grateful for helpful discussions with Drs. P. Willett, S. M. Welford, and J. M. Barnard.

REFERENCES AND NOTES

(1) Lynch, M. F.; Barnard, J. M.; Welford, S. M. "Computer Storage and Retrieval of Generic Chemical Structures in Patents. 1. Introduction and General Strategy". *J. Chem. Inf. Comput. Sci.* **1981**, *21*, 148-150.
(2) Barnard, J. M.; Lynch, M. F.; Welford, S. M. "Computer Storage and Retrieval of Generic Chemical Structures in Patents. 2. GENSAL, a Formal Language for the Description of Generic Chemical Structures". *J. Chem. Inf. Comput. Sci.* **1981**, *21*, 151-161.
(3) Welford, S. M.; Lynch, M. F.; Barnard, J. M. "Computer Storage and Retrieval of Generic Chemical Structures in Patents. 3. Chemical Grammars and Their Role in the Manipulation of Chemical Structures". *J. Chem. Inf. Comput. Sci.* **1981**, *21*, 161-168.
(4) Barnard, J. M.; Lynch, M. F.; Welford, S. M. "Computer Storage and Retrieval of Generic Chemical Structures in Patents. 4. An Extended Connection Table Reprsentation for Generic Structures". *J. Chem. Inf. Comput. Sci.* **1982**, *22*, 160-164.
(5) Barnard, J. M.; Lynch, M. F.; Welford, S. M. "Computer Storage and Retrieval of Generic Chemical Structures in Patents. 5. Algorithmic Generation of Fragment Descriptors for Generic Structure Screening". *J. Chem. Inf. Comput. Sci.* **1984**, *24*, 57-66.
(6) Barnard, J. M.; Lynch, M. F.; Welford, S. M. "Computer Storage and Retrieval of Generic Chemical Structures in Patents. 6. An Interpreter Program for the Generic Structure Language GENSAL". *J. Chem. Inf. Comput. Sci.* **1984**, *24*, 66-70.
(7) Welford, S. M.; Ash, S.; Barnard, J. M.; Carruthers, L.; Lynch, M. F.; von Scholley, A. " The Sheffield University Generic Chemical Structures Research Project". In *Computer Handling of Generic Chemical Structures*; Barnard, J. M., Ed.; Gower: Aldershot, 1984; pp 130-158.
(8) von Scholley, A. "A Relaxation Algorithm for Generic Chemical Structure Screening". *J. Chem. Inf. Comput. Sci.* **1984**, *24*, 235-241.
(9) Gillet, V. J.; Welford, S. M.; Lynch, M. F.; Willet, P.; Barnard, J. M.; Downs, G. M.; Manson, G. A.; Thompson, J. "Computer Storage and Retieval of Generic Chemical Structures in Patents. 7. Parallel Simulation of a Relaxation Algorithm for Chemical Substructure Search". *J. Chem. Inf. Comput. Sci.* **1986**, *26*, 118-126.
(10) Lynch, M. F. "Generic Chemical Structures in Patents (Markush Structures)—the Research Project at the University of Sheffield". *World Pat. Inf.* **1986**, *8*, 85-91.
(11) Dittmar, P. G.; Farmer, N. A.; Fisanick, W.; Haines, R. C.; Mockus, J. "The CAS ONLINE Search System. 1. General System Design and Selection, Generation and Use of Search Screens". *J. Chem. Inf. Comput. Sci.* **1979**, *19*, 51-55.
(12) Lederberg, J. "DENDRAL-64. A System for Computer Construction, Enumeration and Notation of Organic Molecules as Tree Structures and Cyclic Graphs. Part 2. Topology of Cyclic Graphs". NASA CR-68898, National Aeronautics and Space Administration Report No. N66-14074, 1966.
(13) Lederberg, J. "DENDRAL-64. A System for Computer Construction, Enumeration and Notation of Organic Molecules as Tree Structures and Cyclic Graphs. Part 3. Complete Chemical Graphs: Embedding Rings in Trees". NASA CR-123176, National Aeronautics and Space Administration Report No. N71-76061, 1971.
(14) Lederberg, J. "Topological Mapping of Organic Molecules". *Proc. Natl. Acad. Sci. U.S.A.* **1971**, *53*, 134-139.
(15) Balaban, A. T.; Filip, P.; Balaban, T.-S. "Computer Program for Finding All Possible Cycles in Graphs". *J. Comput. Chem.* **1985**, *6*(4), 316-329.
(16) Morgan, H. L. "Generation of a Unique Machine Description for Chemical Structures—a Technique Developed at Chemical Abstracts Service". *J. Chem. Doc.* **1965**, *5*, 107-113.
(17) Cooper, D. C.; Lynch, M. F. "Review of Variety Generation Techniques". British Library R & D Report No. 5586, London, British Library, 1980.

ARTS: A Flexible Laboratory Instrument Control Language

W. A. SCHLIEPER and T. L. ISENHOUR*

Department of Chemistry and Biochemistry, Utah State University, Logan, Utah 84321-0300

J. C. MARSHALL

Department of Chemistry, Saint Olaf College, Northfield, Minnesota 55057

Received March 10, 1987

A generalized computer program, ARTS (Analytical Robot Telecommunications Software), has been developed to give the research scientist more flexible control of laboratory robots and instruments. As a stand-alone program, ARTS is a complete laboratory control language. ARTS can also be an extension of other software in either a master or slave mode. As master, ARTS can call on other software to perform certain tasks. In a slave mode, ARTS can act as a sensory extension of the calling software. ARTS is a flexible laboratory control language capable of adapting to changing laboratory requirements.

INTRODUCTION

Robots are important laboratory instruments, and as robot use increases, so does the need for improved external control by computers. Current laboratory robots are principally used to perform repetitive operations.[1-6] In research laboratories, the experiments performed are more varied and require diverse procedures[7] including changing chemical or instrumental procedures on the basis of intermediate results. For laboratory robots to achieve maximum utilization in the research laboratory, it will be necessary to have external computer control designed for maximum flexibility.

Attempts have been made to improve communication between robot systems and external computers.[8,9] These examples show the ability of the robots to interact outside their local environment. Individually these attempts address part of a bigger problem, which is the need for a more versatile robot control language to deal with changing laboratory requirements.

A generalized computer program, ARTS (Analytical Robot Telecommunications Software), has been developed to give the research scientist more flexible control of laboratory robots and instruments. Flexibility has been incorporated into the software design to promote greater software control. ARTS is capable of controlling laboratory robots and instruments on its own or in conjunction with other standard software packages and runs under the MS-DOS microcomputer operating system.

ANALYTICAL ROBOT TELECOMMUNICATIONS SOFTWARE

As a stand-alone program, ARTS is a complete laboratory control language that can interpret commands in either interactive or batch modes. The ARTS interpreter is written in the C programming language and uses Reverse Polish notation invented by Lukasiewicz.[10] Each line of input is parsed according to precedence rules of the arithmetic operators

0095-2338/87/1627-0137$01.50/0

Table I. Precedence Rules of Operators[a]

operators	type	priority
functions	prefix	10
not, unary plus, unary minus	prefix	9
(power)^	infix	8
(multiplication) *, /, MOD	infix	7
(addition) +, -	infix	6
(comparison) <, <=, >, >=	infix	5
(equality) =, <>	infix	4
(logical) and	infix	3
(logical) or	infix	2
(logical) xor	infix	1

[a] Operators of equal importance are specified on the same line.

(Table I). The arguments associated with operators of higher priority are evaluated before arguments corresponding to operators of lower priority. Arguments of operators with equal precedence are evaluated from left to right. Parentheses can be used to override the order of evaluation. Operators in Table I are of two types, infix and prefix. Infix implies that the operator is between two arguments as in

$$3 + 4$$

where the + operator acts on the two arguments 3 and 4 to return the sum. A prefix operator acts upon one argument immediately following it, an example being

$$\text{NOT } A$$

where the operator NOT operates on the value stored in variable A, returning the logical opposite. Although the internal representation is in Reverse Polish notation, the actual language interpreted is a combination of BASIC commands, operating system commands, memory management commands, instrument commands, and commands specific to ARTS. BASIC protocol was chosen for ARTS because most scientists have a working knowledge of BASIC programming. The use of a familiar instruction set will enable most scientists to use ARTS immediately.

Operating system commands consist of drive and file management commands (Table II). Without any additional parameters, the CD command returns the default drive and directory to the screen. When the CD command is followed with a valid directory, the default directory is changed. The COPY, REN, and DEL commands allow the programmer complete control of the status of disk files from within ARTS. While all of the operating system commands can be accessed in batch or interactive mode, the EDIT, TYPE, and DIR commands are mainly used in interactive debugging. The EDIT command invokes a user-specified editor allowing the programmer to modify routines without leaving ARTS. The TYPE command allows the user to display the contents of an ASCII (American Standard Code for Information Interchange) file on the screen for inspection. The DIR command displays information about files in the default or specified directory on the screen. In the interactive mode, ARTS also allows direct access to the operating system.

Memory management commands allow greater control over variables than standard BASIC (Table II). Variables are represented in programs by any combination of ASCII letters, numbers, and nonoperator symbols up to 30 characters in length. The first character of a variable must be a letter. The two define statements, DEF and DEFS, are used to define arrays of numeric and string variables. A nonarray variable can be explicitly defined by using one of the define commands or implicitly defined the first time it is encountered in a routine. ARTS does not require that a string variable end with a dollar sign ($). The DISP command displays the name, type, and value of all defined variables in memory. The ERA command can be used to selectively erase any or all memory variables. All memory management commands can be used in either interactive or batch modes to control the state of variables.

Table II. Commands Used in ARTS

Operating System Commands
- CD: change default drives and/or directory; can also be used to display current drive and directory
- COPY: copy a disk file to a different disk file
- DEL: deletes disk file(s)
- DIR: directory listings
- EDIT: invokes a user-specified editor
- REN: renames a disk file

Memory Management Commands
- DEF: defines numeric variables
- DEFS: defines string variables
- DISP: displays all variables in memory
- ERA: erases variables from memory storage

Additional ARTS Commands and Function Keys
- EVAL: executes a string or variable-containing string
- SDEF: saves default values to a disk file
- LDEF: loads default values from a disk file
- ECHO: toggles echo of subroutine command lines
- ASSIST: displays activity on communication lines
- QUIT: returns control to operating system
- F1: function key used to display mnemonic commands
- F2: function key used to activate DOS shell
- F4: access highlighted mnemonic command
- F5: function key to leave ARTS (same function as QUIT)
- F6: create a stream-oriented file
- F9: function key used to access default values
- F10: function key used to interrupt active routines

A set of commands that are specific to the ARTS environment are presented in Table II. The most nonstandard command is the EVAL command, which allows a string or string variable received from an external source or generated in ARTS to be executed. The EVAL command allows adaptive programming capabilities. The LDEF command can be used to load a file containing system defaults. ARTS loads a default file when it is first executed, but allows a different default file to be loaded at any time. The SDEF saves the current defaults to a disk file. The ECHO command is used to turn on or off the echoing of routine commands on screen as they are being executed. The ASSIST command is a debugging tool for communication between ARTS and laboratory instruments. It can be used to echo the values sent on the communication lines to ensure proper formats are being used. The QUIT command can be executed in batch or interactive mode to return to the operating system. Alternatively, the F5 key can be used in interactive mode to leave ARTS. The F1 key is used to toggle the use of the mnemonic command menu on the upper portion of the screen. When the menu is present, a command can be highlighted by using the arrow, page up, page down, home, and end keys. Once highlighted, the command can be placed on the command line by pressing the F4 key. The F6 key can be used to produce a stream-oriented file. Values typed in or extracted from the mnemonic menu are directed to a specified file. The file is closed the second time the F6 key is pressed. The resulting file can then be executed as a routine. The F9 key is used to gain access to a series of menus to set system and instrumental defaults. The F10 key is use to interrupt execution of a routine. Once the F10 key is pressed, a message is displayed on the screen and the user is given the options of stopping execution of the routine, stepping one line at a time through the routine, entering a series of commands, or continuing the execution of the present routine.

Instrument commands fall into four categories: those that send parameters to instruments, those that receive data from instruments, those that send parameters and receive data, and those that cause the instrument to perform an action. Table III contains some characteristic examples of these categories. Commands in the first category can be used to pass infor-

Table III. Instrument Commands

```
Commands That Send Parameters
  GRIP = 100
  MEASURE 1,0,0,0

Commands That Receive Parameters
  WEIGHT(1) = WEIGH
  MEASUREMENTS = NMEAS

Commands That Send and Receive Parameters
  TIME = CLOCK(4)
  ABSORBANCE = VALUE(750)

Commands That Perform an Operation
  MLS
  REFERENCE
```

mation, such as grip tension for a robot hand or parameters needed to collect a spectrum. The second category is used when instruments return data that can be used in conditional statements or, as in the examples, stored in variables for later use. The first example of this category shows how the weight of a sample is returned by a balance and stored as the first entry of array WEIGHT. The second of these examples stores the number of scans for a spectrum in the variable MEASUREMENTS. Combinations of the previous two commands are shown in the third category of commands. In the first example of this type, the time is returned from clock 4 and stored in the variable TIME, while the second example stores the absorbance value at 750 nm in variable ABSORBANCE. The last category of commands only requires that the command be sent to the instrument. The examples of this category show commands that control a syringe station (MLS) and collect a reference on a spectrophotometer (REFERENCE). The number of parameters to be sent, the direction of data transfer, and the instrument to which the command is to be sent are required to communicate with any laboratory instrument. The variety of commands offered by ARTS allows the user to exploit the full power of a microcomputer to control laboratory equipment.

COMMUNICATION WITH LABORATORY INSTRUMENTS

ARTS is capable of controlling any type of programmable laboratory instrument or robot, although it was originally used to control a ZYMATE I robot. Control is implemented by using RS232 communication links between the microcomputer and the laboratory instruments. The methods of communicating with different instruments vary with manufacturer. For this reason, a menu is used to define the communication scheme necessary for each instrument to be controlled. Parameters, such as baud rate, parity, etc., must be specified, stored in a file, and used to control communication with the instruments.

The robot(s) is (are) operated just as another instrument in the system. Some commercial laboratory robots use predefined positions in their normal operations. These positions are defined by using a teaching method supplied with the robot. During the teaching process, the robot is positioned at a point in its field of influence. The user supplies this position with an ASCII name that is used to access it. These position names along with the other robot control commands can be grouped into routines to perform tasks like picking up a test tube. Line numbers are only used when needed for a GOTO statement. Collections of routines can be combined to perform complete analyses.

To facilitate communication with instruments, a look-up table is used. The table allows the user to give a simple mnemonic definition to a complex sequence of commands that contain the number of parameters to be sent, the direction of data transfer, and the communication port to use for the instrument. In addition to simplifying commands, this method also allows the program to check for the proper number and type of parameters to be sent. The error checking of incorrect argument lists helps the user in debugging experimental procedures. As a convenience, a menu with a list of command mnemonics can be displayed at the top of the screen. Along with the command, the communication port is displayed to show to which instrument the command is directed.

The mnemonic definitions are stored in an ASCII file. The format for a mnemonic definition is

MNEMONIC, COMMUNICATION PORT,
NUMBER OF ARGUMENTS, ARGUMENTS

Each mnemonic definition takes one line of the file. The argument list is used to notify ARTS of the direction of communication (to the instrument, from the instrument) and the type of the argument (string, floating-point number, integer, array of strings, array of floating-point numbers, or array of integers). If the argument is a valid string or number, it is sent to the instrument as in the following example:

MLS, 1, 1, 301

In this example, the mnemonic MLS is used to send one parameter, 301, to a robot using communication port number 1. If data are to be returned from an instrument, the argument consists of a question mark (?) followed by a one-letter specifier (F, floating point; D, integer; S, string; AS, array of strings; AF, array of floats; or AD, array of integers) to determine the type of data returned from the instruments. The following entry

NMEAS, 2, 2, NMEAS, ?D

is used to send one argument, NMEAS, to a spectrophotometer and to receive data from the spectrophotometer via communication port 2. The data from the instrument are evaluated as an integer. When the type consists of an array, the number of members immediately precedes AS, AD, or AF. For commands that need to have parameters evaluated during execution, an asterisk (*) precedes the type description. One such command is GRIP, where the tension follows the mnemonic name. The file entry would be

GRIP, 1, 2, 100, *F

where two arguments are to be sent to the robot through communication port 1. The first argument is a 100 and is used by the robot to issue a grip command. The robot expects a second argument to be sent specifying the tension of the grip. This parameter will be evaluated by ARTS as a floating-point number. A final example for the four types of commands given in Table III consists of sending a command and a parameter and then recieving data from the instrument, as in

VALUE, 2, 3, VALUE, *D, ?F

When the command "VALUE(750)" is issued in ARTS, the string "VALUE" is sent to a spectrophotometer. The 750, representing the wavelength in nanometers, is evaluated as an integer and is transmitted. The transmission of these commands causes the spectrophotometer to return the absorbance at 750 nm to ARTS. The data received are evaluated as a floating-point number. The mnemonic definitions allow ARTS to communicate with laboratory instruments and notify the user when an incorrect number of parameters have been entered.

COMMUNICATION WITH THE ZYMATE I SYSTEM

The ZYMATE I system is a laboratory robotic system manufactured by the Zymark Corp. The ZYMATE I system allows communication to external computers through an

Table IV. Partial Listing of the Zymate I Control Program

```
1000  VALUE = SDATA.IN    ! WAIT FOR NUMBER FROM
                            ARTS
      GOTO VALUE          ! GOTO CORRECT LINE NUMBER
1     CLEAR.GP.HAND       ! ROBOT POSITION
      GOTO 1000           ! GO BACK TO TOP
2     HAND1POS1           ! ROBOT POSITION
      GOTO 1000           ! GO BACK TO TOP
100   GRIP = SDATA.IN     ! GET VALUE FOR GRIP FROM
                            ARTS
      GOTO 1000           ! GO BACK TO TOP
101   WRIST = SDATA.IN    ! GET VALUE FOR WRIST FROM
                            ARTS
      GOTO 1000           ! GO BACK TO TOP
200   DATA.OUT = WEIGH    ! RETURN WEIGHT TO ARTS
      GOTO 1000           ! GO BACK TO TOP
201   DATA.OUT = ADC      ! RETURN A/D VALUE TO ARTS
      GOTO 1000           ! GO BACK TO TOP
205   VALUE = SDATA.IN    ! GET CLOCK NUMBER (1-4)
      DATA.OUT =          ! RETURN TIME FROM CLOCK TO
        CLOCK(VALUE)        ARTS
      GOTO 1000           ! GO BACK TO TOP
```

RS423 ASCII link.[11] The data sent between the external computer and the ZYMATE I system must be packaged according to preset specifications. The ZYMATE I controller can interpret four types of commands: TRANSMIT, RECEIVE, ACKNOWLEDGE, and NOT ACKNOWLEDGE. The TRANSMIT command is used to send numeric information between the two systems. The RECEIVE command is used by the ZYMATE I system to notify the external computer that it is ready to receive data. The ACKNOWLEDGE and NOT ACKNOWLEDGE commands are used by the ARTS and ZYMATE I systems to notify each other that a transmission was received correctly or incorrectly. The format for the packaging of the message is as follows:

one character to specify type of message (T, R, A, N)
one-character identification code set by ZYMATE I system
two characters to specify length of numeric data
actual numeric data
two characters to specify a check sum
carriage return
line feed

The check sum is optional and can be replaced with two spaces if not used. The ZYMATE I system uses identification numbers to verify correct transmission of data. The identification number must be sent back to the ZYMATE I system with the messages from the external computer. Since the identification number changes, the software on the external computer must be able to interpret the messages and extract the correct identification number. This identification number must then be repackaged and sent back to the ZYMATE I system.

The ZYMATE I system is used in a slave mode by ARTS. A program running on the ZYMATE I system allows ARTS to control the working environment of the robotic system. A portion of the program is given in Table IV. The SDATA.IN command is a predefined module that returns a value from an external computer. When the SDATA.IN command is executed by the ZYMATE I system, a RECEIVE message is sent to the external computer. ARTS packages a command in a TRANSMIT message and sends it to the ZYMATE I controller. After the TRANSMIT command is received, an ACKNOWLEDGE message is sent back to ARTS. The value sent to the ZYMATE I system is stored in a variable (VALUE). A GOTO command transfers control to the line number specified in VALUE. If the command is a position, like CLEAR.GP.HAND or HAND1POS1, the robot is given the position command and control is transferred back to the beginning of the program. If the command requires a parameter, like GRIP or WRIST, the ZYMATE I controller waits for the next value to be sent. The parameter is combined with the command and executed. Control is again transferred to the start of the program. When a command returns data to ARTS, it is through a DATA.OUT command. Upon executing a DATA.OUT command, the ZYMATE I system sends a TRANSMIT message and waits until a proper ACKNOWLEDGE message is received from ARTS. Examples of using a DATA.OUT command are with the WEIGH and ADC commands. The CLOCK command is an example of a value being sent and returned by using both of the data-transfer commands.

PACKAGING DATA FOR TRANSMISSION

ARTS allows the operator to specify the transmission masks needed for each instrument. The masks are used by ARTS to interpret messages from instruments and to package messages to be sent to instruments. The masks need only be set once and are stored in a file that is read at initialization of ARTS. Usually the messages sent to instruments require little or no packaging before they are sent. The ZYMATE I system requires an extensive packaging scheme and will be used as an example. The RECEIVE message sent from the ZYMATE I system has an identification code that is needed for later transmission. With the following mask entered for the RECEIVE command

```
R%1s00   \r\n,CODE
```

ARTS knows that the variable CODE will be needed for later communication with the ZYMATE I system. The "%1s" is used to specify that the length of CODE is one character long. The mask for the TRANSMIT message would be represented as

```
T%1s%2s%s   \r\n,CODE,LENGTH,VALUE
```

In this case, three variables are needed to specify the proper communication scheme. The CODE message is needed to send an ACKNOWLEDGE message back to the ZYMATE I system. The LENGTH always occupies two characters in the message. The VALUE variable is used to hold the values sent to and from the ZYMATE I system. VALUE is used internally by ARTS to store in user-specified variables or to make decisions. When the ZYMATE I system sends a TRANSMIT message, it requires that an ACKNOWLEDGE message be sent back using the CODE sent with the TRANSMIT message. The mask for the ACKNOWLEDGE command takes the form

```
A%1s00   \r\n,CODE
```

The NOT ACKNOWLEDGE message sent by the ZYMATE I system contains no CODE or other information. ARTS represents this message in the following form:

```
N000   \r\n
```

When the ZYMATE I system sends or receives this message, it immediately restarts the current transmission. These masks are needed by ARTS to communicate with the ZYMATE I system, although other systems may only require a TRANSMIT message without the RECEIVE, ACKNOWLEDGE, and NOT ACKNOWLEDGE commands. With these instruments, the following TRANSMIT mask would be the only mask needed

```
%s\r\n,VALUE
```

where the message is sent to and from the instrument followed by a carriage return and a line feed. The user set masks allow ARTS to communicate with various instruments regardless of the complexity of the communication scheme of the instrument.

COMMUNICATION WITH EXTERNAL PROGRAMS

The ability to work in conjunction with other programs increases the power and flexibility of ARTS. Under the MS-DOS operating system, programs can give temporary control to other executable software. With this operating scheme, ARTS can be used in ways other than those originally devised.

ARTS can be an extension of other software in either a master or slave mode. As master, ARTS can call on other software to perform certain tasks. This implementation requires the same format as the operating system except the command must be preceded by "RUN". The RUN command signals ARTS to allow the external software to have temporary control. This method is convenient when data or commands must be manipulated in ways other than those originally planned by this control language. Executable software written in algorithmic languages, such as Fortran, Pascal, or Algol, can be called to perform complex calculations such as sorts or optimizations.

In a slave mode, external software can call on ARTS to perform manipulative tasks. The format of the calling procedure depends on the source language of the external software. This arrangement is important when the robot and instruments are sensory extensions of "intelligent" software, such as expert systems. The external software can ask ARTS to perform a certain task. The data returned from the instruments can be used to help with decision-making processes in the external software. This example is analogous to the present use of robots in which the robot performs a task for the researcher who then analyzes the resulting data.

PARAMETER PASSING

Parameter passing to external programs can be evoked in several ways that depend upon the programming language and style of the external software. The most familiar method of entering data is through the standard input device, the keyboard. Queries are displayed on the screen, and data are entered by the user. This type of data processing is the simplest of all methods to implement.

The standard method of entering data can also be redirected from files. In this case, the data must exist in the file in the same order as requested by the program. Instead of monitoring the keyboard, the program reads the data directly from the file. In this method, the program name and the data file are entered on the same line with a less than (<) symbol between the two names:

```
program_name < input_file
```

Information normally echoed on the screen can also be redirected to files by including a greater than (>) symbol in front of a file name on the command line:

```
program_name > output_file
```

Redirection of both input and output data can occur as in the following example:

```
program_name < input_file > output_file
```

This type of data redirection occurs implicitly and is controlled by the operating system, not the program.

Another method of communicating data to external programs is through explicit manipulation of files. The read and write statements must be explicitly stated in the source code of the program. An example of this procedure in BASIC for input would be (without line numbers)

```
INPUT#1,D,...
```

where "1" represents a file number used in an OPEN statement and "D,..." represents a list of variables used to hold data. A similar expression for output would be

```
PRINT#1,D,...
```

where "1" and the "D,..." have the same definitions as in the INPUT statement. This type of data manipulation also requires the program to open and close the data file.

Some computer languages allow command-line processing of data. Command-line processing occurs when the program name and data are entered on the same line. An example of this procedure can be seen in a standard copy command:

```
COPY FILE1 FILE2
```

In this example, the program reads the file names from the command line and performs the copy procedure. Assembly and C programming languages allow command-line processing. This type of data manipulation must be specifically added in the external software. In C programming, command-line processing is explicitly stated in the declaration of the main routine. In Assembly programming, the command-line parameters must be accessed according to where they were placed in random-access memory (RAM) by the operating system.

TECHNICAL DETAILS

ARTS was written in the C programming language and runs on a Leading Edge microcomputer under the MS-DOS version 2.11 operating system. The Leading Edge microcomputer is fully compatible with IBM and IBM-compatible hardware and software. The microcomputer comes equipped with 640K RAM, one floppy-disk drive, and a 10-MB Winchester hard disk. The memory requirement of ARTS is only 128K. If other programs are executed in the ARTS environment, the memory requirement increases to account for execution of the additional software.

EXPERIMENTAL SECTION

The routines in Appendix A and the program in Appendix B (both appendixes are included in the supplementary material) show some of the power and flexibility of ARTS at controlling a robot, a UV–vis spectrophotometer, and an external program. The routines used in ARTS are stored in separate files, allowing them to be easily modified. The external program in Appendix B was written in the C programming language and demonstrates both command-line processing and direct access of files. The main routine in this set is located in the file names TITRATE.SUB (Table V). This routine can be executed in two ways via ARTS. The first way would be to give the command "GOSUB TITRATE.SUB" while in ARTS. The second would be to give the command "ARTS TITRATE.SUB" while under the control of the operating system. After completion of the routine, the first implementation would end with ARTS still in control. The second implementation would automatically stop execution of ARTS when the routine was finished and would be the best way for external software to call upon ARTS.

This collection of routines contains all the needed commands to perform a complexometric titration of a metal cation in an aqueous solution. In this demonstration, a solution of copper sulfate is titrated with ethylenediaminetetraacetic acid (EDTA). The absorbance values are read at 724 nm, which corresponds to the absorbance maximum for the Cu^{2+}–EDTA complex. A break, corresponding to the end point, occurs in the curve formed by plotting the absorbance of the solution vs. volume of titrant (Figure 1). The points preceding and following the break form straight lines that intersect at the end point of the titration.

The titration starts with the routine TITRATE.SUB by setting variables needed for the titration. Variables can be set by explicitly equating them in the routines or by querying

Table V. Main Subroutine for Use in Titration

```
REM       TITRATE.SUB
REM
REM ROUTINE TO TITRATE AN UNKNOWN SAMPLE
REM
   TITRE = 0.0              ! SET TITRANT VOLUME TO 0
   LAMBDA = 724             ! SET WAVELENGTH FOR MEA-
                              SUREMENT
   LOOPS = 13               ! SET NUMBER OF ITERATIONS
   INC = 0.15               ! INCREMENT FOR TITRANT
   TIP.INDEX = 1            ! SET DISPOSABLE SYRINGE TIP
                              INDEX
   INPUT "ENTER CONCENTRATION OF TITRANT :
     ",CONC.TITRANT
   INPUT "ENTER VOLUME OF UNKNOWN SAMPLE :
     ",VOL.UNK
   INPUT "ENTER VOLUME OF BUFFER TO ADD :
     ",VOL.BUFF
   INPUT "ENTER VOLUME OF WATER TO ADD :
     ",VOL.H2O
   INPUT "ENTER TEST TUBE NUMBER OF THE SAMPLE
     : ",UNK.INDEX
   INPUT "ENTER TEST TUBE NUMBER OF THE BUFFER
     : ",BUFF.INDEX
   OPEN "O",1,"UN           ! OPEN OUTPUT FILE
     KNOWN. DAT"
   PRINT #1,(VOL.           ! STORE INITIAL VOLUME
     UNK+VOL.BUFF+
     VOL H2O)
   PRINT #1,(LOOPS+1)       ! STORE NUMBER OF MEA-
                              SUREMENTS TAKEN
   GOSUB BLANK.SUB          ! TAKE A REFERENCE
   GOSUB UNKNOWN.           ! ADD SAMPLE AND BUFFER
     SUB
   GOSUB SPECSAMP.          ! TAKE INITIAL READING
     SUB
REM START TITRATION
   FOR I=1 TO LOOPS
   TITRE = TITRE +          ! TITRE IS AMOUNT OF TI-
     INC                      TRANT ADDED
   GOSUB ADD.SUB            ! ADD TITRANT AND MIX SAM-
                              PLE
   GOSUB SPECSAMP.          ! MEASURE ABSORBANCE
     SUB
   NEXT I
   CLOSE 1
REM RUN PROGRAM TO ANALYZE DATA
   RUN DIRECT.EXE UNKNOWN.DAT ENDPOINT.DAT
REM OPEN ENDPOINT.DAT TO READ THE ENDPOINT
   OPEN "I",1,"ENDPOINT.DAT"
   INPUT #1,END             ! READ DATA FROM FILE
     POINT
   CLOSE 1
   PRINT "THE ENDPOINT WAS AT ";ENDPOINT;"ml" CO
     NC.UNK = ENDPOINT*CONC.TITRANT/VOL.UNK
   LAMP 0                   ! TURN SPECTROPHOTOMETER
                              LAMP OFF
   STOP.SPEC                ! TURN SPECTROPHOTOMETER
                              OFF
   STOP.ROBOT               ! TURN ROBOT OFF
   QUIT                     ! LEAVE ARTS
```

the user for information using INPUT or INKEY$. The data file UNKNOWN.DAT is opened, and the initial volume of the solution is stored. The routine BLANK.SUB is called to fill the spectrophotometer cell with water and to take a reference scan. The next step involves combining the sample and buffer in a test tube. After the initial absorbance is measured, the titration begins by adding the EDTA in increments. The solution absorbance is taken and recorded after each addition. The method of detection used does not require points to be taken at the end point, but rather at points on both sides of the end point. TITRATE.SUB does not monitor the absorbance to determine if the end point has been reached, although

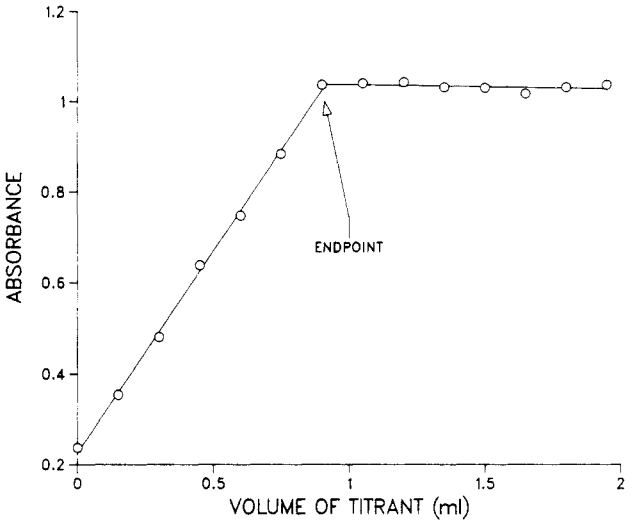

Figure 1. Characteristic titration curve is formed when absorbance is plotted against the volume of added titrant. This curve represents the titration of a copper sulfate solution with a 0.1004 M EDTA solution. The solution was buffered with an acetic acid/acetate buffer (pH 5). Absorbance values correspond to the Cu^{2+}–EDTA complex at 724 nm.

Table VI. Test Example To Determine the Concentration of Copper Ions in an Aqueous Solution Using Three Duplications

	trial 1	trial 2	trial 3	av
end point, mL	0.9188	0.9080	0.9134	0.9134
concn, M	0.0922	0.0912	0.0917	0.0917
SD in av concn			0.0005 M	
coefficient of variation			5.89%	
95% confidence level of av concn			0.0013	

it could with minor modifications. After the titration, the file is closed and DIRECT.EXE is called upon to read the file UNKNOWN.DAT, analyze the data, and store the result in ENDPOINT.DAT. The data are read from ENDPOINT.DAT and reported to the user. The routine TITRATE.SUB ends by shutting off the lamp in the spectrophotometer and returning to the interactive mode of ARTS. Each of the routines called by TITRATE.SUB can also call other routines to perform tasks.

The program DIRECT.EXE uses the absorbance and volume data stored in the file UNKNOWN.DAT to calculate the end point. Before the end point is calculated, the absorbance values are corrected for dilution effects caused by the addition of the titrant. The corrected absorbance values form two lines that intersect at the end point of the titration.[12] The difficulty of this experiment is in determining to which line each data point belongs. This program treats this problem by setting the first three points to the first line and the rest of the points to the second. Regression lines are formed through the two sets of data points, and the standard deviation of the residuals of the lines are calculated and summed. Shifting one point at a time from the second line to the first and recalculating yields a plot of the sum of the standard deviation of the residuals vs. the last point in the first line that shows a minimum in the curve corresponding to the best two lines through the data (Figure 2). The regression lines can be used to determine the end point of the titration. This process is performed in the program by the routine INTERSECTION. All the functions performed in DIRECT.EXE could have been performed in ARTS, although they are better implemented with an algorithmic language like C.

The results of three titrations using these procedures are given in Table VI. In each of the titrations, a 1.0-mL sample of a copper sulfate solution is titrated, in 0.15-mL increments,

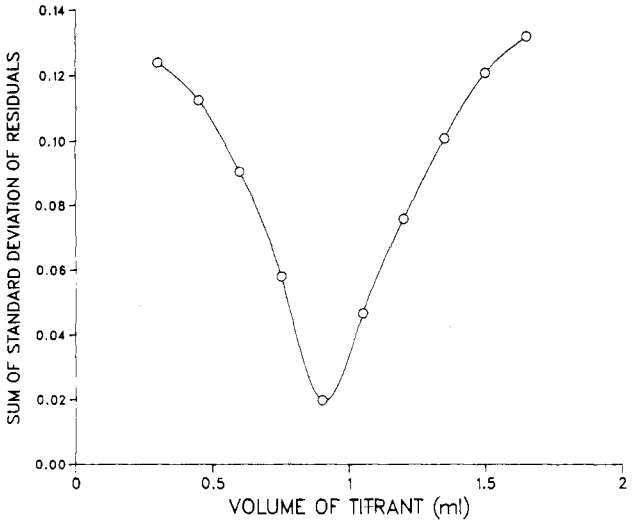

Figure 2. Shifting the data points from the second line to the first and calculating the standard deviation of the residuals for each line yields a plot of the sum of the standard deviations vs. the last point used to represent the first line that results in a mininum occurring when proper classification of data points to the lines exists.

with 0.1004 M EDTA. The solution is buffered with 3.0 mL of a acetic acid/acetate buffer solution (pH 5.0) and diluted to a total volume of 8.75 mL. The absorbance values are recorded at 724 nm. From these titrations, the concentration of the copper solution was determined to be 0.0917 M with a standard deviation of 0.0005 M.

CONCLUSION

The need exists for improved control of laboratory instruments through external software. ARTS is a powerful software tool for the analytical laboratory capable of controlling programmable instruments. The extensive commands give the researcher maximum control over the programming environment. The error-checking ability of ARTS reduces the time required in the debugging phase of software development.

The ability of ARTS to work in conjunction with external software increases the versatility of laboratory control. In a slave mode, ARTS can act as a sensory extension. In a master mode, ARTS can direct external software to perform tasks for which it was not originally designed. The researcher is no longer limited by the instrument control software since external software can be incorporated into ARTS.

ACKNOWLEDGMENT

This research was supported by the National Science Foundation under Grant CHE-8415295. Grateful acknowledgment is given to Peter Harrington, Sally Ekert, and Jiann-Rong Lee for their helpful suggestions.

Supplementary Material Available: Routines (Appendix A) and program (Appendix B) for controlling a robot, a spectrophotometer, and an external program through ARTS (22 pages). Ordering information is given on any current masthead page.

REFERENCES AND NOTES

(1) Owens, Grover D.; Eckstein, Rodney J. "Robotic Sample Preparation Station". *Anal. Chem.* **1982**, *54*, 2347–2351.
(2) Woo, Nancy H.; Rosenberg, Arthur S. "Robots in the Laboratory: Part II". Dessey, Raymond, Ed. *Anal. Chem.* **1983**, *55*, 1234A–1240A.
(3) Bellus, Peter "Robots in the Laboratory: Part II". Dessey, Raymond, Ed. *Anal. Chem.* **1983**, *55*, 1240A–1242A.
(4) Dittenhafer, Mark L.; McLean, James D. "Robots in the Laboratory: Part II". Dessey, Raymond, Ed. *Anal. Chem.* **1983**, *55*, 1242A.
(5) Beni, G. "Robotic Measurements in Liquids". *J. Electroanal. Chem.* **1982**, *140*, 137–140.
(6) Papas, A. N.; Alpert, M. Y; Marchese, S. M.; Fitzgerald, J. W.; Delaney, M. F. "Evaluation of Robot Automated Drug Dissolution Measurements". *Anal. Chem.* **1985**, *57*, 1408–1411.
(7) Kramer, G. W.; Fuchs, P. L. "Automation in Organic Synthesis". *BYTE* **1986**, 263–284.
(8) Brosemer, J.; Liscouski, J. "Computers and Robotics: A Synergistic System". *Am. Lab. (Fairfield, Conn.)* **1986**, *18*(19), 80–83.
(9) Binkley, D. P. "A System for Laboratory Automation". *Am. Lab. (Fairfield, Conn.)* **1986**, *18*(2), 68–73.
(10) Brown, P. J. *Writing Interactive Compliers and Interpreters*; Wiley: Chichester, UK, 1979; pp 112–121.
(11) Zymate System Instruction Manual, H1-H7.
(12) Ringbom, Anders "Complexation Reactions", Treatise on Analytical Chemistry. Kolthoff, I. M., and Elving, P. J., Eds.; Interscience Encyclopedia: New York, 1959; Vol. 1, pp 543–628.

COMPUTER SOFTWARE REVIEWS

Molecular Editor

JOHN S. WINN
Department of Chemistry, Dartmouth College, Hanover, New Hampshire 03755

Received March 27, 1987

There are almost as many molecular model programs as there are brands of computers, and one can spend from next to nothing to many thousands of dollars for them. Generally, the utility of such programs is a very strongly rising function of something like the product of the software and hardware prices. This program, called Molecular Editor, seems to be an exception. It is trivially priced for its power, and, although specific for the Macintosh series of computers, it demands only a modest hardware investment. Developed as part of the microcomputing program at Drexel University under the supervision of Professor Allan Smith, the program was written with instructional applications in mind but sensibly retains the power of many research-grade programs.

The program is supplied with an adequate but brief instruction manual. In keeping with the Macintosh programming philosophy, however, any regular Macintosh user can start the program and begin to use much of it "intuitively" without training or detailed reading of the manual. Such users will also likely look for, find easily, and be rewarded with an exceptionally good set of on-line help messages. One can use