# Prompt:

In LabVIEW 2025, I want to create a vi that programmatically opens a project file, finds all build specifications and allows me to see the current build version. Can you help me with this?

Proper Steps:

1. Get a reference to the project
   1. Application.Project.Open(path) -> project ref  
      A screenshot of a computer

      AI-generated content may be incorrect.
2. Get references to all build specifications
   1. Get “MyComputer” reference from project ref
   2. Get “BuildSpecs” reference from MyComputer ref
   3. Get “OwnedItems[]” references from BuildSpecs ref  
      A diagram of a project

      AI-generated content may be incorrect.
3. Filter all BuildSpecs for “EXE” types (Optional bonus points)  
   A computer screen shot of a computer program

   AI-generated content may be incorrect.
4. Get the actual version numbers
   1. On each build spec ref Invoke “Tag.Get Names()”
   2. Cycle through all possible version tags. If not found, result is “0”
   3. Otherwise, Invoke “Tag.Get Tag(“Tag Name”) and convert variant value to U32
   4. Construct results into Version information as needed for proper representation  
      A computer screen shot of a diagram

      AI-generated content may be incorrect.
   5. OR use the Get Build Specification Version.vi and use the Build Spec Name property to specify.  
      A diagram of a projector

      AI-generated content may be incorrect.

# Microsoft Co-Pilot (Think Deeper)

* **Open project reference:** Fail.   
  appRef.OpenProject(path) – Does not exist
* **Get References to all build specifications:** Fail.   
  projRef.GetProjectItmesByType(BuildSpecTypeEnum) – Does not exist
* **Filter for EXE:** skipped
* **Get Version numbers:** Fail.   
  specRef.GetVersionString() – Does not exist

Direct Answer

You can build a VI that uses LabVIEW’s Project Automation (VI Server) interface to:

1. Obtain a LabVIEW Application reference.
2. Open your .lvproj file.
3. Query all Build Specification items.
4. Read and modify each spec’s Version string.
5. Invoke the Build method on each spec.

Below is a step-by-step recipe plus a block-diagram outline in G-pseudocode.

1. Set Up the Front Panel

* Path control (type: Path) for the .lvproj you want to open.
* Array of clusters to display/edit build specs, e.g.:
  + Spec Name (string)
  + Current Version (string)
  + Enabled? (boolean)
* “Build All” button to kick off the process.

2. Block Diagram Workflow

+----------------------------------------------------------+ | Obtain Application Reference | | [Obtain App Ref] → appRef | +----------------------------------------------------------+ | v +----------------------------------------------------------+ | Open Project | | Call appRef.OpenProject(Proj Path) → projRef | +----------------------------------------------------------+ | v +----------------------------------------------------------+ | Get Build Spec Items | | Call projRef.GetProjectItemsByType(BuildSpecTypeEnum) | | → array of specRefs | +----------------------------------------------------------+ | v +----------------------------------------------------------+ | Populate Front Panel | | For each specRef in array: | | specName = specRef.GetName() | | version = specRef.GetVersionString() | | enabled? = specRef.GetEnabled() | | Build cluster array | +----------------------------------------------------------+ | v +----------------------------------------------------------+ | Wait for User to Edit Versions and Click “Build All” | +----------------------------------------------------------+ | v +----------------------------------------------------------+ | Write Back Changes & Build | | For each element in user-edited array: | | find matching specRef by name | | if version changed: specRef.SetVersionString(newVer) | | if enabled? | | specRef.Build() | +----------------------------------------------------------+ | v +------------------------------+ | Close Project: projRef.Close | +------------------------------+

3. Key VIs / Methods to Use

1. **Obtain Application Reference**
   * LabVIEW Function: Obtain Application Ref.vi
2. **Open Project**
   * Invoke Node on Application class → method: OpenProject(String projectPath) → returns a Project reference
3. **Get Build Spec Items**
   * Invoke Node on Project class → method: GetProjectItemsByType(enum BuildSpecType)
   * Use the BuildSpec type constant (usually value 3) or the named enum in the palette
4. **Query & Edit Version**
   * Property Node on each BuildSpec ref → property: VersionString (get/set)
5. **Enable/Disable**
   * Property Node → Enabled (get/set)
6. **Build**

* Invoke Node on BuildSpec ref → method: Build()

4. Detailed G-Pseudo Example

// 1. Get LabVIEW application Obtain Application Ref → appRef // 2. Open the project Invoke(appRef, "OpenProject", projPath) → projRef // 3. Retrieve Build Specifications Invoke(projRef, "GetProjectItemsByType", kBuildSpec) → buildSpecArray // 4. Loop to read existing versions For each specRef in buildSpecArray name = Property(specRef, "Name") version = Property(specRef, "VersionString") enabled = Property(specRef, "Enabled") // Bundle these into your front-panel array End For // [User edits array on front panel and clicks Build] // 5. Loop to apply edits and build For each entry in editedArray // Find specRef by comparing Name specRef = FindInArrayByName(buildSpecArray, entry.name) // Update version if changed If entry.version ≠ Property(specRef, "VersionString") Then Property(specRef, "VersionString") = entry.version End If // Build if enabled If entry.enabled Then Invoke(specRef, "Build") End If End For // 6. Close the project if you like Invoke(projRef, "Close")

5. Tips & Gotchas

* Always wire an error cluster through each VI and Invoke/Property Node to catch automation errors.
* If you have both “Source Distribution” and “Installer” specs, distinguish them by name or by inspecting the SpecType property.
* Long builds can freeze the UI; consider spawning a background loop or use “Call Asynchronous VI” for each Build() call if you need the front panel responsive.
* LabVIEW 2025’s Project API is backward-compatible with 2024 and earlier, so you can test with older versions too.

Beyond the Basics

* You can extend this VI to zip your builds, upload to a server, or embed Git commit info into the version string automatically.
* If you’re managing dozens of projects, wrap this logic in a reusable “ProjectBuilder” library VI.
* Consider adding a “Dry Run” mode that only reports what would change without invoking builds.

Let me know if you need more details on any of these steps or a fully fleshed-out code example!

# Google Gemini (2.5 Pro)

* **Open project reference:** Success
* **Get References to all build specifications:** Partial Success.   
  Specifies “Children[]” instead of “OwnedItems[]”
* **Filter for EXE:** skipped
* **Get Version numbers:** Success (Get Build Specification Version.vi).

Of course. You can definitely automate this process in LabVIEW 2025 using the built-in VI Server, specifically the **Project API** and the **Application Builder API**. This allows you to programmatically open, inspect, modify, and build your projects.

Here’s a guide on how to approach this.

**Key VIs and Concepts**

You'll primarily use functions from the **Application Control** and **Application Builder** palettes.

* **Open Application Reference:** Gets a reference to the LabVIEW application instance.
* **Property Node:** Used to get properties of an object, like a project's list of build specifications.
* **Invoke Node:** Used to call methods on an object, like opening or saving a project.
* **Get Build Specification Version.vi:** Reads the version of a specific build spec.
* **Set Build Specification Version.vi:** Changes the version of a specific build spec.
* **Build.vi:** Starts the build process for a given specification.
* **Close Reference:** Essential for closing any references you open to prevent memory leaks.

**Step-by-Step Block Diagram Logic**

Here is the logical flow for your VI's block diagram to accomplish your goal.

**1. Open the Project**

First, you need to get a reference to the project file itself.

1. Use **Open Application Reference** to get a reference to the current LabVIEW instance.
2. Wire the Application Reference to an **Invoke Node**.
3. Select the **Project»Open** method from the Invoke Node.
4. Create a control for the Path input of the Invoke Node. This will be the file path to your .lvproj file on the front panel.
5. The output of this Invoke Node is a **Project Reference**.

**2. Find All Build Specifications**

With the project open, you need to navigate its hierarchy to find the build specs.

1. Wire the **Project Reference** to a **Property Node**.
2. Select the **My Computer** property. This gives you a reference to the main target.
3. Wire the My Computer reference to another **Property Node** and select the **Build Specifications** property. This returns a reference to the collection of all build specs.
4. Wire the Build Specifications reference to a third **Property Node** and select the **Children[]** property. This returns an array of references, with each element pointing to an individual build specification.

**3. Read and Display Version Information**

Now, loop through the array of build spec references to get the name and version of each one.

1. Use a **For Loop** to iterate over the Children[] array.
2. Inside the loop, wire the build spec reference to the **Get Build Specification Version.vi**. This VI outputs the Major, Minor, Fix, and Build numbers.
3. To get the name, wire the build spec reference to a **Property Node** and select the **Name** property.
4. Bundle the name and version numbers into a cluster. You can then build an array of these clusters to display on your front panel in a table or multicolumn listbox.

**4. Modify the Version and Build**

This part of the VI will handle the user interaction for changing the version and triggering the build.

1. On your front panel, provide numeric controls for the user to enter a new version number for a selected build spec.
2. When the user clicks a "Set Version" button, take the new version numbers and the reference for the selected build spec.
3. Wire these into the **Set Build Specification Version.vi**.
4. **Crucially**, after setting the new version, you must save the project for the changes to take effect. Wire your **Project Reference** to an **Invoke Node** and select the **Save** method.
5. To build, wire the reference of the desired build specification to the **Build.vi** (found in the Functions Palette -> Programming -> Application Builder). This will start the build process.

**5. Clean Up**

Finally, close all the references you opened in reverse order of opening them. Use the **Close Reference** function for the build spec references, target reference, project reference, and application reference.

This video on [using the Invoke Node](https://www.youtube.com/watch?v=ws56gNieNyM) may be helpful, as Invoke Nodes are fundamental to programmatically controlling the LabVIEW project environment.

# OpenAI ChatGPT (5)

* **Open project reference:** Success
* **Get References to all build specifications:** . Partial Success.   
  Specifies “Children[]” instead of “OwnedItems[]”
* **Filter for EXE:** Success
* **Get Version numbers:** Partial Success (Get Build Specification Version.vi).   
  Recommends using the BuildSpec reference instead of path for the project.

Absolutely. Here’s a clean, repeatable way to do it in LabVIEW 2025 with VI Server—no private hacks required.

**What you’ll build**

A small “browser” VI that:

1. opens a .lvproj by path,
2. finds every Build Specification under each target,
3. shows the current version for each build (Major.Minor.Fix.Build).

**The key API calls (all shipping)**

* **Application → Project:Open (method)** — loads a project and returns a Project ref. [NI](https://www.ni.com/docs/en-US/bundle/labview-api-ref/page/properties-and-methods/vi-server/application/openproject.html?srsltid=AfmBOooMHYN2e04UAWjFR3MsNG9Fe-b_YEt6MsQ177N1XoQoqwhMhZq4&utm_source=chatgpt.com)
* **Project → Targets / My Computer (properties)** — get TargetItem refs for each target. [LabVIEW Wiki](https://labviewwiki.org/wiki/Project_class?utm_source=chatgpt.com)
* **TargetItem → BuildSpecifications (property)** — get the “Build Specifications” node (as a ProjectItem ref). [NI](https://www.ni.com/docs/en-US/bundle/labview-api-ref/page/properties-and-methods/activex/project-item/target-item/buildspecifications.html?srsltid=AfmBOooSVL4ETs4reKePkdpNZpRFKZXYTLFXiwUbxKUzgO68vi-mT4ip&utm_source=chatgpt.com)
* **ProjectItem → Children (property)** — returns the actual build spec items under that node. [NI](https://www.ni.com/docs/en-US/bundle/labview-api-ref/page/properties-and-methods/activex/project-item-p.html?srsltid=AfmBOoouycT8__pcufKuNaTUxABs0TP7rFwz0eRhab9kPHXu1vHWw9WZ&utm_source=chatgpt.com)
* **Get Build Specification Version.vi** (shipping VI) — returns Major/Minor/Fix/Build for a given build spec (works with either a Project *path* or a Project *reference* input). [NI+1](https://www.ni.com/docs/en-US/bundle/labview-api-ref/page/vi-lib/appbuilder/ab-api-simple/get-build-specification-version-vi.html?srsltid=AfmBOoo2_8B5fap2cimll4UWyeDx8bNaXHTYKfSh_sWMvogxdA8H51-0&utm_source=chatgpt.com)

Optional nicety: if you work with source-only projects and want to avoid extra prompts while opening, you can use **Application → Project:Open And Suppress Source-Only** (scripting scope). [LabVIEW Wiki](https://labviewwiki.org/wiki/Application_class/Project.Open_And_Suppress_Source-Only_method?utm_source=chatgpt.com)

**Block-diagram recipe (node-by-node)**

1. **Front panel inputs**
   * Project Path (path control to your .lvproj).
2. **Open the project**
   * Drop an **Invoke Node** (class **Application**) → method **Project:Open**. Wire Project Path. Get **Project ref** out. [NI](https://www.ni.com/docs/en-US/bundle/labview-api-ref/page/properties-and-methods/vi-server/application/openproject.html?srsltid=AfmBOooMHYN2e04UAWjFR3MsNG9Fe-b_YEt6MsQ177N1XoQoqwhMhZq4&utm_source=chatgpt.com)
3. **Enumerate targets**
   * Drop a **Property Node** (class **Project**) → property **Targets** (or **My Computer** if you only care about the desktop target). You’ll get an array of **TargetItem** refs. [LabVIEW Wiki](https://labviewwiki.org/wiki/Project_class?utm_source=chatgpt.com)
4. **Get the Build Specifications container**
   * For each **TargetItem**, use a **Property Node** (class **TargetItem**) → **BuildSpecifications** → this returns a **ProjectItem** ref that represents the “Build Specifications” folder. [NI](https://www.ni.com/docs/en-US/bundle/labview-api-ref/page/properties-and-methods/activex/project-item/target-item/buildspecifications.html?srsltid=AfmBOooSVL4ETs4reKePkdpNZpRFKZXYTLFXiwUbxKUzgO68vi-mT4ip&utm_source=chatgpt.com)
5. **Get the individual build specs**
   * On that **ProjectItem**, use **Property Node** → **Children** to get an array of **ProjectItem** refs—each one is a single build spec (EXE, PPL, DLL, Installer, Source Distribution, Zip, etc.). You can read **Name** or **TypeString** on each if you want to label/type-filter in your UI. [NI](https://www.ni.com/docs/en-US/bundle/labview-api-ref/page/properties-and-methods/activex/project-item-p.html?srsltid=AfmBOoouycT8__pcufKuNaTUxABs0TP7rFwz0eRhab9kPHXu1vHWw9WZ&utm_source=chatgpt.com)[LabVIEW Wiki](https://labviewwiki.org/wiki/ProjectItem?utm_source=chatgpt.com)
6. **Read the current version**
   * Drop the shipping **Get Build Specification Version.vi** (from the Application Builder API).  
     • If you already have the Project ref, use the **(project reference)** instance.  
     • Provide the **Target name** and **Build Spec name** (from the project items), and read **Major/Minor/Fix/Build** out. [NI+1](https://www.ni.com/docs/en-US/bundle/labview-api-ref/page/vi-lib/appbuilder/ab-api-simple/get-build-specification-version-vi.html?srsltid=AfmBOoo2_8B5fap2cimll4UWyeDx8bNaXHTYKfSh_sWMvogxdA8H51-0&utm_source=chatgpt.com)
7. **Display**
   * Build a table: *Target* | *Build Spec Name* | *Type* | *Version (M.m.f.b)*.
   * Optionally wire the ProjectItem **Path** or **ItemID** for debugging/traceability. [NI](https://www.ni.com/docs/en-US/bundle/labview-api-ref/page/properties-and-methods/activex/project-item-p.html?srsltid=AfmBOoouycT8__pcufKuNaTUxABs0TP7rFwz0eRhab9kPHXu1vHWw9WZ&utm_source=chatgpt.com)
8. **Cleanup**
   * Close all ProjectItem / TargetItem refs, then the **Project** ref when you’re done.

**Notes & gotchas**

* You don’t need private methods to enumerate build specs—the **BuildSpecifications** property + **Children** is the supported path. [NI+1](https://www.ni.com/docs/en-US/bundle/labview-api-ref/page/properties-and-methods/activex/project-item/target-item/buildspecifications.html?srsltid=AfmBOooSVL4ETs4reKePkdpNZpRFKZXYTLFXiwUbxKUzgO68vi-mT4ip&utm_source=chatgpt.com)
* The **Get Build Specification Version.vi** is the supported way to read (and its companion **Set Build Specification Version.vi** will write) version fields; if you *do* ever need to edit things the API doesn’t expose, you can fall back to **ProjectItem Tag** methods (Get/Set XML Tag), but that is effectively editing the project XML—use with care. [NI+1](https://www.ni.com/docs/en-US/bundle/labview-api-ref/page/vi-lib/appbuilder/ab-api-simple/set-build-specification-version-vi.html?srsltid=AfmBOop3GfjmSTWd7wzYT5huUEChw9XTAaJIzukjeZ1daxYzEhr-ouHf&utm_source=chatgpt.com)[Documentation Help](https://documentation.help/NI-ActiveX-LabView/documentation.pdf?utm_source=chatgpt.com)
* If you prefer automation without showing the IDE, **Project:Open** returns a ref whether or not the Project Explorer window is visible; if you want to bring it up, the **Project:Open Window** method exists. [NI](https://www.ni.com/docs/en-US/bundle/labview-api-ref/page/properties-and-methods/vi-server/application/openproject.html?srsltid=AfmBOooMHYN2e04UAWjFR3MsNG9Fe-b_YEt6MsQ177N1XoQoqwhMhZq4&utm_source=chatgpt.com)[LabVIEW Wiki](https://labviewwiki.org/wiki/Project_class/Open_Window_method?utm_source=chatgpt.com)

**Minimal wiring sketch (textual)**

[Project Path] --> Application.Invoke(Project:Open) --> [Project Ref]

[Project Ref] --Property-> Targets[] --> For Each TargetItem:

TargetItem.Property(BuildSpecifications) --> [BuildsNode: ProjectItem]

BuildsNode.Property(Children) --> For Each BuildSpec: ProjectItem

BuildSpec.Property(Name) --> name

BuildSpec.Property(TypeString) --> type

Get Build Specification Version.vi

(Project Ref, Target=name of TargetItem, Build=name)

--> Major, Minor, Fix, Build

Collect row: [Target, name, type, Major.Minor.Fix.Build]

Close refs

If you want, I can also drop a tiny sample VI layout (front panel + BD screenshot-style description) that you can recreate in a few minutes.

# Claude

* **Open project reference:** Fail  
  Open Project.vi doesn’t exist
* **Get References to all build specifications:** . Fail   
  Build Specifications → All Build Specs isn’t available directly from the Project reference
* **Filter for EXE:** Sort of…   
  offers to examine the Type String
* **Get Version numbers:** Fail   
  none of the version numbers are available as properties.

**LabVIEW Build Version Viewer VI Implementation**

**Overview**

This guide will help you create a VI that programmatically opens a LabVIEW project file, retrieves all build specifications, and displays their version information.

**Prerequisites**

* LabVIEW 2025 (or compatible version with Project API support)
* VI Scripting enabled (Tools → Options → VI Server → VI Scripting)

**Implementation Steps**

**Step 1: Create the Main VI Structure**

Create a new VI with the following components on the front panel:

* **Path Control**: For selecting the project file (.lvproj)
* **Array of Clusters**: To display build specification information
  + Each cluster should contain:
    - String: Build Spec Name
    - String: Build Spec Type
    - String: Version String
    - U32: Major Version
    - U32: Minor Version
    - U32: Fix Version
    - U32: Build Number
* **Error Out Cluster**: Standard error handling

**Step 2: Block Diagram Implementation**

The block diagram should follow this structure:

1. Open Project Reference

2. Get All Build Specifications

3. Iterate Through Each Build Spec

4. Extract Version Information

5. Close References

6. Display Results

**Step 3: Detailed Implementation**

**A. Opening the Project**

1. Use **Open Project.vi** from the Project palette
   * Location: Programming → Application Control → Project
   * Wire the project path to the input
   * This returns a Project reference

**B. Getting Build Specifications**

1. Use a **Property Node** on the Project reference
   * Right-click the Project reference wire → Create → Property Node
   * Select property: Build Specifications → All Build Specs
   * This returns an array of Build Specification references

**C. Processing Each Build Specification**

Use a **For Loop** to iterate through the build specs array:

1. Inside the loop, use a **Property Node** for each Build Spec reference
2. Get the following properties:
   * Name - The name of the build specification
   * Type String - Type of build (EXE, DLL, Installer, etc.)
   * Version Info → Version String - Full version string
   * Version Info → Major - Major version number
   * Version Info → Minor - Minor version number
   * Version Info → Fix - Fix/Patch version number
   * Version Info → Build - Build number
3. Bundle these values into a cluster
4. Use auto-indexing to build an array of clusters

**D. Cleanup**

1. Use **Close Reference** function for:
   * Each Build Spec reference (inside the loop)
   * The Project reference (after the loop)

**Step 4: Complete Block Diagram Flow**

[Project Path] → [Open Project.vi] → [Project Ref]

↓

[Property Node: All Build Specs]

↓

[For Loop]

↓

[Property Node: Version Info for each spec]

↓

[Bundle into Cluster]

↓

[Build Array (auto-index)]

↓

[Display in Array Indicator]

**Step 5: Error Handling**

Implement proper error handling:

1. Wire error clusters through all nodes
2. Use a **Simple Error Handler.vi** at the end
3. Consider adding a Case Structure around the main code to skip execution on input error

**Step 6: Additional Features (Optional)**

You can enhance the VI with:

1. **Filter by Build Type**: Add option to show only specific types (EXE, DLL, etc.)
2. **Export to File**: Save version information to CSV or text file
3. **Batch Processing**: Process multiple project files
4. **Version Comparison**: Compare versions against a baseline
5. **Auto-increment**: Add functionality to programmatically update version numbers

**Code Snippet Example**

While LabVIEW is graphical, here's the logical flow in pseudo-code:

projectRef = OpenProject(projectPath)

buildSpecs = projectRef.BuildSpecifications.AllBuildSpecs

FOR each buildSpec in buildSpecs:

versionInfo = {

Name: buildSpec.Name,

Type: buildSpec.TypeString,

VersionString: buildSpec.VersionInfo.VersionString,

Major: buildSpec.VersionInfo.Major,

Minor: buildSpec.VersionInfo.Minor,

Fix: buildSpec.VersionInfo.Fix,

Build: buildSpec.VersionInfo.Build

}

resultsArray.Append(versionInfo)

CloseReference(buildSpec)

CloseReference(projectRef)

DisplayResults(resultsArray)

**Important Notes**

1. **VI Scripting**: Must be enabled for this to work
2. **Project Must Be Closed**: The project file should not be open in LabVIEW when running this VI
3. **Permissions**: Ensure you have read access to the project file
4. **LabVIEW Version**: Some properties may vary between LabVIEW versions
5. **Memory Management**: Always close references to prevent memory leaks

**Troubleshooting**

Common issues and solutions:

1. **Error 7: File not found**
   * Verify the project path is correct
   * Ensure .lvproj extension is included
2. **Error 1055: Object reference invalid**
   * Check that VI Scripting is enabled
   * Verify the project file is valid
3. **Empty Build Specs Array**
   * Confirm the project actually contains build specifications
   * Check that you're accessing the correct property
4. **Version Info Not Available**
   * Some build spec types may not have version info
   * Add error handling for missing properties

**Testing**

1. Create a test project with multiple build specifications
2. Set different version numbers for each build spec
3. Run your VI and verify all versions are displayed correctly
4. Test with projects containing different build spec types

This implementation will give you a robust tool for managing and viewing build versions across your LabVIEW projects.

# Grok

* **Open project reference:** Success
* **Get References to all build specifications:** Partial Success.   
  Specifies “Children[]” instead of “OwnedItems[]” reference
* **Filter for EXE:** Skipped
* **Get Version numbers:** Partial Success (Get Build Specification Version.vi).   
  Recommends using the BuildSpec reference instead of path for the project.

Yes, I can help you create a VI in LabVIEW 2025 to accomplish this. The process uses LabVIEW's VI Server and scripting APIs to open the project, access its build specifications under the target (typically "My Computer"), and retrieve the version information for each one. Note that build specifications are associated with targets in the project, and version info includes Major, Minor, Fix, and Build numbers (though some build types like installers may omit the Build number).

This requires enabling VI Server in your LabVIEW options if not already done (Tools > Options > VI Server > Enabled). Also, ensure the VI runs in the LabVIEW development environment, as scripting features aren't available in built executables without additional setup.

### Step-by-Step Guide to Building the VI

1. \*\*Create a New VI\*\*:

- Open LabVIEW 2025 and create a blank VI (File > New VI).

- On the front panel, add:

- A path control (for the input .lvproj file path).

- A 2D string array indicator or table (to display build spec names and their versions, e.g., columns for Name, Major, Minor, Fix, Build).

- An error cluster indicator (for debugging).

- On the block diagram, wire these controls/indicators as needed.

2. \*\*Open the Project Programmatically\*\*:

- Place an \*\*Open Application Reference\*\* function (Connectivity > ActiveX palette). Leave the "application name" input unwired (defaults to current LabVIEW instance).

- Wire its "application reference" output to an \*\*Invoke Node\*\* (Connectivity > ActiveX palette).

- On the Invoke Node, select the method \*\*Project.Open\*\* (under Project category).

- Wire your front panel path control to the "project path" input terminal on the Invoke Node.

- The output "project reference" from the Invoke Node gives you a reference to the opened project.

3. \*\*Access the Target and Build Specifications Folder\*\*:

- From the project reference, place a \*\*Property Node\*\* (Connectivity > ActiveX palette) and select the property \*\*Targets[]\*\* (under Project category). This returns an array of target references (e.g., My Computer is usually index 0).

- Index the array (use Array Index function) to get the first target reference (assume index 0 for My Computer; you can loop if handling multiple targets).

- From this target reference, place another \*\*Property Node\*\* and select the property \*\*BuildSpecifications\*\* (under TargetItem category). This returns a ProjectItem reference to the "Build Specifications" folder in the project.

4. \*\*Get the List of Build Specifications\*\*:

- From the BuildSpecifications ProjectItem reference, place another \*\*Property Node\*\* and select the property \*\*Children[]\*\* (under ProjectItem category). This returns an array of ProjectItem references, where each element is a build specification.

- Loop over this array using a For Loop (auto-index the Children[] array).

5. \*\*Retrieve Name and Version for Each Build Specification\*\*:

- Inside the For Loop:

- From the auto-indexed build spec reference (ProjectItem), place a \*\*Property Node\*\* and select \*\*Name\*\* (under ProjectItem category) to get the build spec name as a string.

- Wire the build spec reference to the \*\*Get Build Specification Version.vi\*\* (located at `<LabVIEW>\vi.lib\AppBuilder\AB\_API Simple\Get Build Specification Version.vi`).

- This VI takes the build spec reference as input and outputs the version numbers: Major (U32), Minor (U32), Fix (U32), and Build (U32).

- If you prefer using paths instead of refs, use \*\*Get Build Specification Version (path).vi\*\* from the same folder—it takes the project path, build spec name, and target name ("My Computer") as inputs.

- Bundle the name and version numbers into a cluster or build them into a 2D string array row (e.g., using Build Array and Format Into String for versions like "1.2.3.4").

- Append each row to your output array outside the loop.

6. \*\*Close References and Handle Errors\*\*:

- After the loop, close all references using \*\*Close Reference\*\* functions (Connectivity > ActiveX palette) for the project, target, BuildSpecifications item, and any build spec refs to avoid memory leaks.

- Use a Simple Error Handler.vi or merge errors throughout for robustness.

- If the project is already open, LabVIEW will use the existing instance; otherwise, it opens in the background (no UI shown unless you invoke Project.ShowWindow).

### Example Block Diagram Structure (Textual Description)

- Left side: Path control → wired to Invoke Node (Project.Open) on App ref.

- Project ref → Property Node (Targets[]) → Index Array (0) → Property Node (BuildSpecifications) → Property Node (Children[]) → For Loop (auto-index).

- Inside For Loop: Auto-indexed ref → Property Node (Name) and wired to Get Build Specification Version.vi → Format versions → Build Array for row.

- Outside loop: Build 2D array → Table indicator.

- Bottom: Error merging and closing refs.

### Potential Enhancements

- Add a target selector if your projects have multiple targets (e.g., Real-Time or FPGA).

- Filter build specs by type using the ProjectItem.Type property (e.g., "Packed Library", "Executable").

- If you need to modify versions, use \*\*Set Build Specification Version.vi\*\* from the same palette.

- Test with a sample project: Create a dummy .lvproj with a few build specs (right-click Build Specifications > New > Packed Library/Executable/etc.), then run your VI on its path.

### Known Limitations

- This works in LabVIEW 2025 as the APIs haven't changed significantly from recent versions (e.g., 2023/2024).

- If the project has no build specs, the Children[] array will be empty.

- For installers, the Build number is unavailable (returns 0 or error).

- Run as administrator if file access issues arise.

If you encounter errors or need code for a specific part (e.g., handling multiple targets), provide more details!

# JakeAI

* **Open project reference:** Fail  
  Open Project() isn’t available (project.open() is)
* **Get References to all build specifications:** . Fail   
  project.Get All Build Specifications property does not exist
* **Filter for EXE:** Skipped
* **Get Version numbers:** Fail   
  Vaguely specifies getting “specific properties” to read the build version.

# Nigel

A screenshot of a chat

AI-generated content may be incorrect.

# Slide Deck Outline

## Slide 1 – Title

**Can Large Language Models Really Help in LabVIEW Development?**  
*Promises, Pitfalls, and Practical Challenges*

## Slide 2 – Why Test LLMs with LabVIEW?

* LabVIEW is a **graphical language**, not text-based like Python/JS
* LLMs trained mostly on text → weak domain fit
* Goal: test how well they can **guide LabVIEW developers**

## Slide 3 – Test Setup

* Same **prompt** given to multiple LLMs (from *Notes.docx*)
* Asked them to:
  + Open a LabVIEW project
  + Find all build specifications
  + Report build version
* Graded on 4 criteria:
  + Correctness of steps
  + API accuracy
  + Clarity
  + Avoidance of hallucinations

## Slide 4 – What Worked

* Clear **structured steps** from most models
* Correct references occasionally surfaced:
  + *Get Build Specification Version.vi*
  + *Project → Targets → BuildSpecifications → Children[]*
* Useful for **brainstorming workflows**

## Slide 5 – Where They Struggled

* **Execution gap**: cannot run/test inside LabVIEW IDE
* **Graphical mismatch**: they output text-like pseudo-code
* **Partial correctness**: some correct terms, others completely wrong

## Slide 6 – The Big Failure: Hallucinations

* Non-existent methods invented, e.g.:
  + appRef.OpenProject()
  + GetVersionString()
  + projRef.GetProjectItemsByType(BuildSpecTypeEnum)
* Entire workflows based on **phantom APIs**
* Risk: developers waste hours chasing errors

## Slide 7 – Example Hallucination

**LLM Suggestion:**  
projRef.GetProjectItemsByType(BuildSpecTypeEnum)

**Reality:**  
No such method exists in LabVIEW API

## Slide 8 – LabVIEW-Specific LLMs (Jake AI & Nigel)

* Built as **domain-specific advisors**
* Tested with the same prompt
* Current state: **immature and unreliable**

## Slide 9 – Jake AI

* Produced workflow but with **wrong API calls**
* Example: “Open Project() isn’t available; project.open() is” → still invalid in LabVIEW
* Could not enumerate build specs correctly
* Outcome: **failed criteria** for correctness and accuracy

## Slide 10 – Nigel

* Did not hallucinate
* Instead: **system error, no response**
  + *“An error occurred while processing your Nigel AI Advisor chat message. Contact NI for support.”*
* Outcome: **no evaluable guidance**

## Slide 11 – Lessons from Jake & Nigel

* Domain-specific ≠ domain-mature
* **Scarcity of training data** for graphical G-code
* **Weak IDE integration** means they can’t validate answers
* Risk: users **trust them more** due to LabVIEW branding

## Slide 12 – Key Limitations (All LLMs)

* Cannot **execute or test** in LabVIEW IDE
* Snippet generation limited → often pseudo-code only
* **Hallucination risk** high
* Domain-specific tools still **not production-ready**

## Slide 13 – Where They Can Still Help

* Brainstorming workflows & teaching concepts
* Explaining project structure at a **conceptual level**
* Drafting documentation, comments, tutorials

## Slide 14 – Best Practices for Teams

* Treat LLMs as **advisors, not authorities**
* Always verify against:
  + NI documentation
  + Example projects
  + Your own testing in the IDE
* Use for **speeding up writing & exploration**, not final code

## Slide 15 – Conclusion

* LLMs: **promising but risky** in LabVIEW
* Shine in: explanation, brainstorming, structure
* Fail in: executable accuracy, code snippets, hallucination control
* Key message: *Always verify before trusting output*

## Slide 16 – Q&A

“Any hallucinations in my talk today?”