Certified LabVIEW Developer (CLD)
Certification and Exam Overview

Certification Overview

The National Instruments LabVIEW Certification Program consists of the following three certification levels:

1. Certified LabVIEW Associate Developer (CLAD)
2. Certified LabVIEW Developer (CLD)
3. Certified LabVIEW Architect (CLA)

Each level is a prerequisite for the next level of certification. The CLAD certification is a prerequisite to taking the CLD exam. The CLD certification is a prerequisite to taking the CLA exam. There are no exceptions to the requirements for each exam.

A CLAD demonstrates a broad and complete understanding of the core features and functionality available in the LabVIEW Full Development System and possesses the ability to apply that knowledge to develop, debug, and maintain small LabVIEW modules. The typical experience level of a CLAD is approximately 6 to 9 months in the use of the LabVIEW Full Development System.

A CLD demonstrates experience in developing, debugging, and deploying and maintaining medium-to-large scale LabVIEW applications. A CLD is a professional with cumulative experience of approximately 12 to 18 months developing medium to large applications in LabVIEW.

A CLA demonstrates mastery in architecting LabVIEW applications for a multi-developer environment. A CLA not only possesses the technical expertise and software development experience to break a project specification into manageable LabVIEW components, but also has the experience to see the project through by effectively utilizing project and configuration management tools. A CLA is a professional with a cumulative experience of approximately 24 months in developing medium to large applications in LabVIEW.

© 2012 National Instruments Corp. All rights reserved

Certified LabVIEW Developer (CLD)
Certification and Exam Overview
Exam Overview

Your test computer will have the LabVIEW Full Development System version 2011 or later installed for developing your application. After October 1, 2012, LabVIEW 2012 may be installed.
Contact your proctor or testing center prior to the exam to get additional details and familiarize yourself with the specific LabVIEW version that will be used to develop your application.
You may request the proctor to allow you a few minutes before your exam to customize the LabVIEW environment for your needs or to familiarize yourself with the environment. The proctor will only hand over the exam when you are ready to begin working on the exam.
Please note that you will not receive extra time for the exam to compensate for unfamiliarity with the LabVIEW environment.
Refer to LabVIEW Development Systems comparison for details about LabVIEW Full Development System features.

Exam Duration: 4 hours
Style of exam: Practical – application development
Passing grade: 70%

The exam validates problem solving skills, knowledge, and experience in the development of measurement and automation applications using LabVIEW. The exam involves software development only and does not involve any hardware.

The use of resources available in LabVIEW, such as the *LabVIEW Help*, examples, and templates are allowed during the exam. Externally developed VIs or resources are prohibited.

The exam packet will include a USB memory stick with a VI and controls in a folder hierarchy that you must use to develop you application. You must not rename the main VI, provided components, or alter the folder hierarchy. All your development VIs and controls must be saved in the provided folder hierarchy.

A detailed application specification will be provided. The specifications consist of general and technical requirements for the application. You **must not** detach the binding staple, copy, or reproduce any section of the exam document. Failure to comply will result in failure.

After you have completed your exam, you must transfer the solution to the provided USB memory stick. Please validate the copied solution on the USB stick before returning it to your proctor.

You **must not** detach the binding staple, copy, or reproduce any section or solution of the exam. Failure to comply will result in failure.

© 2012 National Instruments Corp. All rights reserved

Certified LabVIEW Developer (CLD)
Certification and Exam Overview

Exam Topics

1. Design Concepts
2. User Interface Design
3. Block Diagram Layout and Style
4. Programming Practices
5. SubVI Design Practices
6. Architecture Selection
7. Timing
8. Error Handling
9. Documentation
10. Testing

© 2012 National Instruments Corp. All rights reserved

Certified LabVIEW Developer (CLD)
Certification and Exam Overview
Exam Topics (Overview)

Topic	Subtopic
1. Design Concepts	a. Modularity, scalability, readability, and maintainability b. Cohesion and coupling c. Hierarchical design d. File structure
2. User Interface (front panel window) Design Practices	a. Coloring scheme b. Grouping and aligning objects c. Setting properties d. Customizing objects e. State management i. Static or dynamic ii. At initialization and application stop f. Icon design
3. Block Diagram Design Practices	a. Data flow b. Enhancing readability
4. Programming Practices	a. Data elements b. Functions and subVIs c. Programming structures d. Data structures e. References, Property Nodes
5. SubVI Design Practices	a. Modularity and cohesion b. Front panel layout c. Connector pane and icon
6. Design Pattern Selection	a. Scalability and maintainability b. Responsive and non-blocking c. Design patterns: i. Simple state machine ii. User interface event handler iii. Queued message handler iv. Producer/consumer (data) v. Producer/consumer (events) vi. Functional global variable
7. Timing	a. Timing functions b. Timing mechanisms i. Event structure timeout ii. Synchronization function timeout iii. Timed structures c. Timing Express VIs and functional global variables
8. Errors	a. Error handling b. Error reporting
9. Documentation	a. Front panel window b. Block diagram c. VI Properties
10. Testing	a. Code and documentation review b. Functionality d. Errors

© 2012 National Instruments Corp. All rights reserved

Certified LabVIEW Developer (CLD)
Certification and Exam Overview

CLD Topics Details

1. Design Concepts

a. Modularity, scalability, readability, and maintainability
   1. Develop a LabVIEW application (VI) that is:
      a) Modular – VI functionality is subdivided into modules or subVIs
      b) Scalable – VI requires little or no change to the user interface or block diagram to handle larger data sets or additional program states
      c) Readable – VI conveys information about itself through good documentation and programming style
      d) Maintainable – VI facilitates modifications without changing the original intent of the module or application

b. Cohesion and coupling
   1. Develop LabVIEW modules that are:
      a) Highly cohesive—module has a clearly defined and published goal
      b) Loosely coupled—module minimizes dependency on other modules for completing or complementing its functionality

c. Hierarchical design
   1. Develop a LabVIEW VI that utilizes the preceding techniques to create a logical hierarchical design

d. File structure
   1. Organize the VIs in the file system to reflect the hierarchical nature of the software
   2. Create a folder for the application and give it a relevant name
   3. Make the main (top-level) VI accessible in the application folder
   4. Create separate folders for subVIs and controls

2. User Interface (front panel) Design Practices

a. Coloring scheme
   1. Design the user interface of a VI using a consistent system coloring scheme
   2. Use pastel colors where needed and avoid the use of bright colors
   3. Use guidelines from the *LabVIEW Style Checklist* topic of the *LabVIEW Help* for coloring schemes of background and user interface objects

b. Grouping and aligning objects
   1. Group user interface objects that are logically related by using arrays, clusters, or decorations
   2. Align objects and their labels to provide a uniform and consistent layout

c. Setting properties
   1. Choose appropriate settings for the front panel object to improve usability and performance

d. Customizing objects
   1. Change the cosmetic appearance of a front panel window object
   2. Extend application scalability by creating a type definition or strict type definition of a custom control

© 2012 National Instruments Corp. All rights reserved

e. State management
   1. Set the value or attributes of a control by statically using the property dialog box of an object, or dynamically using Property Nodes
   2. Initialize or set control values at application, load, start, and stop

f. Icon design
   1. Design main (top-level) VI and subVI icons to represent the application or module functionality
   2. Maintain a consistent and uniform icon design scheme between main and subVIs

3. Block Diagram Design Practices

a. Data flow
   1. Enforce data flow by using error terminals on VIs and Property Nodes
   2. Enforce data flow by using Sequence structures for VIs / functions that do not have error terminals

b. Enhancing readability
   1. Develop block diagrams to fit a screen resolution of 1024 x 768
   2. Limit block diagram size so that a user has to scroll in only one direction
   3. Evenly space VIs and functions– avoid crowding too many VIs or functions in a small area
   4. Evenly align VIs and functions using a consistent scheme
   5. Avoid wire bends and keep the wires as straight as possible
   6. Connect wires so that they appear to be connected to the correct terminals
   7. Wire VIs and functions to follow left-to-right and top-to-bottom data flow
   8. Avoid wiring under structures or under structure borders
   9. Avoid overlapping of tunnels on structure borders
   10. Avoid using colors to distinguish between block diagram sections—use system colors only if needed

4. Programming Practices

a. Data elements
   1. Use appropriate data types for controls, indicators, and constants
   2. Directly read from controls—avoid using local or global variables or Property Nodes
   3. Directly update indicators—avoid using local or global variables or Property Nodes
   4. Use local variables to update controls
   5. Use Property Nodes to set attributes for controls and indicators
   6. Use references only if front panel controls or indicators are to be affected from a subVI
   7. Protect access to local and global variable(s) to avoid race conditions
   8. Type define data elements to maintain the same type for all instances and improve scalability
   9. Use constants for data elements that do not need user interface access

© 2012 National Instruments Corp. All rights reserved

b. Functions and subVIs
   1. Optimize function usage—use minimal number of functions to perform a task
   2. Avoid data coercion at inputs
   3. Utilize error terminals where available
   4. Close references where explicitly opened

c. Programming structures
   1. Select an appropriate programming structure for the VI
   2. Initialize shift registers where necessary and recognize the implications of un-initialized shift registers
   3. Avoid deeply nested structures – keep nesting to no more than two levels deep
   4. Use a single frame Flat Sequence structure to enforce data flow where a wire is not available
   5. Avoid using sequence locals to pass data or state information
   6. Wherever possible, replace a Sequence structure with a Case structure to improve scalability
   7. Avoid using default tunnels at structure borders
   8. Use an Event structure to handle user interface events
   9. Use a Timed structure to handle timing events deterministically

d. Data structures
   1. Group logically associated data elements in appropriate data structures
   2. Create working data structures on the block diagram to group and handle data and state information that does not need any user interaction on the user interface

e. References, Property Nodes (obtaining, closing references)
   1. Use implicitly linked Property Nodes for affecting the attributes of objects in the same VI
   2. Use references to front panel objects only if they need to be affected from within a subVI
   3. Recognize the performance implications of Property Nodes and apply them appropriately
   4. Close references if explicitly opened

5. SubVI Design Practices

a. Cohesive and modular
   1. Design a subVI so that it performs one clearly defined function
   2. Call functions and subVIs to help the subVI perform its task

b. Front panel window and block diagram
   1. Design the user interface of a subVI so that the input, output, and working data sections are clearly defined
   2. Include error in and error out clusters and wire them to the connector pane
   3. Wire the error in terminal to a selector terminal on an Error / No Error Case structure
   4. Ensure that the Error case passes through an upstream error
   5. Place all subVI code in the No Error case and ensure the error wire connects the error terminals of functions, subVIs, and Property Nodes

© 2012 National Instruments Corp. All rights reserved

6. Merge errors from different sources and pass the error to the calling VI through the error out terminal
7. Use programming practices from the *Programming Practices* section

c. Connector pane and icon
   1. Use techniques from the *LabVIEW Style Checklist* topic of the *LabVIEW Help* to develop a connector pane
   2. Identify terminals as required, recommended, or optional
   3. Design an icon for the main VI and subVIs that identifies their functionality
   4. Maintain consistent icon style between the main VI and subVIs

6. Design Pattern Selection

a. Scalable and maintainable
   1. Identify the most appropriate design pattern for the main VI by analyzing the project specification
   2. Identify the most appropriate design pattern for subVIs to achieve desired functionality
   3. Utilize an architecture that does not limit scalability or maintainability
   4. Use type defined data structures for state and data management

b. Responsive and non-blocking
   1. Utilize a design pattern that responds to user interface interaction within 100 milliseconds and updates indicators at a reasonable frequency
   2. Utilize a design pattern and techniques that do not utilize 100% of CPU resources

c. Design patterns
   1. Select a simple state machine design pattern for top-level VI in which the UI is polled
   2. Select a simple state machine design pattern in a subVI that needs to execute one or more functions or subVIs based on an input state
   3. Select a user interface event handler design pattern in which a quick response to user interface activity is required
   4. Select a queued message handler design pattern to store one or more states
   5. Select a producer/consumer (data) design pattern to generate and store data/state in the producer loop in parallel with other data consuming loops
   6. Select a producer/consumer (events) design pattern to respond to user interface events in the producer loop and defer the processing of the event to one or more consumer loops
   7. Select a functional global variable design pattern in a subVI instead of a global variable for better performance and access control

7. Timing

a. Execution and software timing
   1. Determine the most appropriate timing function to control the speed at which the VI executes on the system and allows the processor to respond to external events and system tasks
   2. Determine the most appropriate mechanism to execute a software operation or task in or after a set amount of time

© 2012 National Instruments Corp. All rights reserved

b. Timing functions
   1. Use the Wait function to control the loop execution rate and allow the processor to respond to external events and system tasks
   2. Use the Wait Until Next ms Multiple function to control the loop execution rate and to synchronize multiple loops
   3. Avoid using any of the Wait functions to time a software operation
   4. Use the Tick Count function and the Get Date/Time in Seconds function to time software operations
   5. Account for time wrap when using the Tick Count function
   6. Warn user of errors that result due to changes to the system clock if using the Get Date/Time in Seconds function

c. Timing mechanisms
   1. Use the timeout on the Event structure to time software operations
   2. Use the timeout input on Queue and Notifier functions to time software operations
   3. Use a Timing structure to perform deterministic software timing operations

d. Timing Express VIs and functional global variables
   1. Use the Elapsed Time Express VI for timing software operations
   2. Develop timing subVIs using the functional global variable design pattern and utilizing either the Tick Count function or the Get Date/Time in Seconds function

8. Errors

a. Error handling
   1. Initialize the error cluster at application start
   2. Wire to error terminals on functions, subVIs, and Property Nodes
   3. Merge errors from multiple sources where necessary
   4. Define custom error codes using the General Error Handler VI
   5. Determine actions to take when an error occurs
   6. Use corrective error handling wherever possible
   7. Ensure that all running loops terminate upon critical errors by directly using the error wire or an error handling case

b. Error reporting
   1. Use Error Handler or Dialog VIs to report errors or warnings

Documentation

c. User Interface (front panel)
   1. Label user interface objects with a name that represents its function
   2. Provide concise tip strips for user interface controls
   3. Provide special operating instructions where appropriate
   4. Provide an appropriate group name to controls that are grouped in clusters or decorations

d. Block Diagram
   1. Provide concise documentation of the overall algorithm in the main VI and subVIs

© 2012 National Instruments Corp. All rights reserved

2. Document each programming structure with a brief description of its functionality
3. Label wires to show the data on the wire and the direction of data flow
4. Label constants with names that represent their functions
5. Use self-documenting programming practices

e. VI properties
1. Provide concise documentation in the VI Properties of the main VI with the overall purpose of the VI
2. Provide concise documentation in the VI Properties of the subVIs with the overall purpose of the subVI and the description and data type of the

9. Testing

a. Code and documentation review
1. Review user interface, block diagram, and program design to meet the preceding requirements and the *LabVIEW Style Checklist* topic of the *LabVIEW Help*
2. Review documentation to meet the preceding requirements and the *LabVIEW Style Checklist* topic of the *LabVIEW Help*

b. Functionality
1. Ensure that all subVIs are linked when the main VI is opened and a broken arrow does not result
2. Ensure that relative paths are used to access application data files
3. Ensure that each subVI produces the desired output by running VI with test data
4. Test that the integrated application meets the specified functionality
5. Ensure that the application does not utilize the 100% of the system CPU resources and is responsive to user interface interaction within 100 milliseconds
6. Test if front panel controls and indicator are set appropriately at application initialization and shutdown

c. Errors
1. Test if the application produces an error for unintended data input through the user interface controls
2. Test if errors are handled appropriately in subVIs and passed to the respective callers
3. Test if errors are handled appropriately and reported to the user

© 2012 National Instruments Corp. All rights reserved

Certified LabVIEW Developer (CLD)
Certification and Exam Overview

CLD Exam Evaluation Criteria

The CLD exam consists of a total of 40 points, allocated as follows:

- Programming style: **15 points**
- Functionality: **15 points**
- Documentation: **10 points**

  Passing Score: (70%) **28 points**

The following criteria are taken into consideration during exam evaluation:

Style

- Does the application follow the LabVIEW development guidelines?
- Is the VI
  - Readable?
  - Constructed for scalability?
  - Easily maintainable?
  - Overly complex?
- Is the VI constructed in a professional manner?
  - Does the VI use LabVIEW frameworks or design patterns?
  - Minimum requirement is to use a state machine
  - Is the VI hierarchical?
  - Repeated code should be in subVIs
  - Are type-defined enumerated controls used to define states?
- Does the VI use unnecessary temporary variables?
- Are appropriate data types, ranges, and format/precision used for front panel controls?
- Is data grouped in appropriate data structures: arrays or clusters?
- Does the VI use deeply nested structures (2 or more)?
- Does the VI use sequence structures for purposes other than initialization or cleanup?
- Does the VI use local and global variables?
  - Local variables can be used to update controls
  - Are global variables protected to avoid race conditions?
- Are Property Nodes (value) used for updating indicators?
- Are front panels and block diagrams well laid out?
  - Are block diagrams cramped into small spaces?
- Are there unnecessary bends in wires?
- Are objects/wires overlapping?
- Are wires running under structures or structure borders?
- Are the error terminals wired on VIs?
- Are references appropriately closed?
- Is the VI optimized for memory and performance?

© 2012 National Instruments Corp. All rights reserved

Certified LabVIEW Developer (CLD)
Certification and Exam Overview

Functionality

- Is the **Run** arrow broken?
- Does the VI properly perform the requirements listed in the specifications?
- Is the logic correct for Boolean inputs and outputs?
- Does the VI respond to user inputs within the stated time limit (100 milliseconds)?
- Does the VI / subVIs use 100% of CPU resources?
- Is file I/O implemented correctly?
- Does the application stop on error?

Documentation

- Is the VI documented through **File»VI Properties**?
- Are the subVIs documented?
- Are wires documented with appropriate labels?
- Is the functionality documented?
    - Block diagram level
    - Main and nested structure level
- Do front panel controls and indicators have descriptive names?
- Do VIs have descriptive icons?
- Are constants documented?
- Do front panel controls have associated tip strips?
- Does the top-level VI have a non-default icon?
    - Do all subVIs have consistent icon design?

CLD Exam Preparation Resources

Use the following resources for exam preparation. You may also want to use the resources given the CLAD preparation guide, should you need a refresher on some topics.

CLD Exam Preparation

- CLD Preparation E-Kit (includes links to preparation guides, sample exams and webcast)

National Instruments Instructor-led or Self-paced training courses

- LabVIEW Core 1
- LabVIEW Core 2
- LabVIEW Core 3
- LabVIEW Performance (not in the core training path, but a great complementary course for the exam)

© 2012 National Instruments Corp. All rights reserved

Certified LabVIEW Developer (CLD)
Certification and Exam Overview

CLD Exams

The following table lists possible exam scenarios that you may receive to develop a solution for your CLD exam. This list is intended to give a general idea of what exams will be administered, and there may be variations within each exam.

Exam Scenario	Description
Coffee machine	The coffee machine simulates ingredient storage, and performs grinding, brewing and dispensing operations to prepare hot water, coffee and latte.
Microwave oven	The microwave oven simulates manual or pre-configured staged, recipe based cooking.
Security system	The multi-zoned security system simulates the arming, disarming, tamper, bypass and alarm functions.
Thermostat	The thermostat simulates scheduled programmatic heating and cooling control for heating, ventilation and air-conditioning (HVAC) system.
Treadmill	The treadmill simulates manual or multi-staged program mode that controls speed, incline, time and tracks run time and distance.
Vending machine	The vending machine simulates the storage, purchase, and dispensing of products using U.S. currency.

© 2012 National Instruments Corp. All rights reserved