Choosing the Right Software ADE for Your Automated Test System

Today, test engineers will spend most of their development time working with an ADE. Hence, it is critical to select an ADE that is not only intuitive, but can support multiple platforms and integrate easily with measurement and control services such as drivers. Other features that should be considered when selecting an ADE for the development of your test system are its presentation and reporting features, how protected you are from the obsolescence of the product and what kind of training and support is available worldwide. This paper discusses how three different ADE’s, NI LabVIEW, NI LabWindows/CVI and Microsoft® Visual Studio .NET compare on these characteristics.

Factors to Consider When Selecting an ADE

Ease of Use for New Software Engineers
Because the ADE is where the heart of an automated system is developed, ease of use in these tools is critical to the productivity of a new software engineer. Ease of use goes beyond how quickly someone can get up and running. For example, developers should be able to easily integrate processing routines with multiple measurement devices, create sophisticated user interfaces, deploy and maintain an application, and modify the application as product designs evolve and system needs expand. Other features that should be included with the ADE include extensive documentation and example code.

Measurement and Analysis Capabilities
It is critical that the ADE used to develop a test system can seamlessly manage and process measurements. To do this effectively the ADE should incorporate measurement data types directly in the environment so that the data is easy to use in additional processing routines. For maximum development productivity, the ADE should include comprehensive statistical and numerical analysis functions, as well as high-performance signal processing and control algorithms common in measurement applications.

Multicore and Parallelism Support
Multicore technology has become a standard feature in automated test systems and a necessity for today’s electronic devices that are processing unprecedented amounts of data. Multicore processors present new software challenges that must be overcome to fully take advantage of processing capabilities in a multithreaded application. An ADE must offer developers with programming techniques to create processes to execute in parallel.

Integration with Measurement and Control Drivers
Too often, developers of test systems assume the existence of a device driver alone is sufficient for effectively integrating their measurement device. The existence of a driver is not enough; measurement and control drivers should be integrated as seamlessly as possible with the ADE. In the ideal case, the software that controls the measurement devices is transparent, appearing only as part of the ADE. This ideal implementation guarantees maximum flexibility in
development and a scalable architecture that organizations can deploy on all of the platforms targeted by the ADE.

**Training and Support**

The ease of use of an ADE can only go so far in making it simple for new users to learn the application. Hence, the ADE vendor should provide manuals and online training in order for engineers to quickly learn how to use their product. Advanced users might also need classroom training to further their knowledge and learn more about system level design concepts. This classroom training should give developers the opportunity to attain proof of their knowledge by going through a certification process. Another option that should be considered when selecting an ADE is the type of support the vendor provides you have access to when developing your application, such as phone and email support. Furthermore, if you are going to standardize on an ADE worldwide you will want to consider whether your engineers around the world will have access to support in their own language.

**Platform Independence**

Test software applications today target multiple different architectures. It is important that whatever ADE you choose can be flexible enough to support all these different architectures as seamlessly as possible. Different operating systems such as Windows, Linux® and Macintosh can offer different benefits depending on the application. Engineers should be able to port their code from one platform to the other. If the ADE does not support these multiple platforms you will need to use different ADEs for different projects and spend unneeded time porting your existing intellectual property from one platform to the other.

**Presentation and Reporting Features**

Test applications present many challenges in the area of presentation and reporting due to their emphasis on the graphical representation of data. The ADE should have multiple visual components for data visualization such as charts, graphs, knobs and meters. Reporting should also be easy in order to facilitate the communication of the information acquired by the system. Some of the most popular reports, such as MS Word, MS Excel, etc., should be easy to generate. The communication of results should also be easy to implement by either publishing the application on the web or logging information to a database.

**Protection Against Obsolescence**

Standardizing on an ADE for the development of your test system is a big commitment. It is important that your investment is protected from the obsolescence of the product. One of the characteristics you should consider is the product’s track record of integrating with the latest software technologies and its commitment to protecting you against discontinuous shifts in test software development. Furthermore, the product should go through routine upgrades to add new functionality.
Table 1. Different ADEs Provide Different Benefits and Challenges When Used to Develop Your Test System.

**NI LabVIEW**

NI LabVIEW is a graphical development language that helps engineers and scientists create flexible and scalable test applications rapidly and at minimal cost. NI LabVIEW uses a graphical development paradigm instead of relying on text based programming. The LabVIEW graphical dataflow language and block diagram approach naturally represent the flow of your data and intuitively map user interface controls to your data, so you can easily view and modify your data or control inputs. Figure 1 depicts the block diagram of an application and its respective front panel written in NI LabVIEW.
NI LabVIEW also includes features to facilitate the reference of the extensive documentation included with the product. The Context Help feature lets you leverage the graphical nature of NI LabVIEW to access a subVI’s documentation by simply hovering over it. NI LabVIEW also emphasizes the use of the hundreds of example programs available with the product and online as a means of demonstrating and teaching different features.

Despite the sophistication of the underlying algorithms, LabVIEW analysis tools are easy to use. More than 15 analysis Express VIs, such as the Spectral Measurements Express VI, reduce the complexity of implementing measurement analysis in your application through interactive configuration dialogs in which you can preview analysis results immediately. These and other measurement analysis tools can input real-world, time-domain signals directly from data acquisition hardware and provide results ready for charting, graphing, or further processing. With these functions, you easily can determine signal characteristics such as DC/RMS levels, total harmonic distortion (THD/SINAD), impulse response, frequency response, and cross-power spectrum.

To benefit from the improved processing performance of multicore technology, however, engineers must be able to program their test code to target the different cores. NI LabVIEW is inherently parallel and can automatically generate programs optimized for multiple processing
cores. In LabVIEW, two loops that do not share a data dependency automatically execute in separate threads, as shown in Figure 2.

![Multithreaded Application](image)

*Figure 2. NI LabVIEW Inherently Handles the Parallelization of Code to Target Multiple Cores.*

Consider Table 2 for a representative example of the performance improvement achieved with an existing LabVIEW application on a dual-core processor versus a single-core processor, like the 2.5 GHz Intel Core i5-2510E dual-core processor used in the new NI PXIe-8115 embedded controller. Through LabVIEW, engineers can take advantage of existing code on multicore processors to enhance the performance of their test applications.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Single-Core 2 GHz</th>
<th>Dual-Core 2 GHz</th>
<th>Percentage Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program to find the prime numbers in the first 1,000,000 natural numbers</td>
<td>6.87s</td>
<td>3.59s</td>
<td>47.74%</td>
</tr>
<tr>
<td>Program to calculate 1,000 digits of Pi</td>
<td>3.96s</td>
<td>3.06s</td>
<td>22.73%</td>
</tr>
</tbody>
</table>

*Table 2. By Executing Tasks in Parallel, the same LabVIEW Application, Which Finds all the Prime Numbers in the First 1,000,000 Natural Numbers, Runs 47.74 percent Faster on a Dual-core processor.*

LabVIEW offers tight integration with measurement and control drivers. NI LabVIEW simplifies connecting to and communicating with thousands of instruments from hundreds of vendors.
With NI LabVIEW, you can quickly acquire data from GPIB, serial, Ethernet, PXI, USB, and VXI instruments using instrument drivers, interactive assistants, and built-in instrument I/O libraries. Furthermore, LabVIEW includes easy to use libraries and interactive assistants to communicate with National Instruments’ line of Modular Instruments and Data Acquisition products.

National Instruments offers LabVIEW training for any level of expertise. While basic courses are targeted towards non-programmers and existing developers who want to learn the product, intermediate and advanced users will also find content that is useful to their level of expertise. Onsite courses help organizations train a large number of developers quickly without having to leave the company. On-line and self-paced courses target those developers who wish to increase their knowledge on their own time and at their own pace.

Although NI LabVIEW is usually seen as a MS Windows application, the product’s original operating system was the Macintosh. National Instruments ported NI LabVIEW to Windows as the operating system’s importance increased in the desktop PC industry. NI LabVIEW’s commitment to supporting new platforms continues today. NI LabVIEW continues to support both MS Windows and Macintosh but has also added support for Linux® due to its increasing popularity amongst customers. Being able to run NI LabVIEW VIs on different operating systems means that no matter what computing platform you need to work with, you will be able to leverage your NI LabVIEW knowledge. NI LabVIEW can even run on other targets such as Real-Time systems and even FPGA’s and DSPs.

The presentation and reporting features of LabVIEW are a big part of why the ADE is so suited for test software development. LabVIEW contains multiple graphs, charts, meters, knobs, and switches both in 2D and 3D in order to facilitate the representation of measurement data graphically. The ADE also includes the LabVIEW Report Generation Toolkit which facilitates the creation of reports in MS Word and Excel format. If it is necessary to communicate results by exporting the application through the web LabVIEW Remote Panels can be used to display the front panel over the web on any browser. On the other hand, if the results of your measurements need to be logged to a database, the LabVIEW Database Connectivity toolkit provides a set of easy-to-use tools with which you can quickly connect to local and remote databases and perform many common database operations.

Finally, National Instruments has emphasized throughout time that it is committed to help its LabVIEW customers fight obsolescence. Even though a large amount of development effort has been focused on adding new features and integrating new technologies, running code from previous versions on newer version has always been a priority. Running older code on newer versions of the product means that the valuable resources that were dedicated to creating previous applications will not be wasted and can be leveraged in newer developments.

**NI LabWindows/CVI**

LabWindows/CVI is a proven test and measurement ANSI C development environment that greatly increases the productivity of engineers and scientists. Figure 3 displays the LabWindows/CVI development environment.
Figure 3: LabWindows/CVI features a complete workspace you can use to quickly develop, debug, and manage large applications.

Engineers and scientists use LabWindows/CVI to develop high performance, stable applications in the manufacturing test, military and aerospace, telecommunications, design validation, and automotive industries. LabWindows/CVI streamlines development in these areas with hardware configuration assistants, comprehensive debugging tools, and interactive execution capabilities developers can use to run functions at design time.

Toolkits such as the Advanced Analysis Library complement the analysis libraries included with LabWindows/CVI to help engineers make analyze their measurement data. The LabWindows/CVI Advanced Analysis Library offers a comprehensive set of functions for analyzing your data. With these powerful analysis routines, you can easily convert raw data into useful information and build test applications. The Advanced Analysis Library includes functions for signal generation, one-dimensional (1D) and two-dimensional (2D) array manipulation, complex operations, signal processing, statistics, and curve-fitting.

LabWindows/CVI is an industry leader in instrument control and connectivity, through an Instrument Driver Network of more than 8,000 instrument drivers from more than 200 vendors. You can use these drivers to easily program instrument control applications. With Instrument I/O Assistant, you can generate code to communicate with devices such as serial, Ethernet, and
GPIB instruments without using an instrument driver. Instrument I/O Assistant offers a simple interface for quickly prototyping applications and auto parsing instrument data without any programming. You can easily import the code generated into any existing application, which removes the tedium of writing instrument connectivity, basic communication, and string parsing code. In addition to the integrated NI-DAQmx Libraries, LabWindows/CVI also provides DAQ Assistant, an interactive interface to the data acquisition driver framework.

The training and support structure available for LabVIEW, is also for LabWindows/CVI. LabWindows/CVI has different training courses which target different levels of expertise with the product. Onsite courses are also available for organizations that need to train a large number of developers quickly without having to leave the company. Options are also available for engineers who wish to increase their knowledge on their own time and at their own pace in the form of on-line and self paced courses. In order to complement the training opportunities for LabWindows/CVI worldwide support is provided by National Instruments application engineers from local branches around the world.

By maintaining the backwards compatibility of LabWindows/CVI, National Instruments helps to protect you from obsolescence. Not only can you run C code developed many years ago, or LabWindows/CVI code created in previous version of the product, but you can also have the applications run faster with new optimizing compiler integration. LabWindows/CVI's commitment to backwards compatibility is critical to industries that value longevity and continuity such military and aerospace.


Visual Studio .NET provides a very powerful ADE by supporting multiple programming languages such as C++, Visual Basic .Net, C# and ASP.NET. By offering you the option to select any of these programming languages you can use the same tool and leverage the expertise of your developers even if their knowledge focuses on different programming languages. Applications developed in Visual Studio .NET can be run of a personal computer as well as through the web by using the ASP.NET language.

The Visual Studio .Net provides functionality to develop in different programming languages such as C++, Visual Basic .Net and C#. By enabling these programming languages to compile to the Common Language Runtime you can add libraries developed in different languages. On the other hand, the fact that the .Net platform is only supported by the Microsoft Windows operating system means that you are very limited in the number of operating systems that can run your application. Furthermore, porting your application to another operating system in the future might require rewriting the application in a different language.

By default, Visual Studio .NET does not include any functionality to integrate with measurement and control drivers or perform any analysis operations. Components such as those offered by NI Measurement Studio, as shown in Figure 4, can provide access to measurement and instrument drivers and analysis functionality. These components increase the integration of the ADE with instrument and measurement drivers by providing interactive assistants to generate code automatically. In contrast, there are certain features of the .NET framework that make it inherently difficult to communicate with certain instruments. The .NET framework executes code in the Common Language Runtime which isolates you from accessing the hardware. By not being able to access the hardware it is very difficult to write directly to an instrument’s registers. In order to do this one would have to create a DLL and then call it from a .NET application.
Figure 4. NI Measurement Studio Provides Access to Measurement and Instrument Drivers and Analysis Functionality.

Visual Studio .NET offers few presentation and reporting capabilities by default. Out of the box, the ADE provides enough features to generate a standard Windows application by offering text boxes, combo boxes, list boxes, buttons and other components that are needed to create a basic application. In order to use more powerful components to display data such as graphs and charts you will need to purchase a set of components for this particular application. This problem is also reflected in the lack of reporting tools for any of the programming languages in Visual Studio .NET. On the other hand, the .NET framework includes powerful features for reporting by storing information to a database. ADO .NET, a rich library of database functionality, can be used to communicate with and perform operations on many different databases.

The focus of .NET lies in business, IT and web based application instead of automated test. For this reason, guaranteeing the longevity of the programming language and avoiding discontinuous shifts is not a priority. Applications that focus on IT instead of automated test can have a life cycle of a few months instead of years in the case of automated test. For example, even though it is possible to integrate DLLs into the .NET this requires the developer to manually invoke the function and guarantee that the DLL data types match those in .NET. At
first this might not seem very challenging, but if you need to communicate with hundreds of functions of an instrument driver, this process can be very time consuming. On the other hand, incorporating your existing ActiveX components into a .NET automated test application is easier than incorporating DLLs. Visual Studio .NET can generate wrappers around your ActiveX components to expose them as .NET objects.

**Choosing an ADE that Integrations with Instrument Drivers**

Developing your instrument driver strategy does not have to be a tedious process, but you should have a good understanding of your instrument control options before you finalize your test strategy. For example, using the driver that is shipped with each of your instruments may not always be the best solution for your test system. In an ideal world this might work, but in reality not all instrument drivers are created equal. Most test systems use instruments from multiple vendors and may not be interoperable, or you might have to program a legacy instrument that does not have a driver supported in your ADE. It is also important to document an official driver strategy when you are the one developing the tester because you may not be the one maintaining it or developing the test code for each DUT. This section provides background information on the different instrument driver approaches.

**Instrument Driver Options**

An instrument driver is a set of software routines that control a programmable instrument. Each routine corresponds to a programmatic operation such as configuring, reading from, writing to, and triggering the instrument. Instrument drivers simplify instrument control and reduce test program development time by eliminating the need to learn the programming protocol for each instrument.

You can choose from three standard approaches for programming your instruments: Plug and Play instrument drivers, IVI instrument drivers, and Direct I/O. Each approach offers advantages for different use cases and application needs, as shown in Figure 5. Plug and Play instrument drivers are the most common instrument control method because they offer the ideal mix of instrument accessibility (typically 80 percent of the instrument functions are available) and they deliver ease of use through a straightforward programming paradigm that abstracts the low-level instrument function calls. Use the following descriptions of each approach to help you make educated decisions about your instrument driver strategy.
Plug and Play Instrument Driver
A Plug and Play instrument driver is a set of functions used to control and communicate with a programmable instrument. Each function corresponds to a programmatic operation such as configuring, reading from, writing to, and triggering the instrument. Plug and Play instrument drivers comply with programming guidelines. Because these drivers maintain a common architecture and interface and they include application examples, you can quickly and easily connect to and communicate with your instruments with little or no code development. Moreover, with the standard programming model of Plug and Play instrument drivers, you can easily add instruments to your test system without worrying about learning new communication protocols or spending time understanding new programming paradigms. Plug and Play instrument drivers provide source code native to the development environment. With access to source code, you can modify, customize, optimize, debug, and add functionality to the instrument driver. Source code also makes Plug and Play instrument drivers operate cross-platform, so you can use them in any OS that works with LabVIEW or LabWindows/CVI software.

To obtain instrument drivers for NI products (for example, NI-SCPE, NI-DMM, and so on) or for third-party products (for example, Agilent 34401A digital multimeter), visit the Instrument Driver Network (ni.com/idnet). The Instrument Driver Network makes available drivers for more than 10,000 instruments for over 350 third-party vendors and for LabVIEW and LabWindows/CVI.

IVI Instrument Driver
Updating a test system can be a costly and time-consuming task. What initially seems like a simple change to a test system can lead to a considerable amount of re-development, re-verification, and re-documentation. Interchangeable Virtual Instruments (IVI) drivers, which are maintained by the IVI Foundation, are the most sophisticated instrument drivers with their key feature being multivendor instrument interchangeability. IVI provides a standard API for the eight most widely deployed instrument types, including DMMs and oscilloscopes, so you can write a test application based on an Agilent scope and program it using the IVI Scope class driver. If for some reason you need to replace the Agilent scope with a Tektronix scope, you can do so...
without changing your code. In addition, the IVI specifications offer base, extended, and instrument-specific API options; range checking; simulation; and other features that make upgrading instruments easier.

The IVI Foundation defines two architectures for IVI drivers, one based on ANSIC and the other on Microsoft Component Object Model (COM) technology. Although the technologies underlying IVI-C and IVI-COM are different, the implementation technology by itself should not be your primary concern. Instead, you should focus on two key issues: (1) the longevity of the architecture on which the instrument driver is based and (2) the usability of instrument drivers in your ADE.

**Architectural Longevity:** The issue of architectural longevity is particularly important to users of IVI drivers. Interchangeability is one of the most beneficial features of IVI, and a primary reason for achieving interchangeability is to make it easier to replace instruments in systems that must last 10 to 20 years. Having a common API for instrument drivers is not sufficient if the architecture on which those instrument drivers are based changes every few years. It is for this reason that the IVI-C architecture is preferred. ANSIC is a long-established standard that is expected to remain available on all platforms. Conversely, COM is not freely available on all platforms and has already been superseded by .NET.

**Usability:** One of the key motivations behind IVI-COM drivers is the desire to develop one driver that works automatically in all development environments. Unfortunately, this comes at the expense of usability. IVI-COM drivers present an interface style that is optimal only in Microsoft Visual Basic 6.0, which was superseded by Microsoft Visual Basic .NET. Ideally, instrument drivers should present an optimal interface for each ADE in which they are used. For example, instrument drivers should be presented as a set of LabVIEW VIs for use in LabVIEW, a set of C++ classes for use in Microsoft Visual C++, and a set of .NET classes for use in Microsoft Visual Basic.NET and Visual C#.

Although IVI-C drivers are the best choice to build an interchangeable test system that can stand the test of time, if all that is available for your instrument is an IVI-COM or VXIplug&play instrument driver, then you should use it. Your goal is to be able to easily communicate with your instruments and to not be tied to one driver technology.

Before determining if a driver technology is right for your application, you should ask yourself some of the following questions:

- Does my test system need to last longer than 10 years?
- Am I sure I want to sole-source my instrumentation?
- Will the eight IVI instrument classes meet my application requirements?
- Do I have to revalidate my entire test system when I make a change to an instrument driver (common for mil/aero/medical)?
- Am I maintaining a system that is currently using instruments that will probably be obsolete before my test system?

If possible, use IVI if it satisfies your requirements because developing a user-defined IVI Class driver is often a large investment for an organization. However, the investment of many companies and the stability of the standard over many years can reduce your need to design, develop, and maintain a driver, saving time and money.
Direct I/O

Direct I/O is a low-level method for programming instruments. Standard Commands for Programmable Instruments (SCPI), pronounced “Skippy,” is an extension of the IEEE 488.2 standard that defines a standard programming command set and syntax for Direct I/O. Engineers typically use this method only when Plug and Play or IVI instrument drivers are not available. You may also want to use Direct I/O because of the following:

- You are updating an old system that is based completely on SCPI
- You need to send only a few commands to your instrument
- You do not need to distribute a set of instrument commands to other developers
The modern Direct I/O communication standard to instruments is through the Virtual Instrument Software Architecture (VISA) API. VISA is an industry standard communication protocol managed by the IVI Foundation that provides bus- and platform-independent instrument communication. For example, the VISA commands to write an ASCII string to a message-based instrument is the same whether the instrument is serial, GPIB, or USB. Thus, VISA provides interface independence. This can make it easy to switch interfaces and gives the users who must program instruments for different interfaces a single language they can learn.

If there is not an instrument driver available, you should take advantage of the interactive, direct I/O capabilities built into the software development environments. NI ADEs offer the Instrument I/O Assistant, built-in VISA and bus-specific interfaces, and several debugging tools within Measurement & Automation Explorer (not covered in this document) including NI Spy, Interface Bus Interactive Control (IBIC – for GPIB), and VISA Interactive Control (VISAIC).