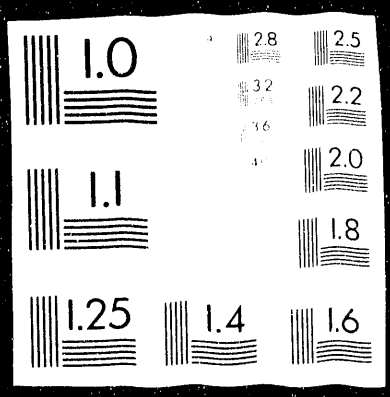


compiler that generates executable code with performance comparable to that produced by a C or

NIPER-596
Distribution Category UC-122

General-Purpose Automation Programming: A Case Study on Using a Graphic Language

NIPER--596

DE92 001071

By
S. M. Mahmood
D. K. Olsen

October 1992

Work Performed Under Cooperative Agreement No. DE-FC22-83FE60149

Prepared for
U.S. Department of Energy
Assistant Secretary for Fossil Energy

Thomas B. Reid, Project Manager
Bartlesville Project Office
P. O. Box 1398
Bartlesville, OK 74005

Prepared by
IIT Research Institute
National Institute for Petroleum and Energy Research
P. O. Box 2128
Bartlesville, OK 74005

TABLE OF CONTENTS

TABLES

ILLUSTRATIONS

GENERAL-PURPOSE AUTOMATION PROGRAMMING: A CASE STUDY ON USING A GRAPHIC LANGUAGE

By S. M. Mahmood and D. K. Olsen

ABSTRACT

Object-oriented programming is the future direction of computer programming and automation. It provides fast, easy and very reliable methodology for developing large complicated software programs to test prototype configurations and develop computer control of multiple operations. This report is part of technology transfer to assist the petroleum industry and other engineering, science and manufacturing areas by conveying background information and the benefits of this approach that were acquired during development of the data acquisition/control/analysis/presentation software to operate a high-temperature, high-pressure steamflood laboratory. NIPER developed the background and extensive applications using object-oriented programming software using National Instruments™ LabVIEW® 2 as the programming platform. The objective of developing this software was to automate the thermal lab; however, it was designed in such a way that the program would be general and could be configured to any laboratory automation during the run time. This modular program has been constructed so that application, with minor modification, can be used in other laboratory, pilot plant or commercial operations for data acquisition/control/analysis/presentation.

EXECUTIVE SUMMARY

The report addresses part of tasks 1 and 6 of research called for in NIPER's FY92 annual plan for project BE11A, Thermal Processes for Light Oil Recovery. It also provides a background and outline of the modular software developed using National Instruments™ LabVIEW® 2 as the programming platform that uses an object-oriented programming language "G." Object-oriented programming is the future direction of computer programming and automation. It provides fast, easy and very reliable methodology for developing large complicated software programs to test prototype configurations and develop computer control of multiple operations. NIPER developed this general modular program for data acquisition/control/analysis/presentation to have the following capabilities: (1) Allow user to operate interactively through a computer panel by communicating with laboratory instruments such as pumps, controllers, balances, temperature sensors, pressure sensors, fluid flow controllers, and alarms; (2) warn operators of conditions out of preprogrammed ranges and shut down operations systematically if the operator does not respond; (3) monitor the test progress by material balance based system and provide early warnings to user for situations that may need user's attention in near future, such as feed tank

running low on liquid or sudden changes in material balance indicating e.g. possible leak in the system; (4) prohibit user from making unsound operations such as running a pump when outflow valve is closed; (5) let user manipulate current and/or previous data interactively, e.g. analyze, compare, plot, curve-fit and print; and (6) communicate with other applications and users for sending data and warnings and receiving instructions.

This modular program, developed by NIPER using LabVIEW 2 has been constructed so that application, with minor modification, can be used in other laboratory or pilot plant operations for data acquisition/control/analysis/presentation. The software permits analysis and presentation within LabVIEW 2 and other software programs written for a Macintosh computer. Some of its features and typical control instruments are listed in Appendix A.

This report along with a companion report[1] completes milestones 1 and 6 of research called for in NIPER's FY92 annual plan for project BE11A, Thermal Processes for Light Oil Recovery. The companion report[1] covers the configuration of equipment and operators' guide for use of equipment and laboratory control systems. The combination of both reports helps NIPER to document some of the operational aspects and procedures to improve and maintain safe, efficient operating procedures for construction and operation of a high-temperature, high-pressure steamflood laboratory.

The software developed is part of a broader task to develop procedures and apparatus for measuring dynamic saturation changes in laboratory steamfloods using an X-ray CT scanner that incorporates temperature, pressure and electrical conductivity measurements as part of the monitoring of steamflood performance. Results from these laboratory steamflood experiments would then be used to calibrate numerical simulators for predictive purposes for support of light oil steamfloods conducted in the field.

OBJECTIVE AND SCOPE OF REPORT

The objective of this report is to convey a background and the benefits of using object-oriented programming as acquired during development of the data acquisition/control/analysis/presentation software to operate a high-temperature, high-pressure steamflood laboratory. This report is an overview of what has been developed and provides a background so that the readers can not only appreciate the simplicity of the programming platform but also follow the discussion and judge the applicability of this software to their specific application. The background demonstrates the simplicity of writing, testing, debugging, and maintaining large programs when using the graphical interface language (LabVIEW 2) compared to traditional line-code programming, such as FORTRAN, with which the reader may be more familiar. Discussion on the concepts of object-oriented programming is included as well as comparison of a simple FORTRAN program because it is necessary to show by example a program written in both

LabVIEW 2 environment and a traditional line-code language. However, such discussion is only illustrative and is not intended as a tutorial or to be exhaustive. References are provided at the end of the report where additional information and independent reviews of object-oriented programming and LabVIEW 2 can be found.

BACKGROUND

NIPER has operated a steamflood research laboratory since 1984. Since that time, it has used two physical models as principal tools to investigate various aspects of steam displacement of both heavy and light crude oil. In 1990, the entire steamflood laboratory including both one-dimensional (1-D) steamflood models and a two-dimensional (2-D) flat plate steamflood models were reviewed. In light of previous laboratory operating problems, and their limitations in modeling field-scale steamfloods as pointed out in a 1991 study of scaling parameters of steamflood physical models,[5] a new titanium steamflood model, which is X-ray CT scannable, yet has large-volume (25L) capacity, was designed and is being constructed. The configuration of that equipment, an operators' guide for use of equipment and laboratory control systems, and plumbing and electrical schematics with specifications are being compiled.[1]

Due to the safety and reliability consideration involved in the operation of a high-temperature, high-pressure steamflood laboratory, a decision was made to adopt National Instrument's™ LabVIEW 2 as the programming platform because of its highly structured object-oriented programming approach and operate the system from a Macintosh II computer.[6] The main components of the steamflood laboratory are the steamflood physical models, the laboratory configuration, NIPER's automation software, and a Macintosh II computer installed with several digital and analog data transfer boards.

PROGRAMMING CONCEPTS RELEVANT TO LabVIEW 2

This section describes the concepts of programming languages that are pertinent to LabVIEW 2. In light of these concepts, LabVIEW 2 will be compared (in later sections) with traditional languages. The following discussion is aimed to highlight the basic differences in the conventional languages and LabVIEW 2.

To refresh reader's memories, let us first consider a simple program that iteratively reads data from a file, converts it to degrees Fahrenheit, and warns user if the value exceeds a predetermined limit. Fig. 1 lists a FORTRAN program to achieve this objective. To help understand the structure of the program, a control-flow diagram is customarily used, such as the one shown in Fig. 2.

```
      OPEN (5, FILE = 'ALLIANCE:LABVIEW 2:INPUT TEMPERATURE',
     +  STATUS = 'OLD') CELSIUS
C     ALLIANCE IS THE VOLUME I.E., HARD DISK, LABVIEW2 IS THE
C     FOLDER CONTAINING FILE NAMED INPUT TEMPERATURE
   10 READ (5, *, END = 99)
      FAHRENHEIT = ((CELSIUS * 9/5) + 32)
      IF (FAHRENHEIT.GT.450) GOTO 20
      GOTO 10
   20 CALL BEEP (1)
C     BEEP IS A SUBROUTINE THAT ACTIVATES AN AUDIBLE BEEP
   99 STOP
      END
```

FIGURE 1. - A FORTRAN program that iteratively reads data from a file, converts it to °F, and beeps if the value is higher than 450° F.

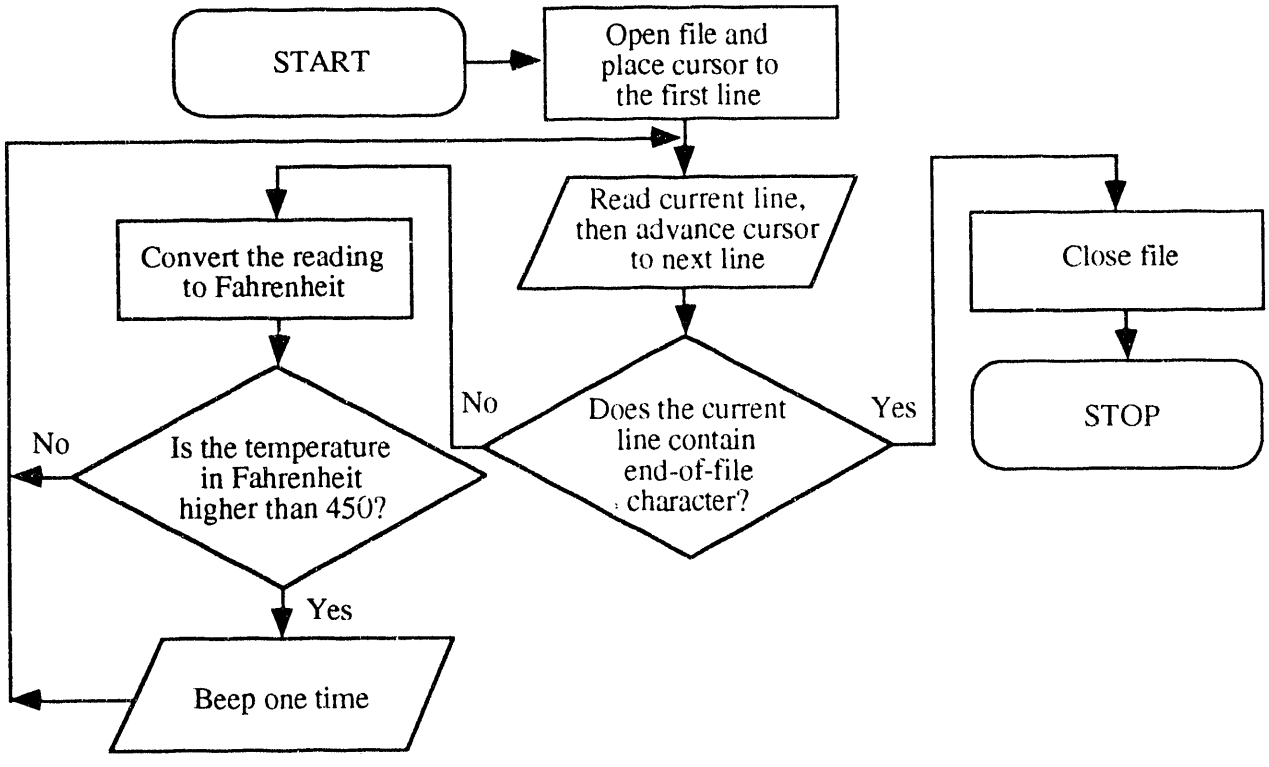


FIGURE 2. - The control-flow diagram of the example FORTRAN program (shown in Figure 1) that iteratively reads data from a file, converts it to °F, and beeps if the value is higher than 450° F.

Now, let us consider a program in LabVIEW 2 (Fig. 3) which achieves the same objective as the FORTRAN program. Notice that the LabVIEW 2 program itself is in the form of a flow diagram, thus no control flow diagram is needed. Further explanation of this program is provided in Appendix B.

At the first glance, it may appear that the FORTRAN program is easier to follow. It may very well be true for small programs like the one presented in this example. However, our experience in writing and maintaining software for the thermal lab using the BASIC language and then LabVIEW 2 shows that as the size and the complexity of the program grows, line-code programs increasingly become more difficult to follow than visual programs.

Object-Oriented Programming (OOP)

The important concepts of object-oriented programming (OOP) are described below. This description is included to help readers visualize the significant advantages that this approach to programming offers over languages such as Basic, FORTRAN, Pascal, and C, which are some of the more familiar line code languages. National Instrument's LabVIEW 2, the basis of NIPER's automation software, is a programming platform (compiler) that uses a high-level object-oriented language called "G."

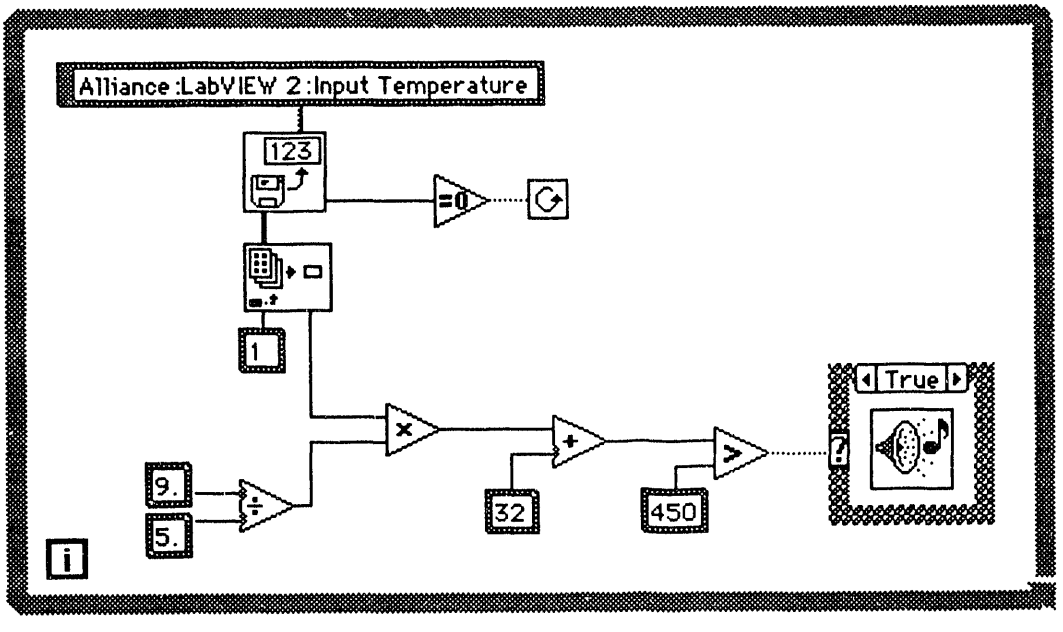


FIGURE 3. - A sample program segment in LabVIEW 2 that iteratively reads data from a file, converts it to °F, and beeps if the value is higher than 450° F. For description of this program, see Appendix B.

OOP refers to a programming style that relies on the concepts of inheritance and data encapsulation. Inheritance is a language facility for defining a new class of objects as an extension of previously defined classes. The new class inherits the variables and operations of the previous classes. Inheritance helps in building complex structures by using the existing simpler objects. Since common properties of objects can be preprogrammed by defining classes, programming effort can be significantly reduced. Data encapsulation (also called implementation hiding, meaning certain details of implementation code are deliberately hidden from the user) allows objects to be packaged so that unnecessary details of implementation are not visible from outside the object. Examples of inheritance and data encapsulation are shown on the following page. An object may include a set of functions, procedures, subroutines, data, type-definitions, arithmetic and/or other operations. Any or all entries in an object may be defined as either public, private, or protected, depending upon their intended use. Objects interact with each other by sending and receiving messages.

The properties of OOP allow one module (an independent program segment like subroutines) to be written with little knowledge of the code in another module. Modules can be reassembled and replaced without reassembly of the whole system. OOP's programming style can be practiced with widely differing languages. For example, C++ (a line code language) allows both inheritance and data encapsulation to deal with the most demanding systems' tasks yet retains C (also a line code language) as a subset for tasks requiring low-level programming.[7] LabVIEW 2 superimposes a graphical editing and execution system upon the object-oriented "G" language to create a platform for the users wherein programs (modules) can be easily written, tested, debugged and modified by copying the objects from the LabVIEW 2 library and user's own library of modules. Thus, it provides a simple yet powerful visual programming environment.

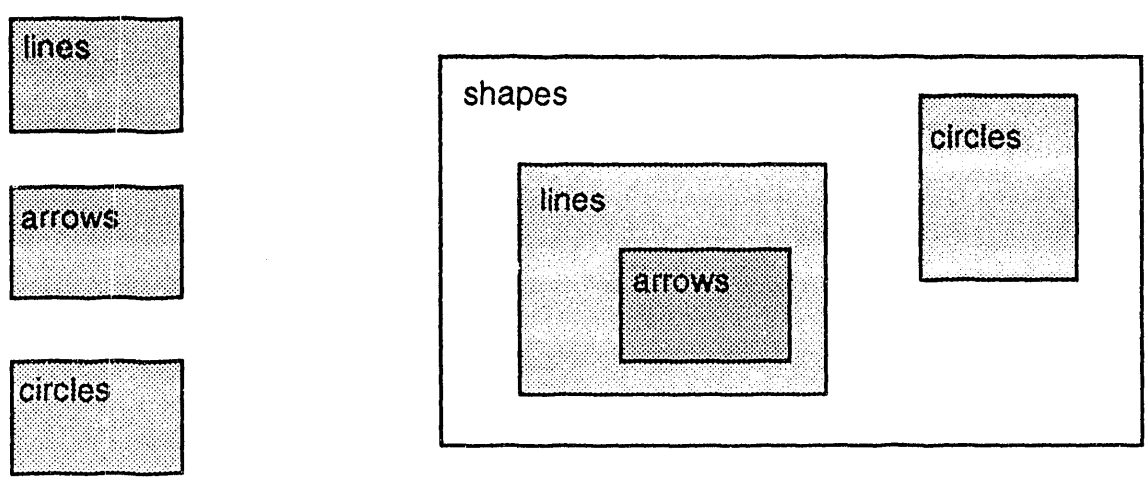
Procedure-Oriented Programming (POP) Versus Object-Oriented Programming (OOP)

Figure 4 illustrates the difference in procedure-oriented programming (POP) and OOP.[8] The requirement in this example is to build figures, which are basically composed of basic shapes—lines, rectangles and circles. POP will structure a program around the operations on shapes. It may include operations for drawing, rotating, and scaling a figure. For each shape in the figure, the procedure will classify shape and then execute the code that is appropriate for drawing that kind of shape. However, the code for each shape will contain only very elementary operations. Because of the use of elementary operations, the code for manipulating shapes is spread across the various procedures, which is often problematic. If a new shape is added, e.g. an arrow head, then the code for handling the new shape has to be added to each procedure. Even if the new

information is small, it is spread across procedures, each of which must be analyzed before the new code is added to ensure that there are no conflicting directions or assignments.

The OOP approach of handling this problem is: A class named "shape" is defined, which has subclasses of lines, rectangles, and circles. Class shape then collects common properties, such as the height, the width, the position, and an operation for moving the shape. Properties that are specific to lines, rectangles, and circles appear in the appropriate subclasses. Inheritance, an OOP property, allows arrows to be added by extending the subclass "line", without touching the code for the other objects. An arrow inherits all the properties of a line, so the only additional code needed to draw an arrow will be the code for drawing an arrowhead. Another property of OOP—data encapsulation—will allow the drawing of various shapes by simply sending messages to the class shape, without the need to see its implementation details. In OOP, each module is a completely executable program in its entirety, and it does not interact with other program segments in any way except by receiving and sending messages. This message-passing mechanism is superior to the use of "subroutine calls" in traditional programming because it eliminates any chances of inadvertently altering the data in the calling program. The user does not need to know the implementation details of a module to use it in any other module. He merely needs to know the abstract information about its actions and about the input/output data to be exchanged through messages.

Since OOP is more of a philosophy than a specific language, differentiating OOP with precision from other programming styles is difficult. In abstract terms, OOP relies heavily on making and using objects (building blocks) instead of using elementary units and operations. If the objects are appropriately defined, the task of manipulating them becomes easier. The objects



(a) Disjoint sets of objects (b) Nested sets of objects

FIGURE 4. - Different ways of classification of objects: (a) POP (b) OOP.

are treated as complete units; hence, operations on them are far less likely to have inadvertent side effects, i.e., unwanted influences on other objects or program are avoided. The power of OOP becomes increasingly apparent as the size and complexity of a program increases. It is easier and safer in OOP to extend the classes, and the code for each individual operation is often small and seems to simply "pass the buck" by invoking operations in other objects.[8] Debugging or editing time is thus significantly reduced for the programmer.

Program Testing

For automation programs (laboratory, pilot plant or production facilities) that deal with instrument control, it is imperative that the correctness of the program be established with reasonable certainty. Because of the program's size and complexity, such programs can seldom be guaranteed to be "absolutely error-free." It is "hopeless to establish the correctness of a program by testing, unless the internal structure of the program is taken into account," argues the noted computer scientist Dijkstra.[9] Programmers wish to design programs carefully so that the correctness can be understood in terms of structure. In his words, "the art of programming is the art of organizing complexity." Large programs are difficult to test and debug because the effect of a change can propagate into various segments, some of them remaining undetected for an extended time. Structure and organization are the key to managing large programs. The readability of a program can be improved by organizing it so that each part can be understood relatively independently of the other parts.

LabVIEW 2 using the object-oriented programming language "G" provides several constructs for organizing computations. It helps the programmer to write good programs, which are easy to read (follow), easy to understand, and most importantly, easy to modify. LabVIEW 2 structured program organization enables the user to package the same amount of information into far fewer symbols. Its superb abstraction mechanism puts the needed information on user's fingertips, allows the user to easily examine the program in various levels of details, and makes the task of remembering easier. The program is divided into isolated program fragments (modules). These modules can be understood, improved, and changed without concern for how these changes will affect the main program because they interact with other modules and elements only through receiving and sending messages (i.e., never access global variables).

DESCRIPTION OF LabVIEW 2

Brief Introduction of LabVIEW 2

LabVIEW 2 is a visual programming environment that can be used effectively by a broad range of people with different programming skill-levels. The two-dimensional graphical notations that LabVIEW 2 uses are much easier to comprehend than the textual notations such as line codes.

The programs look like a dataflow diagram (Fig. 5), in which elements are pictorially represented and dataflow between elements is through color-coded wire connections. With this representation, constructing or understanding a program is easy and natural.[11] Kodosky, MacCrisken, and Rymar[12] have described some of the programming structure behind LabVIEW 2, and we have used a number of their examples because of the clarity of their presentation. The first section provides more detail about the language structure, whereas the later section describes the basic facilities in LabVIEW 2.

Programming Structure of LabVIEW 2

National Instruments LabVIEW 2, by using the high-level, object-oriented language "G," allows LabVIEW programs to be highly structured and modular. Each module is a totally independent and interactively executable program, which can be used as a subprogram by other modules. To use a module as a subprogram, its icon is copied into the program. Data exchange (input/output) with the module can be accomplished by making wired connections between the terminals shown on the icon and other elements in the program. In this fashion, large programs can be developed in an hierarchical manner by starting with small modules and using them within other modules. Since each module has its own independent program and a separate input/output interface, debugging or modifying a large program developed with LabVIEW 2 is quite easy.

LabVIEW 2 handles the execution sequence of a program in a different manner than line-code languages. In the traditional languages such as Basic, Fortran, C, or Pascal, the execution of instructions takes place according to the control flow diagram designed by the programmer. LabVIEW 2, on the other hand, is based on a modified dataflow model such that the sequence of execution need not be predefined.

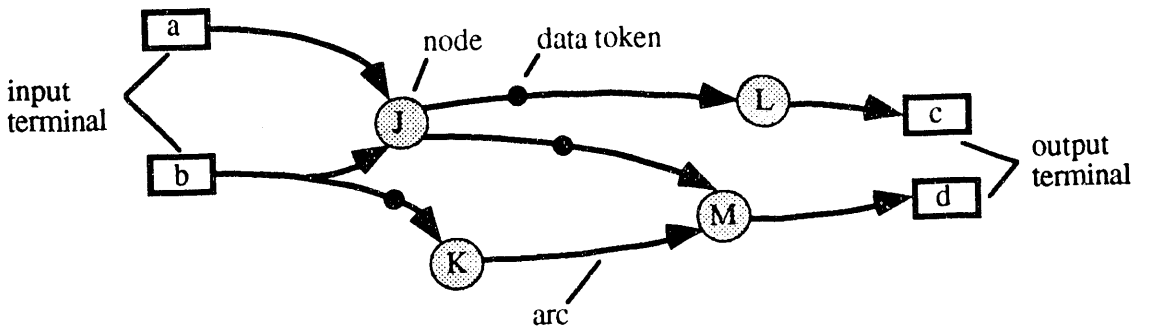


FIGURE 5. - A Simple Dataflow Diagram.

Dataflow diagrams specify the data dependency between computations, but they do not specifically force any particular sequence of independent computations. A simple pure, data-driven dataflow diagram is illustrated in Fig. 5. It is a directed, acyclic graph consisting of nodes, arcs, terminals, and data tokens. Terminals in the program are the connections to the external world, and act as the sources or sinks of data tokens. Arcs are the directed paths over which data tokens move, and nodes are the locations in which computations are performed. The fan-out of an arc implies copying the data token; the fan-in of an arc is disallowed. A node consumes tokens on its input arcs and produces new tokens on its output arcs. What makes the diagram data-driven is the *firing rule*, which states that a node cannot execute until all of its input arcs have a data token available, at which time the node consumes one token from each input arc, performs the computation, and produces one token for each output arc. In Fig. 5, node J has already executed, K and L are eligible to execute, and M is still ineligible because it needs a token on its second input.

In contrast to the control-flow model, the dataflow model has no concept of locus-of-control, no program counter (i.e. no sequence numbers like in text-based program codes), and no global variables (globally accessible memory). A data token exists only from its production by a node or input terminal to its consumption by another node or output terminal. All nodes that are eligible to execute can do so in any order or even in parallel; the results of the diagram will be the same in all cases.

The classical dataflow model, however, lacks the provisions for conditional or iterative computations. Several extensions of dataflow model have been proposed by relaxing the firing rules and allowing cycles in the graph—the modifications that severely compromise the clarity of the programs. LabVIEW 2 provides an extension to overcome this limitation which not only preserves its firing rules and acyclic structure (thus preserving program clarity) but also incorporates the proven benefits of the structured programming methodology. This extension involves redefining a node to be any program segment enclosed in a box-like structure that separates the body (or inside) of the structure from the rest of the program. Because the structure behaves like a node as far as the rest of the program is concerned, the overall dataflow methodology is preserved. The body of a node (inside the structure) behaves like an isolated diagram, in which access to the code is only from the top (or beginning). The program structure-semantics such as loop behavior or conditional behavior have been superimposed on the body of the structure. This can be thought of as a macrostructure (program as a whole) containing some microstructures (program segments inside the node), both of which independently follow the dataflow model. The microstructures, however, have some additional control properties.

Using the extended dataflow strategy, LabVIEW 2 is able to retain the important benefits of both structured programming and dataflow strategy. In addition, LabVIEW 2 incorporates a

compiler that generates executable code with performance comparable to that produced by a C or Pascal compiler.

Basic Facilities in LabVIEW 2

(I) Editing:

LabVIEW 2 contains three interrelated editors, one for each of the three parts of a module, or virtual instrument (also abbreviated as VI) as National Instrument calls it. These three editors consist of the following: the block diagram, the front panel, and the icon. A block diagram is a directed acyclic graph containing nodes, interconnecting signals, and source and sink terminals which correspond to the front panel controls and indicators, respectively. It is constructed by selecting built-in functions, structures, and previously constructed VIs from graphical palettes and arranging them in the block diagram window. The VIs from the user's own library can also be copied. The front panel contains all of the input/output controls and indicators for interactive programming. These controls and indicators define the data types of the inputs and outputs of the VI. The controls and indicators are initialized with the previously stored values, but the values can be reassigned by the user during and prior to the program execution. The icon of each VI contains terminals (non-overlapping subregions) that are in one-to-one correspondence with a subset of the panel controls and indicators. These terminals can be used to import/export data programmatically.

Inside the block diagram, the arcs (or connecting wires) are drawn using the Wiring tool to establish the paths of data exchange. The wiring tool is a cursor that looks like a spool of wire. As each edit transaction is performed, the syntax checker detects and flags any cycles introduced into the dataflow graph, propogates data attributes (type) to all the terminals, computes the data type for each built-in function, and redraws any arcs whose attributes have changed. Each arc is drawn with a distinctive pattern, width, and color code to indicate the data type, array dimensionality, and numeric representation.

(II) Built-in Library:

LabVIEW 2 comes with a large library of VIs (modules) for easy handling of most low-level programming details. Use of these VIs allows the programmer to concentrate on customizing the program. Numerous driver programs for common instruments are also included, and many are available from third-party vendors. For systems involving several instruments or for programs requiring specialized communication with the instruments, the user may have to write his own driver program. Depending upon the complexity of the system and the level of programmer's experience with LabVIEW 2, such programming may take several days. The time required for programming will be significantly lower than the conventional line code languages.

(III) Control Structures:

LabVIEW 2 provides five box-like structures and one file-linking structure as shown in Figure 6. Various elements in these structures are explained in Appendix C. The top three are quite similar to the "while loop", the "for loop", and the "case structure" in other programming languages. The Sequence structure executes one frame at a time in the numeric sequence of the frames, where the values can be passed from any frame to all the following frames in the sequencer. This is accomplished using local registers (arrows inside of the box). The sequence structure, as the name implies, is used to impose an order of execution. The same is true for the Formula Node, in which text-like expressions are evaluated from a top to bottom sequence. The values to the variable are assigned or evaluated through the input/output terminals, e.g. x, y, and z as shown in Figure 6(E). The Code Interface Node (CIN) is equivalent to calling an executable subroutine that is accessible to the program yet is outside the program body itself. The subroutine is imported and evaluated using the input parameters provided through the input terminals on the node (arrows in the left boxes). The result is then copied to the output terminals (arrows in the right boxes) where it is available for other program elements. The CIN allows the user to use C, Pascal, or an assembly code language, while still enabling him to benefit from object-oriented programming; but most importantly, it allows more efficient dynamic memory allocation for arrays and strings that minimizes memory fragmentation.[13]

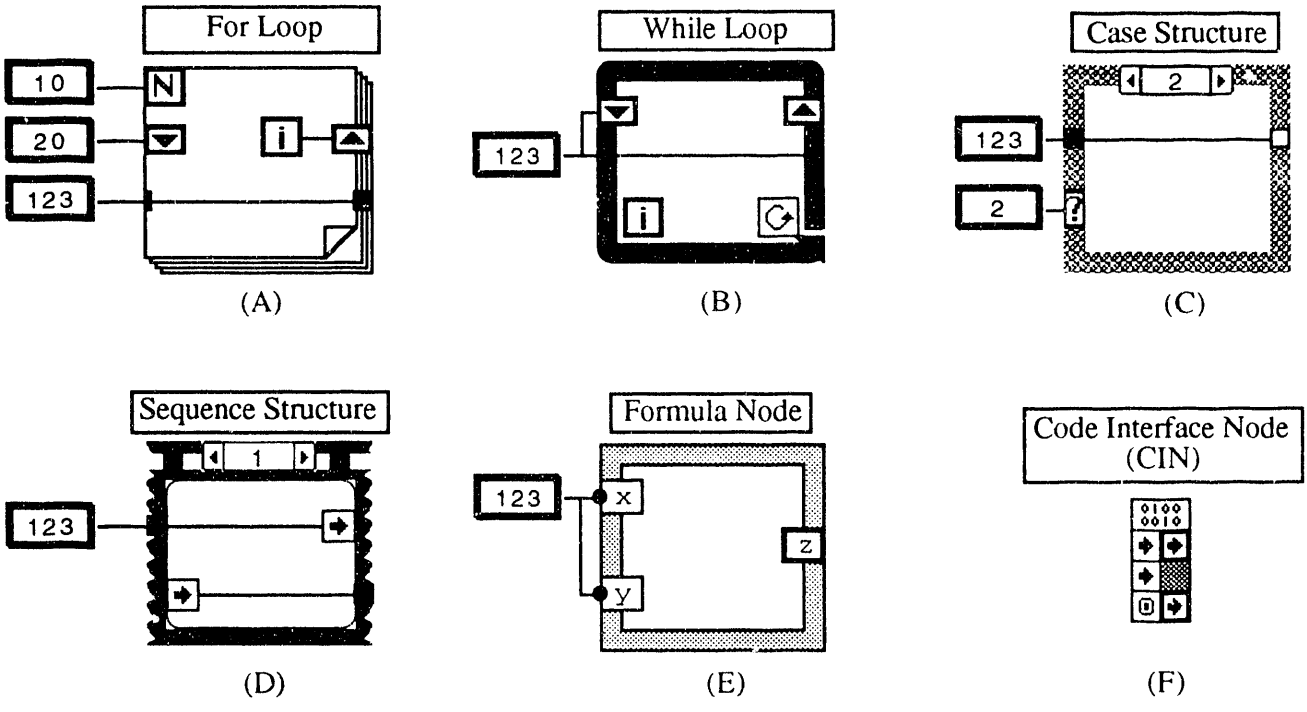


FIGURE 6. - Basic structures in LabVIEW 2.

(IV) On-line Help

Information about any of the SubVIs can be conveniently obtained by selecting Show Help from the Windows menu and then passing the Wiring tool over them. The type of information available on-line includes the name of a module (VI or function), a brief description of its intended use, and a brief visual/textual description of input/output data and terminal locations. For more details, both the front panel and the block diagram can be viewed by double clicking on a module. The entire hierarchy of a module (or a part of it) can also be conveniently viewed. An example of this is shown in Appendix D (Fig. D.1).

DESCRIPTION OF NIPER'S AUTOMATION SOFTWARE

NIPER's automation software consists of the following three main interactive sections:

A. Data Acquisition

B. Instrument Control

C. Interactive Graphics

All three of these facilities are fully integrated, such that while the user interacts with one, the others continue to process at the background. Facilities can be alternated manually or automatically on the basis of event priority. For example, the data acquisition panel may be activated during the actual scanning of data to display the acquired data and report any errors, then the interactive graphics can be called up to display the data graphically with various orientations and styles. Finally, the control panel may be activated to give a visual picture of the process. Events may be assigned priorities so that higher priority messages can automatically be switched to the interactive panel. Following is a brief description of each facility.

Data Acquisition

The control panel for this facility is shown in Fig. 7. This facility has three features: (1) let user manipulate current and/or previous data interactively, e.g. analyze, compare, plot, curve-fit and print; (2) warn user for system errors and out-of-range measured values; and (3) allow user to reconfigure system set-up; e.g. connect or disconnect instruments, define allowable ranges outside of which warning is issued and system is shut down, select interval between scans, and emergency shutdown sequence, etc. The title bar contains nine buttons which allow user to select between features. By pointing and clicking the buttons in the title bar (see Fig. 7), user may switch between these features. Some of these buttons open a new window for allowing user to make selections. Since these subwindows have lower priority, they do not interfere with other activities, i.e., data acquisition and control functions continue in the background. A brief description of various controls and displays is given in Appendix E.

Front Panel

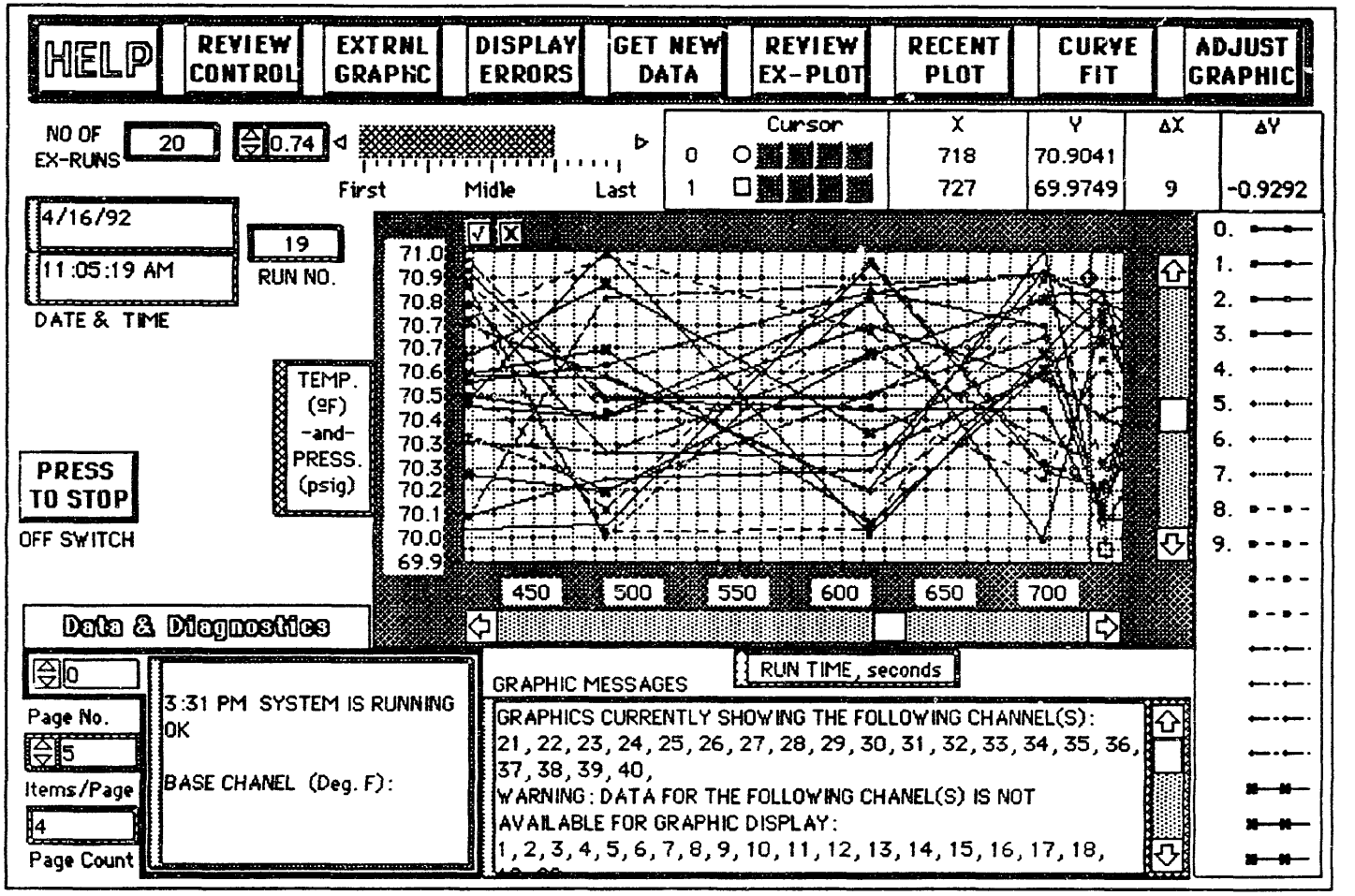


FIGURE 7. - Front panel of NIPER's data acquisition module.

Figure 8 is the flow diagram showing an outline of the functionality of data acquisition facility. It is a simplified diagram because customary flow diagrams cannot easily depict LabVIEW 2 programs, which are data-driven yet incorporate control features. The hierarchical structure of the data-acquisition program modules along with a brief description of each module is given in Appendix D.

Instrument Control

The front panel for this facility is shown in Fig. 9. A brief description of various controls and indicators is given in Appendix F. The process control facility allows the user to operate the remote-sensing instruments such as pumps, controllers, balances, temperature sensors, pressure sensors, fluid flow controllers, and alarms through the front panel. It also provides several safety features (described in the following paragraphs) to minimize common operator mistakes. The instruments are pictorially represented by indicators (see figure) which change their color or shape as the conditions change. For example, the level of fluid in a tank's indicator increases or decreases with the change of fluid weight in actual tank. This combination of pictorial and dynamic representation gives a visual sense of the process which makes it easier and faster to trouble-shoot the problems.

The facility has a built-in logic that checks the validity of user's commands and disallows them when unsound; e.g. it would not allow the user to run a pump if the outflow valve is closed; thus reducing the chances of avoidable accidents and failures. Another safety feature built into the facility is the continuous monitoring of the test progress by material balance, i.e. the injected fluids and discharged fluids are continuously weighed and compared. If a discrepancy beyond the user's acceptable limits is found, user is warned of possible leaks or other mishaps in the system. Similarly, the program continuously monitors the rate of change of preselected parameters, and cautions the user when user-defined tolerance is exceeded. Thus, a leak in the tubing causing sudden reduction in pressure will be reported immediately and a corrective action will be taken by the computer if not overriden by an operator.

This facility also warns operators when a measured value falls out of its preprogrammed range. This feature can be used to provide early warnings for situations that may need user's attention in near future, e.g. to get an alert message when a feed tank starts running low on liquid. The most useful feature of this facility is to act as a warden when user's instructions are not available. When a situation falls outside the user-defined ranges, and no response from the user is received in due time, the facility shuts down operations systematically as per user's pre-set instructions.

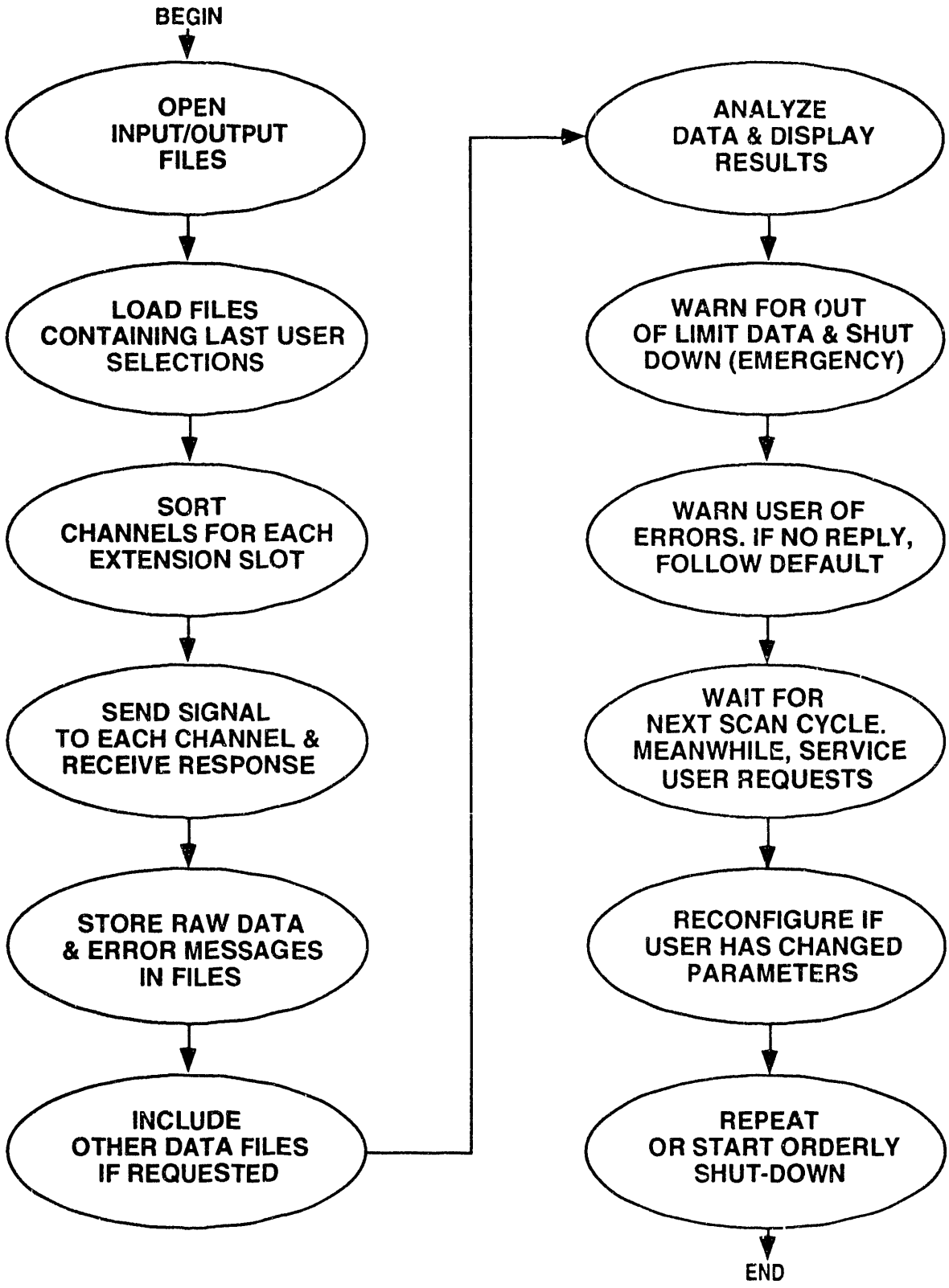


FIGURE 8. - A simplified flow diagram of data acquisition facility.

Front Panel

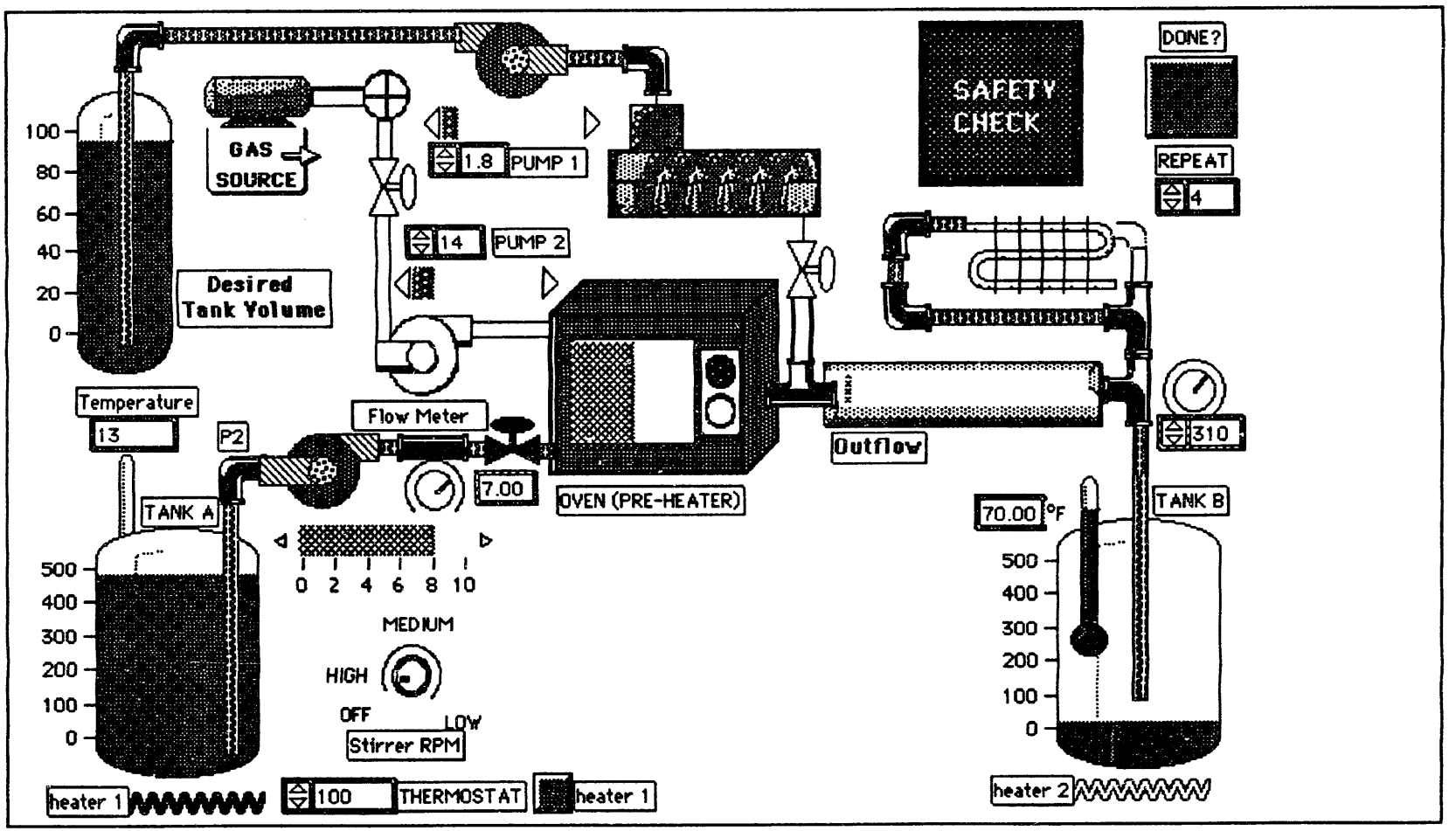


FIGURE 9. - Front panel of NIPER's instrument control module.

Interactive Graphics

The 3-D visualization of data as shown in Fig. 10 is an example of LabVIEW's data export and communication capability with other applications. In this example, every time a new data set was scanned during the lab experiment, the graphic facility in NIPER's automation software automatically opened a Microsoft Excel file containing this chart. The chart was updated with new data. The chart was then automatically rotated with different attributes such as at different angles, aspect ratios, etc. Then, the chart was closed and the control was transferred to LabVIEW 2. All the while, LabVIEW 2 continued to operate in the background acquiring (or waiting for) new data and checking for error and safety messages.

Excel is one of many applications that can be integrated with LabVIEW 2. In fact, any application that supports the Microsoft System 7's feature of Dynamic Data Exchange (DDE) can be integrated, i.e. the applications can link, subscribe, and publish to or from other files. Also, any application that allows macro-expansion (sometimes called scripting) is also a good candidate

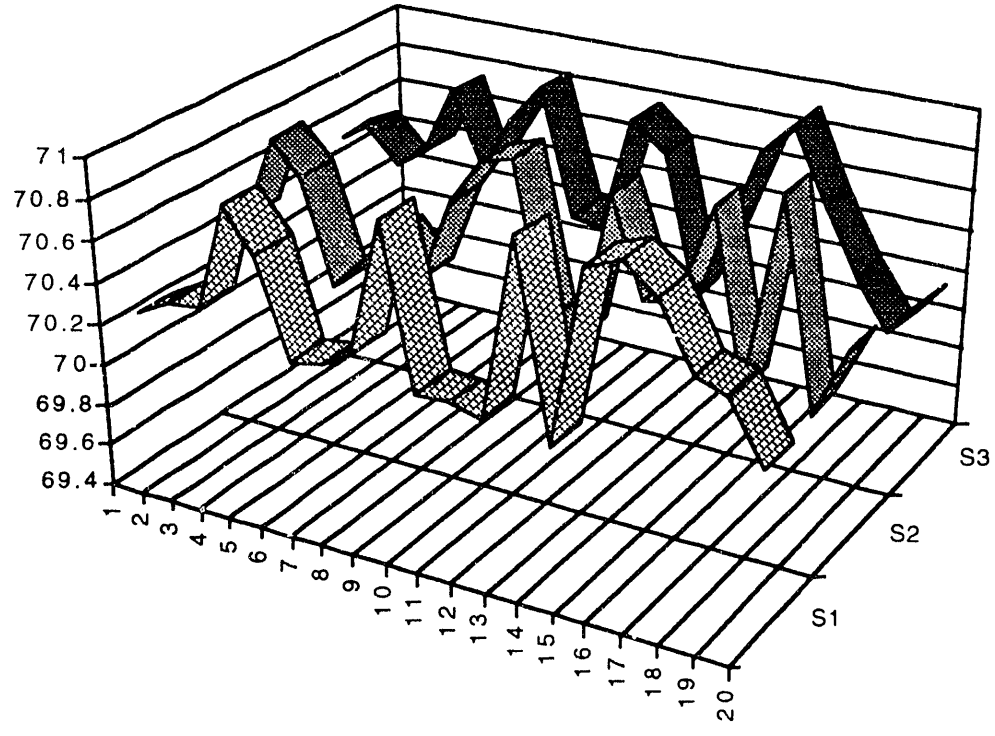


FIGURE 10. - A sample snapshot of automatic 3-D visualization of data in Microsoft Excel. DeltaGraph Professional and Spyglass Transform can also be similarly used.

for integration. The commercial applications that definitely appear to have these capabilities are Excel™, DeltaGraph Professional™, and Spyglass Transform™. The Spyglass Transform can display diffused-color surreal graphics by filling in interpolated data points along with the actual data in a multidimensional spatial field.

SUMMARY

A laboratory automation software has been developed using National Instruments LabVIEW 2, an object-oriented visual programming environment. LabVIEW 2 has shown to be an effective platform for developing comprehensive and integrated programs to acquire data, control instruments, perform extensive analysis, and display data; all in real-time and with enhanced graphics that may use other applications as well. Such automation programs can increase the reliability, efficiency and safety of lab operations because the operator can visualize the process as it happens, listen to the warnings and alarms by the system, and immediately control the process; all without exposing himself to everyday risks involved with lab experiments. Even more important is the computer's ability to act as a backup warden handling emergency situations when operator fails to respond. It is also very useful in situations where experiments have to be alternated, since the configurations are stored and need not be reset.

ACKNOWLEDGMENTS

This work was sponsored by the U.S. Department of Energy under cooperative agreement DE-FC22-83FE60149. The help in revising manuscript by G. Sharma, W. Lucas and Y. Tyagi is acknowledged. The authors thank E. B. Ramzel, G. Sharma, W. Lucas, M. K. Tham and A. Strycker of NIPER; T. B. Reid of the DOE Bartlesville Project Office for their critical reviews; and the staff of National Instruments for their encouragement and review.

REFERENCES

1. Olsen, D. K., S. M. Mahmood, P. S. Sarathi and E. B. Ramzel. *Operating Guide and Specifications for NIPER Steamflood Laboratory*, **in review**, July 1992.

2. Kirkman, I. W. and P. A. Buksh. *Data Acquisition and Control Using National Instruments' "LabVIEW" Software.* Rev. Sci. Instrum., Vol. 63, No. 1, Jan. 1992, pp 869-872.

3. Liles, Ken. *Data Acquisition Software Automates Automotive Fuel Injector Test Facility.* Engineering & Management, Oct./Nov. 1991, pp 18-21.

4. Liles, Ken. *Data Acquisition—A Mac-Based System for Fuel and Lubricant Testing.* Scientific Computing & Automation, Jan. 1992, pp 19-23.

5. Olsen, D. K., P. S. Sarathi, S. M. Mahmood, and E. B. Ramzel. *Thermal Processes for Light Oil Recovery*, Dept. of Energy Report No. NIPER-515, December 1990, pp. 18-20.

6. Apple Computer Inc. *Macintosh II User Manual*, Cupertino, CA, 1986.

7. Stroustrup, B. *The C++ Programming Language*, Addison-Wesley, Reading, MA., 1986.

8. Sethi, Ravi. *Programming Languages: Concepts and Constructs*, AT&T Bell Laboratories, Murray Hill, New Jersey, Addison-Wesley, Reading, MA., 1989.

9. Dijkstra, E. W. Notes on Structured Programming. Contained in Reference 10, pp 1-82.

10. Dahl, O.J., E. W. Dijkstra and C. A. R. Hoare *Structured Programming*. Academic Press, London, 1972.

11. National Instruments Corp. *LabVIEW 2 Getting Started Manual*, Part No. 320246-01, Austin, TX, April 1991.

12. Kodosky, J., J. MacCrisken and G. Rymar. *Visual Programming Using Structured Data Flow*. Proceedings of the 1991 IEEE (Institute of Electrical and Electronics Engineers) Workshop on Visual Languages, Kobe, Japan, Oct. 8-11 1991. Reprinted by IEEE Computer Society, 10662 Los Vaqueros Circle, P.O. Box 3014, Los Alamitos, CA 90720-1264.

13. National Instruments Corp. *LabVIEW 2 User Manual*, Part No. 320244-01, Austin, TX, Sept. 1991.

APPENDIX A

FACILITY FEATURES AND TYPICAL EQUIPMENT CONTROLLED BY THE SOFTWARE

FEATURES

MACINTOSH BASED, USER-FRIENDLY SOFTWARE
DEVELOPED USING NATIONAL INSTRUMENTS LabVIEW 2
(OBJECT-ORIENTED, "G" LANGUAGE)

CAN BE USED WITH MOST EXPERIMENTAL SET-UP
WITHOUT ANY MODIFICATION DUE TO THE
GENERAL-PURPOSE, MODULAR DESIGN

REAL-TIME DISPLAY OF DATA, GRAPHICS, CONTROLS,
ERRORS and STATISTICS

COMPREHENSIVE ON-LINE DATA ANALYSIS INCLUDING
LINEAR & NON-LINEAR CURVE-FITS & STATISTICS

ERROR WARNING & RECOVERY SYSTEMS WITH
USER-DEFINED ACTIONS OR USING DEFAULTS

BOTH DIGITAL & ANALOG CONTROL OF INSTRUMENTS
(VIA RS232/IEEE-488, ELECTRICAL PULSE & WAVES)

DATA ACQUISITION, ANALYSIS, DISPLAY AND CONTROL
OF INSTRUMENTS ARE ALL RUN-TIME ADJUSTABLE

DYNAMIC MEMORY ALLOCATION TO OPTIMIZE MEMORY
REQUIREMENT AND INCREASE EFFICIENCY

ARTIFICIAL DATA DEPENDENCY & NON-INTERRUPT
INSTRUMENT DRIVERS TO MINIMIZE DEADLOCKS

TYPICAL EQUIPMENT CONTROLLED BY THIS SOFTWARE

BACKPRESSURE REGULATOR
THERMOCOUPLES
PRESSURE TRANSDUCERS
SWITCHING VALVES
ELECTRONIC BALANCES
PUMPS
BOILER
ALARMS AND SHUTDOWN
LCR METER (ELECTRICAL CONDUCTIVITY)
RELAYS
FLUID LEVEL SENSORS

APPENDIX B

A SAMPLE OF LABVIEW 2 PROGRAM

A sample LabVIEW 2 program is explained here. This is the same program shown in Fig. 3 and briefly mentioned earlier in the text. The legend for Fig. B.1 (numbers in circles) is presented in Table B.1.

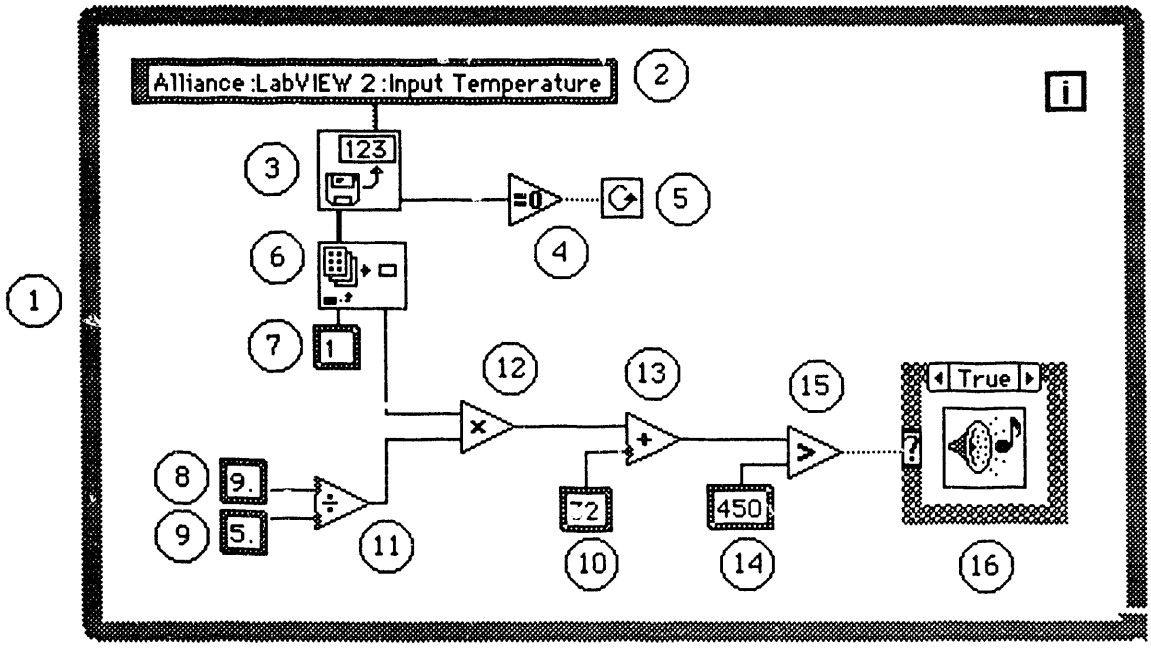


FIGURE B.1 - An example program segment in LabVIEW 2 that iteratively reads data from a file, converts it to °F, and beeps if the value is higher than 450° F. For description of this program, see Appendix B.

TABLE B.1
LEGEND FOR FIGURE B.1

1.	This entire box is a While Loop structure. The While Loop, which can be thought of as a single node itself, executes the diagram inside its borders until the Boolean (True-False) value passed to the conditional terminal (box 5) is False.
2.	This box contains the string which specifies the volume name, directory name, and file name from which the temperature data (in Celsius) is to be read.
3.	This node opens the specified file and reads the current line of temperature data into a numeric array. The node then closes the file, passes the array to node 6, and passes a numeric error message value to node 4.
4.	This node checks to see if the error message from node 3 is equal to zero (indicating no error). If so, the node outputs a Boolean value of True. If the error message does not equal zero, the Boolean output is False.
5.	This box is the conditional terminal for the entire While Loop. The terminal is checked at the end of each iteration and exits the While Loop structure once the Boolean value from node 4 is False.
6.	This node reads a specified element of the numeric array from node 3 and then passes the value of that element to node 12.
7.	This is a numerical constant that contains the integer value that specifies which element of the array node 6 is to read.
8.	This is a numeric constant used in the temperature conversion from Celsius to Fahrenheit. The value is passed to node 11.
9.	This is a numeric constant used in the temperature conversion from Celsius to Fahrenheit. The value is passed to node 11.
10.	This is a numeric constant used in the temperature conversion from Celsius to Fahrenheit. The value is passed to node 13.
11.	This node divides the value from constant 8 by the value from constant 9. The numeric result (a conversion factor of 9/5) is then passed on to node 12.
12.	This node multiplies the output value from node 6 by the output value from node 11. The numeric result is passed on to node 13.
13.	This node adds the value from constant 10 to the output value from node 11. The resulting numeric output (Fahrenheit temperature) is passed on to node 15.
14.	This is a constant numeric value specified as the temperature limit.
15.	This node checks to see if the output value from node 13 is greater than the temperature limit of box 14. If so, then the resulting Boolean output is True. If the value from node 13 is less than the temperature limit of box 14, then the Boolean output is False.
16.	This is a Case structure. The node inside this structure beeps if the Boolean output from node 15 is True, which means that the temperature value read from the file is higher than the temperature limit specified by box 14.

APPENDIX C

BASIC STRUCTURES IN LABVIEW 2

The basic structures in LabVIEW 2 are explained here. These structures were also shown in Fig. 6 and briefly mentioned earlier in the text. The legend for figure C.1 (numbers in circles) is presented in Table C.1. A brief discussion of this figure is also included.

EXPLANATION OF FIGURE C.1

Structure A presents a For Loop, which performs numeric iteration. This structure is comparable to a "For Loop" in Fortran. It executes a specified number of times . The While Loop is displayed in B. This loop continues execution as long as the specified boolean condition is true. Since it checks for the true or false condition at the end of each cycle, it always executes at least once. The Case Structure, shown in structure C, may contain two or more subdiagrams, also called "cases". One of these "cases" is selected during execution as specified by the input value (which may be either Boolean or numeric scalar). Structure D shows a Sequence Structure. This structure holds numerically numbered frames which are executed sequentially. The function of the "Formula Node" in structure E is simply to hold one or more equations. This structure computes

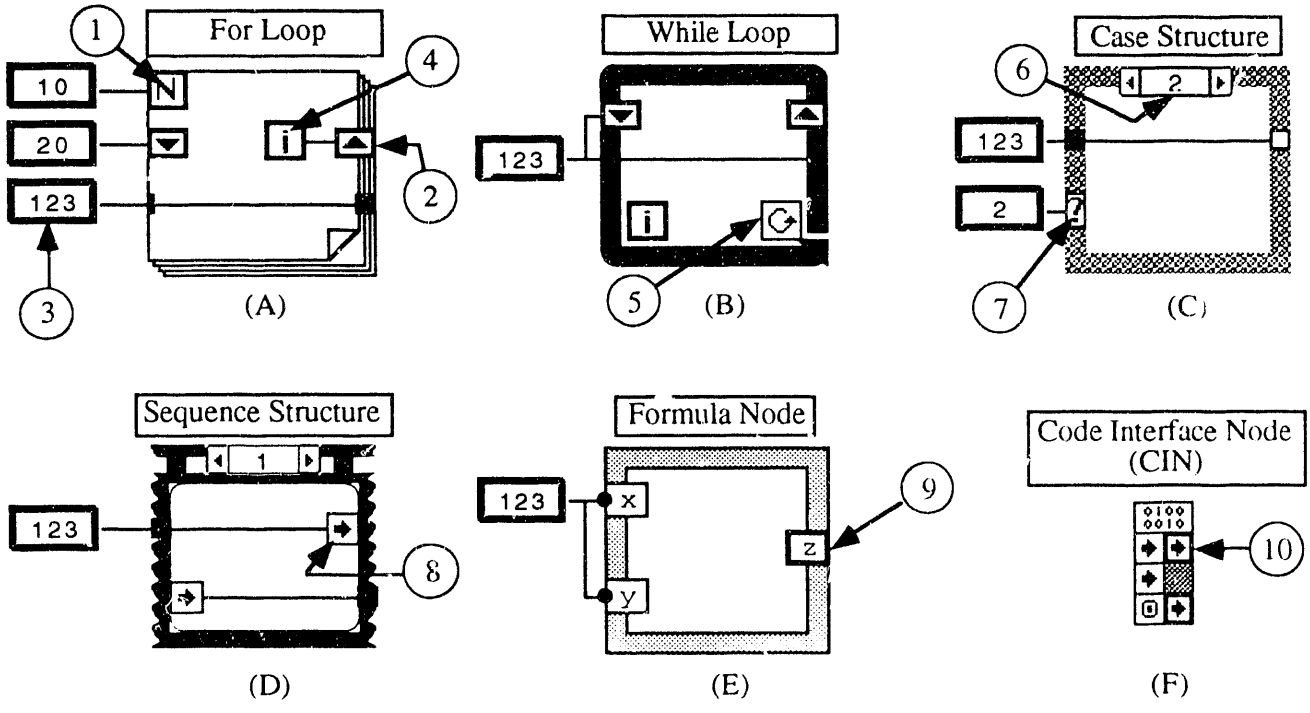


FIGURE C.1. - Basic structures in LabVIEW 2.

the equation sequentially from top to bottom and outputs the result. Structure F, the Code Interface Nodes (CIN), allows the user to program a segment of the block diagram (i.e. LabVIEW 2 program) using "C", "Pascal", or an assembly code language.

TABLE C.1
LEGEND FOR FIGURE C.1

1.	The count terminal holds the value (supplied by the constant wired to it) of how many times the loop is to be executed.
2.	The arrows on the vertical edges are shift registers, which pass values from one iteration cycle to the next. Any data stored in the down-arrow (right shift register) is available at the beginning of next iteration cycle through up-arrows (left shift register). The up-arrow may be initialized by wiring values to be used in the first iteration cycle. The down-arrow may be used to pass the last value outside the loop at the end of execution.
3.	This is an example of parameter passing through the structure. A numeric constant, the value 123, is being passed to be used in the structure.
4.	The **i** is the iteration terminal. It holds the current number of completed cycles.
5.	This curved arrow is the conditional terminal. It receives the Boolean value of the test condition at the end of each iteration. If the test value is false, it finishes the iteration.
6.	This subdiagram display window shows the case being displayed. The right and left arrows are increment buttons that allow user to observe different subdiagrams, or cases. The cases may be numeric or boolean.
7.	The selector receives the case selection information; i.e., the case to be executed.
8.	The arrows inside the frame hold the local variables. These variables pass data from one frame to the subsequent frame. The inward arrow indicates a local variable which is wired to receive the value, whereas the outward arrow indicates a local variable which already has a value which can be distributed.
9.	An example of the parameter passing method for the formula node. Parameters may be passed through the vertical edges. In this example, **z** is the output variable and **x** and **y** are input variables.

APPENDIX D

HIERARCHICAL STRUCTURE OF NIPER'S DATA ACQUISITION SOFTWARE

The hierarchical structure of NIPER's data acquisition program in LabVIEW 2 is presented in Fig. D.1. Whereas the functionality of this facility was described earlier in the text, the functionality of each module in the corresponding program is explained here in Table D.1.

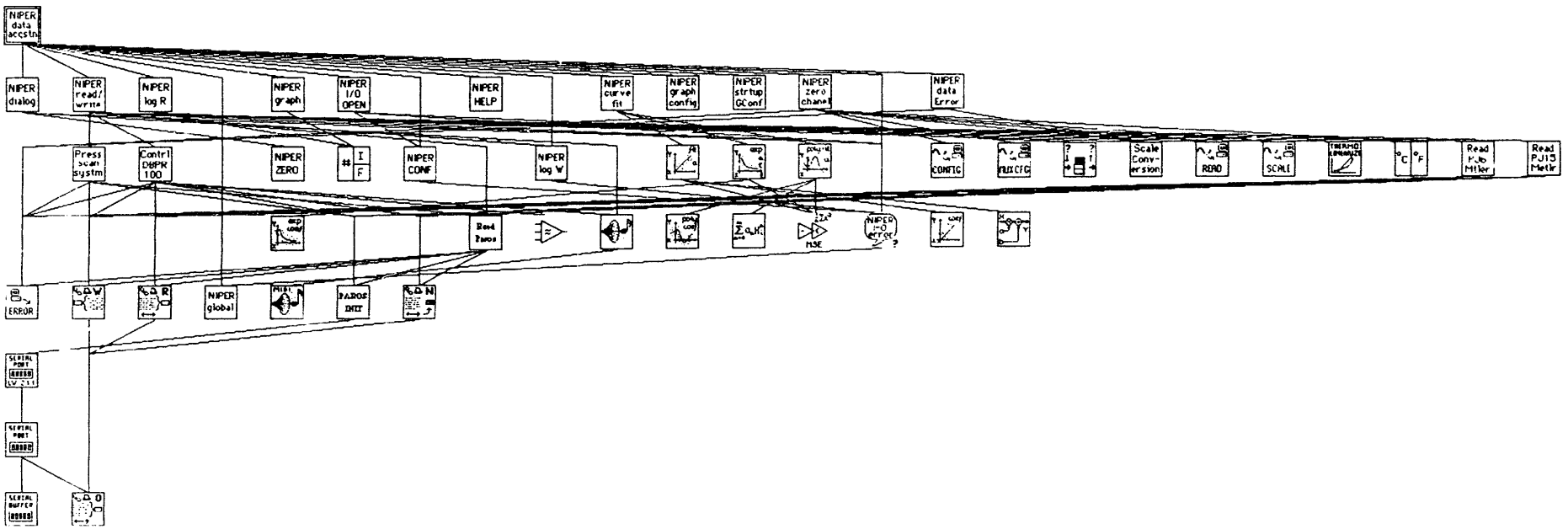


FIGURE D.1. - Hierarchical structure of NIPER's data-acquisition facility.

TABLE D.1

MODULES IN THE DATA-ACQUISITION PROGRAM AS SHOWN IN FIGURE D.1

	Node Icon	Description
1.	NIPER data acqstn	This is the driver VI (virtual instrument) for data acquisition facility.
2.	NIPER dialog	Displays numeric data, warning signals and error messages.
3.	NIPER read/ write	Reads and writes analog and digital data to and from all the boards (GPIB, Serial, Analog I/O), process it, displays it, stores it in files, and pass on to graphic modules.
4.	NIPER log R	Reads data of previous runs from log files one run at a time.
5.	NIPER graph	Receives recent data from NIPER read/write (#3)* or receives previously stored data from NIPER log R (#4) as per user request.
6.	NIPER I/O OPEN	Opens a panel at the beginning of the test for the user to select one of the previous settings or select a new setting. If user does not respond, the program selects the last setting. See NIPER CONFIG (#17) for the type of settings it can handle.
7.	NIPER HELP	Opens a panel that gives basic information about LabVIEW and the data acquisition program. In this program addition, 5/20/92, this facility is not interactive or user selectable.
8.	NIPER curve fit	Receives recent data from NIPER read/write (#3) or previous data from NIPER log R (#4) and superimposes curve fit data/lines.

* The numbers indicated with a "#" symbol in this table refer to one of the other modules listed here, e.g. #3 refers to "NIPER read/write" module.

TABLE D.1

MODULES IN THE DATA-ACQUISITION PROGRAM AS SHOWN IN FIGURE D.1—Continued

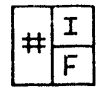
	Node Icon	Description
9.		Selects how much and which channels of data to be displayed. The module also allows selection of the kind of curve fit desired.
10.		This is the startup (default) setting for NIPER graph configuration (#9).
11.		Takes the calibration reading from analog board to compensate for temperature readings.
12.		Receives error messages from various modules, prioritizes them and brings the error message to the user attention if parameters are out of range.
13.		Driver VI for the Scanivalve multiple port scanning system. Advances to a channel selected by user. If no residence time is selected, the module takes reading immediately, otherwise it takes readings in subsequent program cycles. The Scanivalve then advances to the next selected channel and repeats until all channels have been scanned. The program then waits for the next run interval to repeat.
14.		Driver VI for back pressure controller. Communicates in each program cycle to make sure the communication is not lost and the actual pressure is close enough to the control pressure. Communicates with sumcheck protocol and three consecutive OK's to safeguard from mishaps.
15.		Initializes all the registers to zero before beginning of a test to eliminate data corruption.
16.		Separates the integer and fractional parts of any number. For example, for the number 123.456 the integer part is 123.000 and the fractional part is 0.456.

TABLE D.1

MODULES IN THE DATA-ACQUISITION PROGRAM AS SHOWN IN FIGURE D.1—Continued

	Node Icon	Description
17.		Opens a front panel for user to configure the system interactively. The user can select reading intervals, establish links with one or more of the available channels, deactivate a link until further selection, and define maximum and minimum data values for each channel so that if a reading falls beyond these preset limits, the user may be alerted. This module has several other attributes. This is an interactive panel open only on user's request. If the user does not respond, or if user does not choose to use this facility, the last setting or the one selected by the user at the beginning of a test is used a default.
18.		Stores all the information about the current test including data, and updates every program cycle. User can then view or use them interactively in later runs by using NIPER log R (#4) module.
19.		Finds the line, slope and intercept which best describe the input (X,Y) sequence of values using the least mean squared error criterion.
20.		Finds the exponential curve, amplitude and decay parameters which best describe the input (X,Y) sequence of values using the least mean squared error criterion.
21.		Finds the polynomial curve and polynomial coefficients which best describe the input (X,Y) sequence of values using the least mean squared error criterion.
22.		Updates analog input configuration information.
23.		Configures the number of multiplexer boards connected to the IO board.

TABLE D.1

MODULES IN THE DATA-ACQUISITION PROGRAM AS SHOWN IN FIGURE D.1—Continued

	Node Icon	Description
24.		Set the Boolean to the input value (True or False) if mode is TRUE, otherwise retain the last value. Return the current value of the Boolean.
25.		Converts the data to user specified units.
26.		Reads the specified analog input channel (initiates an Analog to Digital conversion on an analog input channel and returns the result.
27.		Converts the binary result from the Read (#26) module to the actual input voltage read from that channel.
28.		Takes a voltage value acquired from a thermocouple of specified type, and converts it into the corresponding temperature at the thermocouple in degrees Celsius.
29.		Converts degrees Celsius to degrees Fahrenheit according to the formula: F = C * 9/5 + 32.
30.		This is the driver VI for Mettler balance (type PJ6). Controls and reads signal via RS232 communication protocol (serial).
31.		This is the driver VI for Mettler balance (type PJ15). Controls and reads signals via RS232 communication protocol (serial).
32.		Finds the exponential amplitude and decay parameters which best describe the input (X,Y) sequence of values using a least mean squared error criterion.

TABLE D.1

MODULES IN THE DATA-ACQUISITION PROGRAM AS SHOWN IN FIGURE D.1—Continued

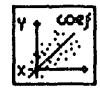
	Node Icon	Description
33.		This is the driver VI for the Paroscientific DigiQuartz pressure computer. Controls and reads signals via RS232 communication protocol (serial).
34.		Returns TRUE if the difference between A (top input) and B (bottom input) is less than the specified tolerance (center input), otherwise FALSE. Tolerance is 0.1 if unwired (default).
35.		Beeps once with the given frequency (pitch), intensity (volume, also depends on the speaker volume set in the control panel), and duration [number of 1/60ths of a second, up to 600 (10 secs)].
36.		Finds the polynomial coefficients which best describe the input (X,Y) sequence of values using the least mean squared error criterion.
37.		Performs a polynomial evaluation of the input sequence X using the specified coefficients a and the polynomial order m. The number of elements in the coefficients array is m+1.
38.		Computes the mean squared error value of the input sequences X and Y.
39.		Handles file input/output errors. Prompts user for an alternate action. If the user does not respond, the module performs a default save.
40.		Finds the slope and intercept which best describe the input (X,Y) sequence of values using the least mean squared error criterion.

TABLE D.1

MODULES IN THE DATA-ACQUISITION PROGRAM AS SHOWN IN FIGURE D.1—Continued

	Node Icon	Description
41.		Performs a linear evaluation on the input sequence X using the specified: Multiplicative constant (a), abd Additive constant (b). The output sequence Y is of the form Y = X*a + b.
42.		Translates a numeric LabDriver error code into a string message describing the error.
43.		Writes the specified no. of bytes to the serial board in order to send out to the connected instrument.
44.		Reads the specified no. of bytes from the serial board that were sent in by the connected instrument.
45.		Sets the Boolean to the input value (True or False) if mode is TRUE, otherwise retain the last value. Return the current value of the Boolean.
46.		Beeps once with the given MIDI frequency (a standard preset frequency scheme), intensity (volume, also depends on the speaker volume set in the control panel), and duration (number of 1/2 milliseconds, up to 32767).
47.		Initializes serial communication parameters (RS232 protocol) for communicate with paroscientific DigiQuartz pressure computer.
48.		Probes the board and determines the number of bytes available to be read.
49.		Initializes serial port with user-provided parameters.

TABLE D.1

MODULES IN THE DATA-ACQUISITION PROGRAM AS SHOWN IN FIGURE D.1—Continued

	Node Icon	Description
50.		Initializes serial port with user-provided parameters.
51.		Sets serial port buffer size.
52.		Opens serial board for communication.

APPENDIX E

NIPER'S DATA ACQUISITION FACILITY

Figure E.1 shows the front panel of NIPER's data acquisition facility. Individual components of this front panel are described here, whereas the functionality of this facility was described earlier in the text. The legend for this figure (numbers in circles) is presented in Table E.1.

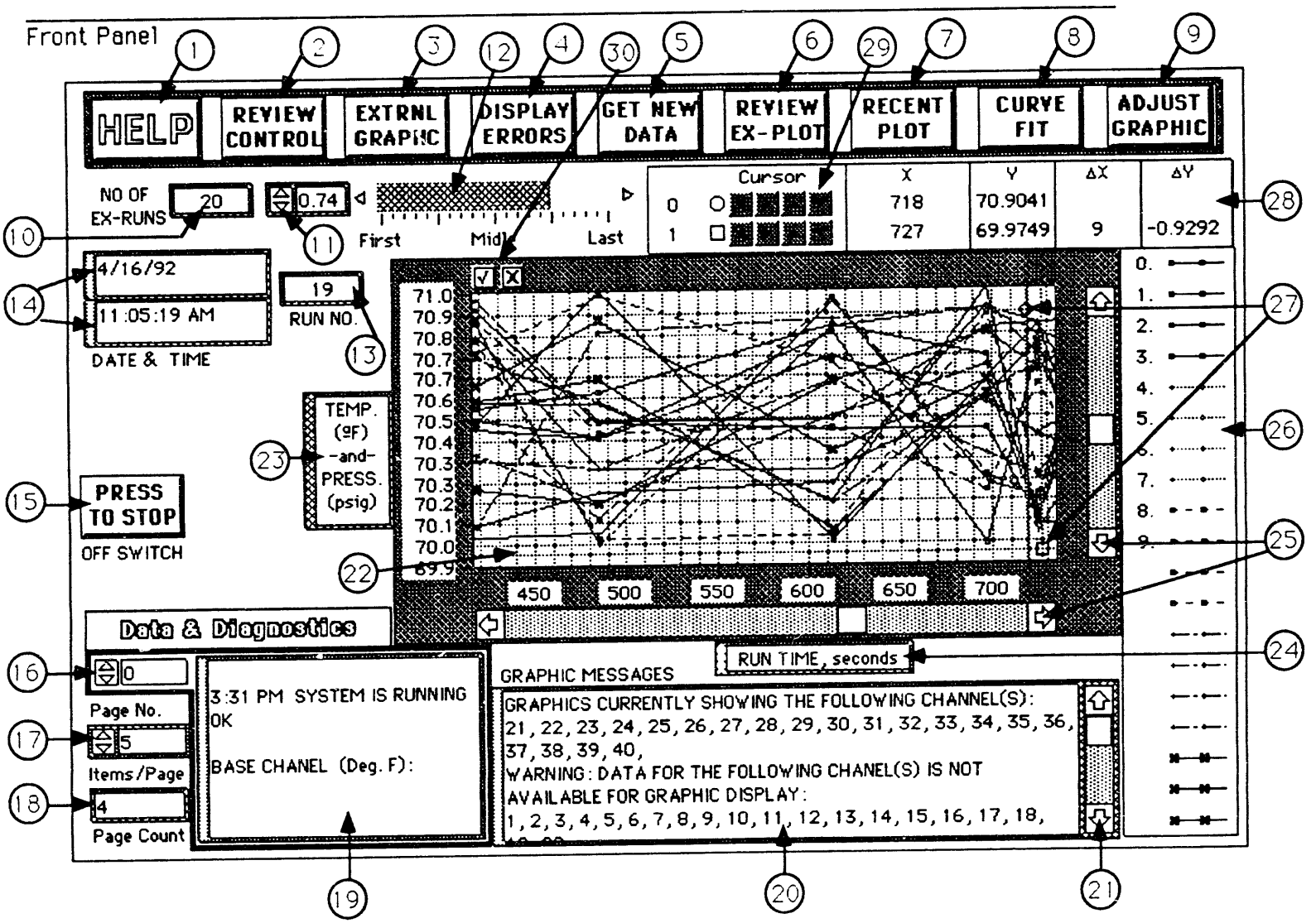


FIGURE E.1 - Front panel of NIPER's data acquisition module. Numbers in the circles refer to the legend on the next page.

TABLE E.1
LEGEND FOR FIGURE E.1

1.	This button opens a scroll window which provides information about LabVIEW 2 and object oriented programming. Like all of the other windows or panels that are opened from the main panel, the help window automatically disappears, and the main panel reappears, after a certain time duration. The user must re-press a button if the corresponding window or panel needs to be displayed.
2.	This button opens a panel whereby user can set the status time, time interval, and number of multiplex boards. Other options allow user, for each individual channel, to choose the status (active or inactive), the channel location, the sensor, and the alarm (for a low limit and a high limit). If any changes are made, the user must press the ACCEPT button at the bottom of the panel. There are also options to cancel the changes, or to store a particular setting for later use.
3.	This button enables user to relay information to external programs. The default program is chosen; an option under ADJUST GRAPHIC lets user change the particular program if that is desired.
4.	This button displays and explains all the errors in the current run. Errors previously suppressed are not displayed till this button is pressed.
5.	This button allows a data reading to be taken at exactly the time this button is pressed. This additional data reading is added to the stored data in the current run.
6.	This button enables user to view plots of previous runs (one run at a time only). The desired run number can be chosen by using either button 12 (clicking on a particular place in the bar) or by using using button 11 (scrolling the arrows). Button 13 displays the number of the previous run. After this previous run number is chosen, pressing button 6 will display the plot of that particular previous run.
7.	This button displays plot of all data in current run. When button 8 (CURVE FIT) is pressed concurrently, the added data from this option is superimposed on top of the original data which was displayed by button 7.
8.	This button provides a fit (linear, polynomial, etc.) to those data sets already selected by user for regression analysis in ADJUST GRAPHICS module (button 9), whereby the type of curve fit and the order of fit can also be specified. The curve fit option is activated by pressing button 8 concurrently with button 7.
9.	This button opens a panel, which, in addition to controlling the type and order of fit, allows user to select the channel(s) that are to be plotted. Another option is pressing PRESS FOR NEW DATA to display only the data from that point in time. Still another feature lets user select the time base of the desired interval. After any of these options have been changed, pressing the ACCEPT button will incorporate these changes while pressing the CANCEL button will return these options to their previous setting. And finally, as mentioned earlier, the EXTERNL GRAPHIC in this panel lets user select the particular program the user wants to communicate with.
10.	This display shows the number of previous runs.
11.	These increment buttons are one of the ways (the slower but more more discriminate way) the user can select the specific run number before button 6 is pressed to display the plot for that run. The fraction shown simply indicates what percentage of the total previous runs is the run number now selected.
12.	Pressing this bar at a particular place is the faster and less discriminate way of selecting the specific number of the previous run. Using the increment buttons accomplishes the same task but at a much slower but more discriminate manner.

TABLE E.1
LEGEND FOR FIGURE E.1—Continued

13.	This indicator shows the number of the specific previous run.
14.	This indicator shows the date and time of the start of specific previous run number. This indicator and indicator 13 display data only after button 6 (REVIEW EX-PLOT) is pressed.
15.	This button ends the current run, but the data from that run is stored and the number of ex-runs increases by one.
16.	These increment buttons let user select the page number the DATA & DIAGNOSTCS screen will display.
17.	These increment buttons let user select the number of items to be displayed per page of the DATA & DIAGNOSTICS screen.
18.	This display shows the total number of pages of available data.
19.	This DATA & DIAGNOSTIC screen displays the current data from all the activated channels and diagnoses any errors.
20.	This GRAPHIC MESSAGES screen indicates the channels that are currently graphed, the channels that are open for graphic display but lack data, and the channels that have data but have not yet been activated.
21.	These arrows let the user scroll the GRAPHIC MESSAGES screen.
22.	This graph displays the plot. The numbers on the y-coordinate can be controlled automatically or manually altered.
23.	This indicator shows the units information for the x-coordinate and y-coordinate. However, this indicator shows all of the units currently in use and in addition, is dynamic.
24.	This indicator shows the units of the chosen time interval.
25.	These arrows let the user scroll the PLOT DISPLAY screen.
26.	This "legend" matches the lines in the plot with their respective channel number.
27.	There are two cursors (a square and a circle) that can be moved around by the arrows in button 29. The part of the plot displayed will change ; it will follow the moving cursor(s). It is also possible that one or both of the cursors are "off" the screen ; this situation can be corrected by either readjusting the scales of the x and/or y coordinates or by moving the cursor(s) onto the view of the displayed screen by using button 29.
28.	*Delta y* indicates the vertical difference, or distance, between the two cursors while *delta x* indicates the horizontal difference.
29.	These arrows move the cursor(s) around the screen. As mentioned earlier, the displayed screen follows the moving cursor(s).
30.	This button allows editing of the plot. Enables user to adjust line segments and other aspects.

APPENDIX F

NIPER'S INSTRUMENT CONTROL FACILITY

Figure F.1 shows the front panel of NIPER's instrument control facility. Individual components of this front panel are described here, whereas the functionality of this facility was described earlier in the text. The legend for this figure (numbers in circles) is presented in Table F.1.

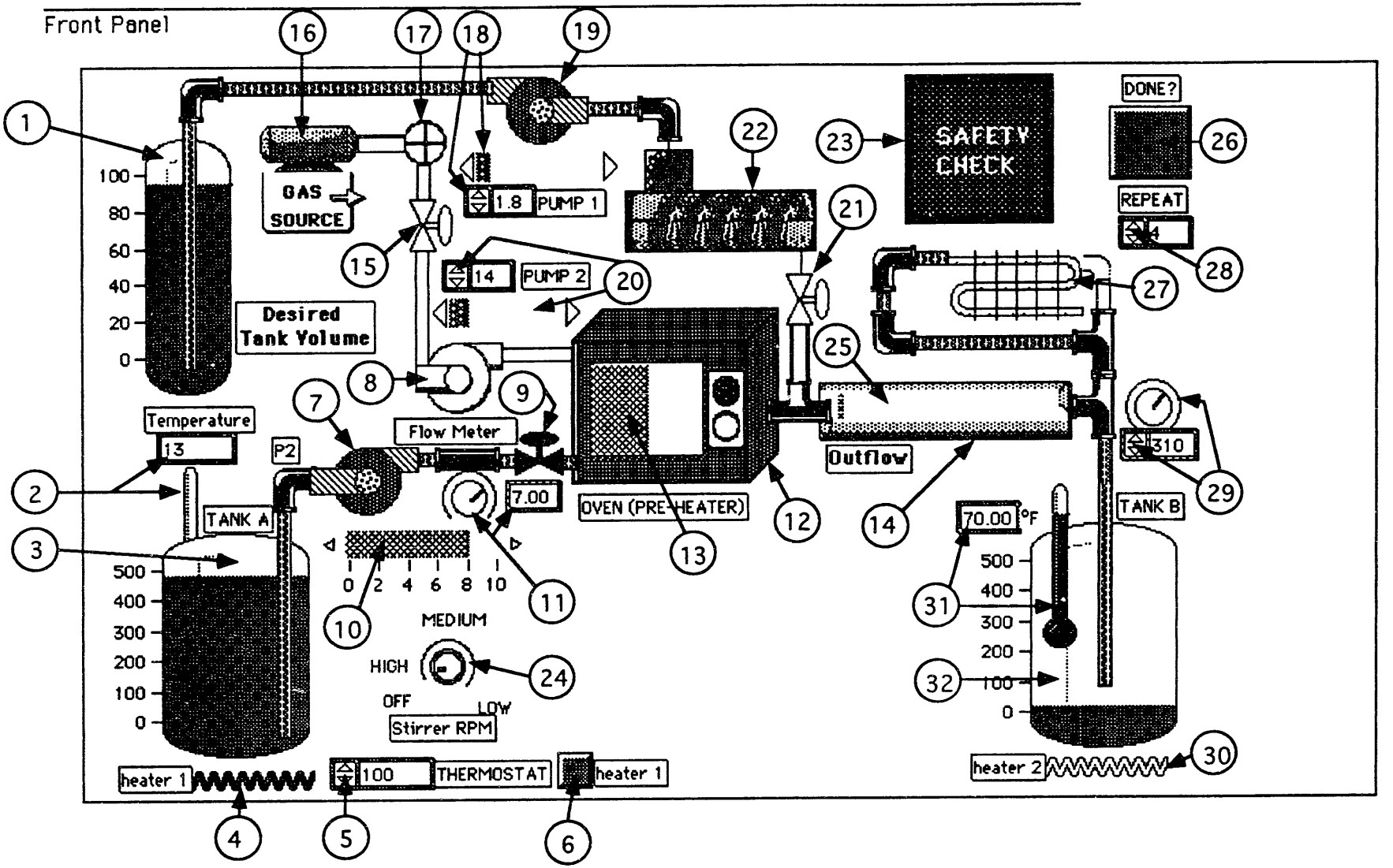


FIGURE F.1. - Front panel of NIPER's instrument control module. Numbers in the circles refer to the legend on the next page.

TABLE F.1
LEGEND FOR FIGURE F.1

1.	This is the indicator of the tank which stores the feed water that will later be converted to steam by the steam generator.
2.	This is the indicator of the thermometer which reads the temperature of liquid in TANK A and displays this temperature.
3.	This is the indicator of TANK A which stores liquid. The amount of this liquid is controlled by a mass balance that can, at all times, determine how much (and at what rate) this liquid is flowing out and replace this amount in the tank. If the material balance system is not working correctly - that is, if the amount of liquid out is not equal to the amount flowing in, sensors can detect a malfunction and the system tries to diagnose this error and indicate the information to the user.
4.	This is the indicator of this sensor which is connected to a heater (heater 1) that heats TANK A. The sensor is *red* when the heater is activated and *blue* when it is not activated.
5.	These increment buttons allow the user to set the thermostat of the heater that heats TANK A at a certain temperature.
6.	This button needs to be pressed if any changes are made in the thermostat setting. Otherwise, the previous setting is used.
7.	The indicator of this pump, which transfers the liquid from TANK A.
8.	Another pump - this one is responsible for pumping gas from the gas source.
9.	This is the valve for TANK A. The user can turn the valve *on* and *off* ; the valve is different colors depending on whether it is *on* or *off*.
10.	This and the two increment buttons on either side, can be used to control the flow rate of the liquid flowing out of TANK A.
11.	These are two flowmeters, one pictorial and the other digital. They are equivalent ways of indicating the flow rate.
12.	The indicator of the OVEN (PRE-HEATER) heats the liquid from TANK A and the gas from the GAS SOURCE to the same temperature as the steam. Otherwise, the mixture of these three will result in the condensation of the steam. The red and green lights are used to turn the OVEN *on* and *off*, respectively. The red light is lit when the heater is *on* and the green light is lit when the heater is off.
13.	This repetitively appearing whitish-grey screen moves across the screen of the OVEN icon to indicate that the OVEN icon is on.
14.	This is the pressure vessel containing porous media through which the liquid surfactant, the gas, and the steam flow.
15.	This is the control of the valve for the GAS SOURCE. The user can turn the valve *on* and *off*. Different colors for the valve indicate this status.
16.	This is the indicator of the GAS SOURCE which stores the gas. Here also, a material balance system, which takes the reading of the flow rate of the gas (through option 20) and the reading of the amount of gas exiting the gas source (through option 29), is responsible for maintaining enough gas in the GAS SOURCE.
17.	This is the indicator of the pressure regulator which can be used to define an upper limit of the gas in the GAS SOURCE.
18.	Either the vertical arrows (used for fine control) or the horizontal arrows (used for gross control) can be used to control the rate of flow of the feed water.
19.	This is the icon of the pump which is responsible for pumping the feed water to the steam generator.

*U.S.GPO:1992-661-026/60052

TABLE F.1
LEGEND FOR FIGURE F.1—Continued

20.	Either the scroll arrows (used for fine control) or the horizontal arrows (used for gross control) can be used to control the rate of flow of the gas from the GAS SOURCE.
21.	This is the control of the valve for the feed water-converted-to-steam system. The user can turn the valve *on* and *off* and different colors for the valve are used to indicate this status.
22.	The indicator of the steam generator which converts the feed water into steam.
23.	This indicator shows the picture of a "green man" when everything is functioning properly. When something is malfunctioning, the system tries to diagnose the cause of the problem and indicate it to the user so remedial steps can be taken.
24.	This control lets the user determine the speed of the stirring mechanism (stir bar) in TANK A. This speed determines the rate at which the different components of the liquid surfactant are mixed.
25.	The repetitively appearing picture of a blue tube that moves across the core indicates that something is flowing through the pressure vessel.
26.	Pressing this button opens the DATA ACQUISITION panel from which users can select any number of options.
27.	The steam circulates and is condensed and is eventually taken into TANK B along this drain.
28.	These arrows allow the user to choose the number of times the experiment is to be repeated. Each experiment takes a set amount of time.
29.	This is the control of the backpressure regulator which releases the gas periodically to maintain a certain pressure in this part of the system. The user can choose this pressure by using the two arrows.
30.	This is the indicator of another sensor which is connected to a heater (heater 2) which heats TANK B. The sensor is *red* when the heater is activated and *blue* when it is not activated.
31.	This is the indicator of the thermometer which takes the temperature of the liquid in TANK B and displays this temperature reading.
32.	This is the indicator of TANK B which stores the mixture of the liquid surfactant and the water condensed from the steam.

END

DATE
FILMED
11/19/92