

Advanced Architectures in LabVIEW

Overview

During the Advanced Architectures in LabVIEW course, students participate in discussions and work independently and collaboratively to learn how to architect an application and then design the components to support the architecture.

Duration

Three (3) Days

Audience

- LabVIEW programmers who want to learn advanced design patterns
- LabVIEW programmers who are managing the development of a large application
- LabVIEW Architects managing a team of developers
- LabVIEW users pursuing the Certified LabVIEW Architect certification

Prerequisites

- LabVIEW Core 3 or equivalent LabVIEW experience

NI Products Used During the Course

- NI LabVIEW Professional Development System for Windows
- NI LabVIEW Control Design and Simulation Module

After attending this course, you will be able to:

- Refine a requirements document and develop an architecture for a LabVIEW-based application.
- Understand the elements of a good architecture
- Collaborate with a team to create an architecture
- Understand advanced design patterns and how to use them to build the components or subsystems of an architecture.
- Understand the design trade-offs when selecting an advanced design pattern
- Design a clean API
- Analyze, critique, and improve the architecture of a LabVIEW application

Registration

Register online at ni.com/training or call (800)433-3488 Fax: (512)683-9300
info@ni.com

Outside North America, contact your local NI Office.
Worldwide Contact Info: ni.com/global

Part Number

910791-xx
-01 NI Corporate or Branch
-11 Regional
-21 Onsite (at your facility)

Suggested Next Steps

- Managing Software Engineering in LabVIEW
- LabVIEW Object-Oriented Programming System Design
- LabVIEW Performance Guide
- Certified LabVIEW Architect Exam

Advanced Architectures in LabVIEW

Architecting an Application

In this initial lesson, you are given a set of requirements for the course project. You then independently prepare and document an initial architecture for the course project. This lesson allows you to:

- Practice the process of architecting an application
- Identify design challenges in the course project
- Have a framework for better understanding each of the advanced design patterns and the value each brings to the implementation of an architecture

Defining Use Cases and Analyzing Programming Techniques

This lesson includes a discussion of typical types of LabVIEW applications (use cases) and the unique challenges inherent in each. You evaluate the course project and consider how it is similar to and different from the typical use cases that you may face in the future. You then discuss advanced programming techniques at a very high level and identify ways to classify and analyze the techniques that will be taught.

Designing the API

In this lesson you learn good programming style as it relates to creating an API in LabVIEW. You learn how to design a clean, readable, and flexible interface to an API that may become part of a general reuse strategy or may become a component in a custom application. Topics include:

- Learning about consistency, organization, and usability as it applies to an API
- Using Project Libraries to enable API development
- Using Polymorphic VIs to enable API development
- Determining ways to pass data in an API

Advanced Design Patterns

In this lesson, you will be equipped to thoroughly understand essential advanced design patterns in LabVIEW. You will understand the types of problems that are elegantly solved with each of the advanced design patterns. Additionally you will learn about important design concepts such as information hiding and coding to interfaces. As such, you will be prepared to begin with an architecture and then be able to design and construct the components and communication mechanisms for that architecture.

Exercises have been crafted to be general enough so that you can utilize them in your own applications after completing the course. They will include:

- Building components from templates
- Identifying and fixing problems with broken code
- Defining build requirements for plug in design patterns
- Scaling existing components to include new features and capabilities
- Analyzing design patterns
- Understanding how the design patterns contribute to the course project and a readable, maintainable, and scalable architecture

Advanced Architectures in LabVIEW

Advanced design patterns include:

- State machine and queue driven message handlers that are based on the producer/consumer design pattern
- State machine as client/server
- Functional global variables
- Plug ins with VI Server
- Plug ins with LVOOP (Native LabVIEW Classes)
- Daemons and processes
- Reentrant VIs
- Sub panels
- XControls
- Graphical Object Oriented Programming (GOOP)
- Patterns that leverage the new by reference feature in LabVIEW 2009
- Additional communication techniques

Additional Topics and Programming Techniques

In this lesson you will learn about advanced programming techniques and topics that fall outside the scope of advanced design patterns, but are important for the knowledge base of a LabVIEW architect. You will:

- Understand the challenges and goals of error handling
- Build a drop in tool that will expand the features and properties of LabVIEW
- Understand the role and value of VI scripting for the LabVIEW architect

Architecting and Designing Components for an Error Handling System

This lesson provides students an opportunity to implement the techniques you have learned in a collaborative environment. You will be given an overview of the high level requirements of an error handling system and then you will independently fine tune those requirements, draft an architecture, and design some of the components that will comprise the system. During this lesson you will:

- Analyze and clarify requirements
- Draft an initial architecture of the overall system and its basic software components
- Have an opportunity to present designs to fellow students
- Collaborate on a final architecture
- Document the architecture
- Design a component or two of the overall architecture
- Be able to return to their corporate environment with the initial plans and designs for a robust error handling system that could be integrated into any application