Smart Programming Preserves Equipment Value

by
Duane D. McDonough
Consultant, Measurement & Automation
Data Science Automation, Inc.
USA

and

Michael L. Porter
Architect, Measurement & Automation
Data Science Automation, Inc.
USA

Category:
Production ATE

Products Used:
LabVIEW 8.2.1

The Challenge:
Develop a mechanism to allow a technically capable – but non-networkable – instrument to perform the data collection required for SPC analysis of the customer's process, and archive the data to a database. The software had to be essentially invisible to the system user, but be accessible to administrative personnel.

The Solution:
Leverage LabVIEW's connectivity capabilities: Taking advantage of the .NET framework, Data Science Automation (DSA) reduced the user interface to an icon in the System Tray. Additionally, LabVIEW's support for ActiveX allowed the application to directly access the built-in ADO drivers to insert the collected data directly into the SPC database.

Abstract:
A manufacturer of laser toner medium had a problem: They wanted to collect analytical data related to the particle size of the toner they produced. However, lacking internal networking capabilities, the instrument making the particle measurements was incapable to passing the data it collected to the SPC analysis software needed to analyze the data. The solution was a program running in the system tray of a computer connected to the test instrument. This application reads the instrument's proprietary output files, extracts the significant data and writes it to a database that the SPC application can access.

Application Overview
The overall structure of the application is event-driven. The application's primary functionality takes place in the Timeout event which, at a user-defined rate, checks to see if there are any new results that need to be transferred to the database. In addition, the event structure implements value change events tied to front panel buttons that reinitialize or stop the application.

The basic processing takes advantage of the instrument's ability to generate output "report files" when it completes a test. The application monitors the directory to which the instrument saves its datafiles and when it finds a new one, it opens the file, extracts the pertinent data performs a few simple calculations and writes the result to the database.

375 Valley Brook Road • Suite 106 • McMurray, PA 15317 • T 724-942-6330 • F 724-942-8390 • DSAutomation.com

Sliding Into the Tray

An important distinction to bear in mind when working with the System Tray is that the application isn't running in the tray. Rather, the tray is simply providing an alternative user interface for the program that is menu driven. As far as the LabVIEW application itself is concerned, it is running with its front panel open but hidden. However, because the menus exist in the .NET environment a means is needed to communicate menu clicks back to LabVIEW. The mechanism for that communications is to register callback events that automatically run a specified VI when an event occurs.

The code for defining these callbacks (Figure 1) first builds the menu structure and then defines the callback associated with each menu item. Each event definition consists of a terminal selecting the event associated with the callback – in this case the "Click" event. The next terminal accepts a reference to the VI that will be run when the event fires. The third terminal is labeled "User Parameter". This polymorphic terminal provides a way of passing to the callback VI a piece of data. Of course this single data value can be a cluster so you can really pass as much data as desired.

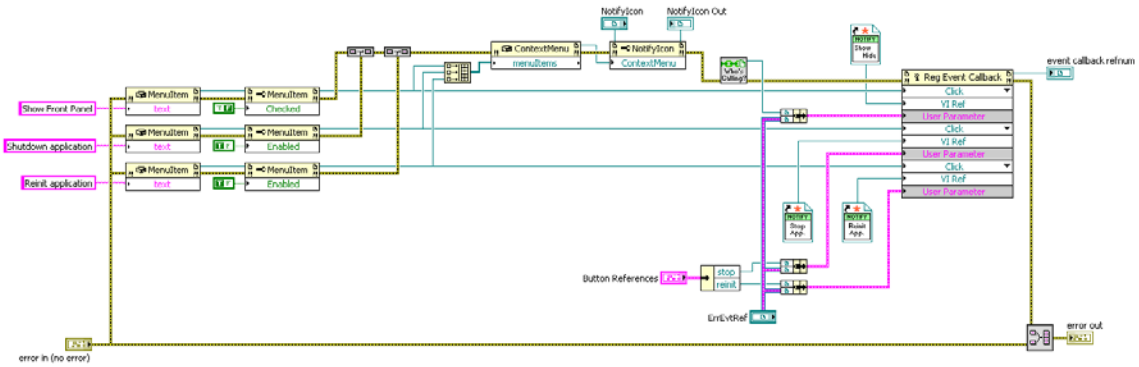

Figure 1 – Just because a callback event is a non-standard way of calling a VI, that doesn't mean it can't have appropriate error handling. Note that part of the "User Parameter" data passed to the callback registration is an event reference. The event associated with the reference is the standard error handling event for the application. Generating that event serves to pass error data from the callback VIs to the main application.

The menu for this application implements three callback events: The first event toggles the visibility of the application window by manipulating the window state ("Standard" or "Hidden"). To make this operation possible, the code passes a reference to the top-level VI to the callback VI. The second callback event will shutdown the application by setting the Val(Sgnl) property of the Stop Engine button on the application's front panel. The third callback reinitializes the application using the same technique to "press" the front panel Reinitialize button. Note that all these options are password protected.

To provide additional feedback to the user, the code updates the tip-strip that is seen when the mouse hovers over the application's icon in the tray. Every time the database is updated this value is set to reflect the time and date of the update.

The result of this structure is a LabVIEW application that exposes its critical features through a simple pop-up menu (Figure 2).

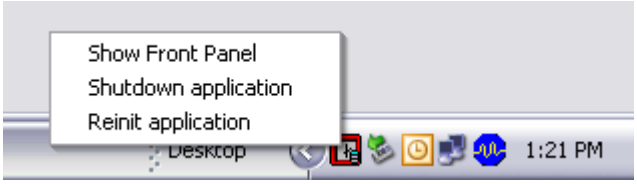

Figure 2 – This application needed to maintain low visibility. A good way of implementing that functionality is to put it in the System Tray. This move makes both the interface much less obvious, and gives the developer greater control over the program features that are exposed to the user.

Database Connectivity

To interface with the database, the code utilizes direct calls to the built-in Windows ActiveX ADO interface. This choice was made for the following reasons:

- Simplicity – The ADO interface is extremely well document and easy to use.
- Maintainability – Many sources of support exist for ADO, SQL (the language used to communicate with relational databases), and the database being used.
- Efficiency – Because the code is talking directly to the ADO interface, the overhead on the LabVIEW-side of the operation is very low.

Another aspect of the database connectivity is that it utilizes database transactions to safe-guard the relational integrity of the database. In essence a database transaction turns multiple operations into a single atomic operation. Hence, either all the operations succeed, or none of them do. This is important because it means that you don't have to deal with partial data updates.

Summation

The basic functionality demonstrated in this application holds the promise for retrofitting a wide variety of legacy applications. Whether the system generating the text files is an old instrument or an old test application, this approach offers modernized connectivity with no modification to the existing system.

375 Valley Brook Road • Suite 106 • McMurray, PA 15317 • T 724-942-6330 • F 724-942-8390 • DSAutomation.com