sine, and one for cosine.
- Line 18
  Gets the data manager object.
- Lines 21 through 25
  Create `IImageDataSets` from the arrays of data.
- Line 27
  Gets the display manager.
- Line 31
  Sets the display to `DataComparisonMode` with four evenly segmented views.
- Line 37
  Gets the view from the chosen display index and mode.
- Line 38
  Displays the cosine `IImageDataSet` with the name cosine.
- Line 39
  Creates on this view a new `DisplaySource` from the sine `IImageDataSet`.
- Line 40
  Adds the sine display source to the view.

Figure 7-4: Four Plots Generated by Code Example 7-8



0012

7.17 IDisplayViewerControl

This class represents the display viewer control. The LightField interface now allows programmers to put LightField views of data into their own window.

7.17.1 Supported Properties

Refer to Table 7-25 for information about properties supported by IDisplayViewerControl.

Table 7-25: IDisplayViewerControl Supported Properties

Name	Type	Accessors	Notes
Control	INativeHandle	get	Gets the handle of this control.
DisplayViewer	IDisplayViewer	get	Gets the display viewer object associated with this control.
IsDisplayTypeSelectable	bool	get, set	Flag that allows the user to tell the display if the user interface for this display is allowed to select the display type or not. For example if the programmer always wants a graph display they can set this to false and force the display type to be unchangeable at the UI level.
IsLiveDataAvailable	bool	get, set	Allows the user to change whether or not this display is allowed to show live data.
SourceAvailability	ImageDataSourceAvailability	get, set	Determines if this display view is showing the selector to choose a source. `public enum ImageDataSourceAvailability { Selectable, Visible, Collapsed }`

7.18 IExportError

7.18.1 Supported Properties

Refer to Table 7-26 for information about properties supported by IExportError.

Table 7-26: IExportError Supported Properties

Name	Type	Accessors	Notes
Exception	Exception	get	Returns a native exception type which contains information about the nature of the problem.
InputPath	string	get	Returns the input path of the file that had a problem.
OutputPath	string	get	Returns the output path of the file that had a problem.

7.19 IExportSelectionError

7.19.1 Supported Properties

Refer to Table 7-27 for information about properties supported by IExportSelectionError.

Table 7-27: IExportSelectionError Properties

Name	Type	Accessors	Notes

Errors	IList<ExportItemSelectionError>	get	Returns a list of selection errors which is a list of enums of type:
			`public enum ExportItemSelectionError`
			`{`
			`LegacyDataFile,`
			`OpenedFileSelected,`
			`NoDataFilesInFolder,`
			`NotEnoughFrames,`
			`NotEnoughRegions,`
			`FileNotFound,`
			`FolderNotFound`
			`}`

InputPath	string	get	Returns the input path of the file that had a problem.

7.20 IExportSettings

7.20.1 Supported Methods

Refer to Table 7-28 for information about methods supported by `IExportSettings`.

Table 7-28: IExportSettings Supported Methods

Return	Method	Parameters	Notes
`void`	`SetFrameRange`	`long? mainframe, long? maxFrame`	Sets a frame range max and min if you wish to only export a specified range of frames from within a file or files.
`IList<IExportSelectionError>`	`Validate`	`IList<string> filesOrFolders`	Gives the user the chance to validate the selection of files/folders and returns errors before even trying to do the export.

7.20.2 Supported Properties

Refer to Table 7-29 for information about properties supported by `IExportSettings`

Table 7-29: IExportSettings Supported Properties (Sheet 1 of 3)

Name	Type	Accessors	Notes
`CustomOutputPath`	`string`	get, set	When `CustomPath` is the selected option for `OutputPathOption`, this is used to set the destination path for the exported file(s).

`ExportFileType`	`ExportFileType`	get	Returns the file type the settings were created with.
			`public enum ExportFileType`
			`{`
			`  Fits,`
			`  Spc,`
			`  Tiff,`
			`  Csv,`
			`  Avi`
			`}`

`ExposureEndedPrecision`	`int`	get, set	Gets or sets the precision for exposure ended.

`ExposureEndedUnit`	`TimeUnit`	get, set	Gets or sets the unit for the exposure ended time.
			`public enum TimeUnit`
			`{`
			`  Picoseconds,`
			`  Nanoseconds,`
			`  Microseconds,`
			`  Milliseconds,`
			`  Seconds,`
			`  Minutes,`
			`  Hours`
			`}`

Table 7-29: IExportSettings Supported Properties (Sheet 2 of 3)

Name	Type	Accessors	Notes
ExposureStarted Precision	int	get, set	Gets or sets the precision for Exposure started.

ExposureStartedUnit	TimeUnit	get, set	Gets or sets the unit for the exposure started signal.
			`public enum TimeUnit`
			`{`
			`Picoseconds,`
			`Nanoseconds,`
			`Microseconds,`
			`Milliseconds,`
			`Seconds,`
			`Minutes,`
			`Hours`
			`}`

GateTrackingDelay Precision	int	get, set	Gets or sets the precision for Gate Tracking Delay.

GateTrackingDelayUnit	TimeUnit	get, set	Gets or sets the unit for the Gate Tracking Delay.
			`public enum TimeUnit`
			`{`
			`Picoseconds,`
			`Nanoseconds,`
			`Microseconds,`
			`Milliseconds,`
			`Seconds,`
			`Minutes,`
			`Hours`
			`}`

GateTrackingWidth Precision	int	get, set	Gets or sets the precision for Gate Tracking Width.

GateTrackingWidthUnit	TimeUnit	get, set	Gets or sets the unit for the Gate Tracking Width.
			`public enum TimeUnit`
			`{`
			`Picoseconds,`
			`Nanoseconds,`
			`Microseconds,`
			`Milliseconds,`
			`Seconds,`
			`Minutes,`
			`Hours`
			`}`

IncludeAllExperiment Information	bool	get, set	Flag to enable/disable including all experiment information in the export.
IncludeSubdirectories	bool	get, set	Flag to turn off/on subdirectories.
IntensityPrecision	Int?	get, set	Gets or sets the precision for intensities. Can be a null for default values.
LaserLine	double	get, set	Gets or sets the reference line.

Table 7-29: IExportSettings Supported Properties (Sheet 2 of 3)

Name	Type	Accessors	Notes
ExposureStarted Precision	int	get, set	Gets or sets the precision for Exposure started.

ExposureStartedUnit	TimeUnit	get, set	Gets or sets the unit for the exposure started signal.
			public enum TimeUnit
			{
			Picoseconds,
			Nanoseconds,
			Microseconds,
			Milliseconds,
			Seconds,
			Minutes,
			Hours
			}

GateTrackingDelay Precision	int	get, set	Gets or sets the precision for Gate Tracking Delay.

GateTrackingDelayUnit	TimeUnit	get, set	Gets or sets the unit for the Gate Tracking Delay.
			public enum TimeUnit
			{
			Picoseconds,
			Nanoseconds,
			Microseconds,
			Milliseconds,
			Seconds,
			Minutes,
			Hours
			}

GateTrackingWidth Precision	int	get, set	Gets or sets the precision for Gate Tracking Width.

GateTrackingWidthUnit	TimeUnit	get, set	Gets or sets the unit for the Gate Tracking Width.
			public enum TimeUnit
			{
			Picoseconds,
			Nanoseconds,
			Microseconds,
			Milliseconds,
			Seconds,
			Minutes,
			Hours
			}

IncludeAllExperiment Information	bool	get, set	Flag to enable/disable including all experiment information in the export.
IncludeSubdirectories	bool	get, set	Flag to turn off/on subdirectories.
IntensityPrecision	Int?	get, set	Gets or sets the precision for intensities. Can be a null for default values.
LaserLine	double	get, set	Gets or sets the reference line.

Table 7-29: IExportSettings Supported Properties (Sheet 3 of 3)

Name	Type	Accessors	Notes

`LightUnit`	`LightUnit`	get, set	Gets or sets the unit.
			`public enum LightUnit`
			`{`
			`ElectronVolts,`
			`Angstroms,`
			`Nanometers,`
			`Microns,`
			`AbsoluteWavenumbers,`
			`RelativeWavenumbers`
			`}`

`MaxFrame`	`Long?`	get, set	Highest frame number to export or null which is all frames.
`MinFrame`	`Long?`	get, set	Lowest frame number to export or null which signifies no minimum frame number.

`OutputMode`	`ExportOutputMode`	get, set	Replaces the CsvOutputMode property since other types of exports besides Csv files can use these values.
			`Public enum ExportOutputMode`
			`{`
			`OneFile`
			`OneFilePerFrame`
			`OneFilePerROI`
			`OneFilePerRoiPerFrame`
			`}`

`OutputPathOption`	`ExportOutputPathOption`	get, set	Provides the option of putting the output result files in the same folder as the input or using the custom option which will then use the custom path option to determine where the file goes.
			• `InputPath` Each output file goes to the input folder.
			• `CustomPath` Every file goes to the custom path set value. Use `CustomOutputPath` to set the destination for the file(s).
			`public enum ExportOutputPathOption`
			`{`
			`InputPath,`
			`CustomPath,`
			`}`

`SelectedRoi`	`Int?`	get, set	If it is null, all ROIs will be exported; otherwise, you can specify a particular ROI to export.
`WavelengthPrecision`	`Int?`	get, set	Gets or sets the precision for wavelengths. Can be a null for default values.

7.21 IAviExportSettings

7.21.1 Supported Properties

Refer to Table 7-30 for information about properties supported by IAviExportSettings.

Table 7-30: IAviExportSettings Supported Properties

Name	Type	Accessor	Notes
Compressed	bool	get, set	Turns compression on or off for exporting to AVI.
FrameRate	double	get, set	Sets the frame rate for AVI playback.

7.22 ICsvExportSettings

7.22.1 Supported Properties

Refer to Table 7-31 for information about properties supported by ICsvExportSettings.

Table 7-31: ICsvExportSettings Supported Properties (Sheet 1 of 2)

Name	Type	Accessors	Notes
HeaderType	CsvExportHeader	get, set	Type of header: none, short, or long.

Layout	CsvLayout	get, set	The type of layout for the comma separated values file. Layout can be in a tabular or a matrix format.
			`public enum CsvLayout`
			`{`
			`  Table,`
			`  Matrix`
			`}`

IncludeFormatMarkers	bool	get, set	Gets or sets whether or not to show the formatting markers in the file.

MatrixMetadata	IList<CsvMatrixMetadata>	get, set	The meta data to be written out in the file can be a list of the following options.
			`public enum CsvMatrixMetadata`
			`{`
			`  ExposureStartTimeStamp,`
			`  ExposureEndTimeStamp,`
			`  FrameTrackingNumber,`
			`  GateTrackingWidth,`
			`  GateTrackingDelay,`
			`  ModulationTrackingPhase`
			`}`

Table 7-31: ICsvExportSettings Supported Properties (Sheet 2 of 2)

Name	Type	Accessors	Notes

MatrixXAxes	IList<CsvMatrixXAxis>	get, set	X Axis information will be written in the order it is placed in the list.
			`public enum CsvMatrixXAxis`
			`{`
			`Column,`
			`Wavelength,`
			`ExposureStartTimeStamp`
			`ExposureEndTimeStamp,`
			`FrameTrackingNumber,`
			`GateTrackingWidth,`
			`GateTrackingDelay,`
			`ModulationTrackingPhase`
			`}`

MatrixYAxes	IList<CsvMatrixYAxis>	get, set	Type of Y Axes information, currently only one option exists.
			`public enum CsvMatrixYAxis`
			`{`
			`Row = 1`
			`}`

TableFormat	IList<CsvTableFormat>	get, set	Type of table formats to have can include any or all of the following items.
			`public enum CsvTableFormat`
			`{`
			`Frame,`
			`Region,`
			`Column,`
			`Row,`
			`Intensity,`
			`Wavelength,`
			`ExposureStartTime,`
			`ExposureEndTime,`
			`FrameTrackingNumber,`
			`GateTrackingWidth,`
			`GateTrackingDelay,`
			`ModulationTrackingPhase`
			`}`

7.23 IFitsExportSettings

7.23.1 Supported Properties

Refer to Table 7-32 for information about properties supported by IFitsExportSettings.

Table 7-32: IFitsExportSettings Supported Properties

Name	Type	Accessors	Notes
IncludeAllExperimentInformation	bool	get, set	Flag that indicates whether or not to copy all experiment information into the exported file.

7.24 ISpcExportSettings

7.24.1 Supported Properties

Refer to Table 7-33 for information about properties supported by ISpcExportSettings.

Table 7-33: ISpcExportSettings Supported Properties

Name	Type	Accessors	Notes

GlobalTimeUnit	TimeUnit	get, set	For SPC settings all of the time stamping must be in the same unit. This value will be the unit for all time-related properties and will override all the TimeUnit properties in the base class (IExportSettings).
			`public enum TimeUnit`
			`{`
			`Picoseconds,`
			`Nanoseconds,`
			`Microseconds,`
			`Milliseconds,`
			`Seconds,`
			`Minutes,`
			`Hours`
			`}`

IncludeAllExperimentInformation	bool	get, set	Flag that indicates whether or not to copy all experiment information into the exported file.

7.25 ITiffExportSettings

7.25.1 Supported Properties

Refer to Table 7-34 for information about properties supported by ITiffExportSettings.

Table 7-34: ITiffExportSettings Supported Properties

Name	Type	Accessor	Notes
IncludeAllExperimentInformation	bool	get, set	Flag that indicates whether or not to copy all experiment information into the exported file.

7.26 Zone Support

This section provides information about the methods and properties required to support the five UI zones within LightField.

7.26.1 Application Toolbar Zone

The following methods and properties are required to support the Application Toolbar zone:

- `Bitmap UIApplicationToolBarBitmap { get; }`
- `void UIApplicationToolBarExecute();`
- `int? UIApplicationToolBarHelpTopicID { get; }`
- `bool? UIApplicationToolBarIsChecked { get; set; }`
- `bool UIApplicationToolBarIsEnabled { get; }`
- `string UIApplicationToolBarTitle { get; }`
- `string UIApplicationToolBarToolTip { get; }`

7.26.2 Data Toolbar Zone

The following methods and properties are required to support the Data Toolbar zone:

- `System.Drawing.Bitmap UIDataToolBarBitmap { get; }`
- `int? UIDataToolBarHelpTopicID { get; }`
- `bool? UIDataToolBarIsChecked { get; set; }`
- `bool UIDataToolBarIsEnabled { get; }`
- `string UIDataToolBarTitle { get; }`
- `string UIDataToolBarToolTip { get; }`
- `void UIDataToolBarExecute();`

7.26.3 Experiment Settings Zone

The following methods and properties are required to support the Experiment Settings zone:

- `int? UIExperimentSettingHelpTopicID { get; }`
- `string UIExperimentSettingTitle { get; }`
- The `AddinBase` property should be set;
- `FrameworkElement ExperimentSettingElement = Users UI Element`

7.26.4 Experiment View Zone

The following methods and properties are required to support the Experiment View zone:

- `int? UIExperimentViewHelpTopicID { get; }`
- `string UIExperimentViewTitle { get; }`
- The `AddinBase` property should be set;
- `FrameworkElement ExperimentViewElement = Users UI Element`

7.26.5 Menu Zone

The following methods and properties are required to support the Menu zone:

- `bool? UIMenuIsChecked { get; set; }`
- `bool UIMenuIsEnabled { get; }`
- `string UIMenuTitle { get; }`
- `void UIMenuExecute();`

This page is intentionally blank.

Chapter 8: Automation Class

In prior versions, an automation object could be created and the LightField Application could be obtained from that. Version 3.0 added more functionality to the automation class. The new constructor takes a list of command-line options that are passed through to the application on startup.

```
public Automation(bool isApplicationVisible, List<string>
    commandLineOptions)
```

8.1 Properties Table

Table 8-1: Automation Class Properties

Name	Type	Accessors	Notes
IsDisposed	bool	get	Returns true if the object can no longer be used.
WindowHeight	double	get, set	Height of the LightField window.
WindowLeft	double	get, set	Left position of the LightField window.

WindowState	Window State	get, set	Controls the window state of the LightField application can be one of the following.
			`public enum WindowState`
			`{`
			`  Normal,`
			`  Minimized,`
			`  Maximized,`
			`}`

WindowTop	double	get, set	Top position of the LightField window.
WindowWidth	double	get, set	Width of the LightField window.

8.2 Methods Table

Table 8-2: Automation Class Methods

Return	Method	Parameters	Notes
bool	ActivateWindow	none	Tries to activate the window as if you had clicked on it. Returns true if successful and false otherwise.

8.3 Events Table

Table 8-3: Automation Class Events

Event Name	Raised Upon
LightFieldClosed	This event signifies that the LightField application is closed and that the automation object is now longer valid to use.
LightFieldClosing	If the user begins to close LightField by clicking the x, then the object that created it is notified with this event (if it registers for it) and given the chance to stop the closing event at this point.

Appendix A: What Is New in LightField 4.0

Refer to Table A-1 for information about the additions and other changes that have been made to the LightField Add-Ins documentation with the release of Version 4.0.

Table A-1: List of Changes in LightField 4.0 (Sheet 1 of 2)

Item	Existing Table	New Table	Item (A)dded, (C)hanged, or (R)emoved	Page
IFileManager	Methods	—	CreateExportSettings (C)	58
		—	GetDataWorkspaceFiles (A)	59
IExperiment	Properties	—	SystemColumnCalibrationErrors (A)	64
		—	SystemIntensityCalibration (A)	64
	Methods	—	Load (C)	62
IDisplay	Methods	—	DisplayFileInDataWorkspace (A)	71
		—	Create (string fileName) (A)	71
		—	Create (string sourceName, IImageDataSet imageDataSet,) (A)	71
IDisplaySource	—	Methods	Dispose (A)	73
IDisplayViewer	Properties	—	DisplayLocation (A)	76
	Methods	—	DisplayFileInDataWorkspace (R)	—
		—	Create (string fileName) (R)	—
		—	Create (string sourceName, ImageDataSet ImageDataSet) (R)	—
IExportSettings	Properties	—	CsvFormat (R)	—
		—	CsvExportHeader (R)	—
		—	CsvOutputMode (R)	—
		—	OutputPath (R)	—
		—	CustomOutputPath (A)	84
		—	FileType became ExportFileType	84
		—	OutputMode (A)	87
		—	OutputPathOption (A)	87
IAviExportSettings	—	Properties	Compressed (A)	88
			FrameRate (A)	88

Table A-1: List of Changes in LightField 4.0 (Sheet 2 of 2)

Item	Existing Table	New Table	Item (A)dded, (C)hanged, or (R)emoved	Page
ICsvExportSettings	—	Properties	HeaderType (A)	88
			IncludeFormatMarkers (A)	88
			LayoutLayout (A)	88
			MatrixMetadata (A)	88
			MatrixXAxesMatrixXAxes (A)	89
			MatrixYAxesMatrixYAxes (A)	89
			TableFormatTableFormat (A)	89
IFitsExportSettings	—	Properties	IncludeAllExperimentInformation(A)	89
ISpcExportSettings	—	Properties	GlobalTimeUnit (A)	90
			IncludeAllExperimentInformation(A)	90
ITiffExportSettings	—	Properties	IncludeAllExperimentInformation(A)	90

This page is intentionally blank.

www.princetoninstruments.com

TELEDYNE PRINCETON INSTRUMENTS
Everywhereyoulook™
Part of the Teledyne Imaging Group

info@princetoninstruments.com
USA +1 877-474-2286 | **France** +33 (1) 60 86 03 65 | **Germany** +49 (0) 89 660 7793 | **UK & Ireland** +44 (0) 1628 472 346
Singapore +65 6408 6240 | **China** +86 10 659 16460 | **Japan** +81 (3) 5639 2741