Implementing communication between Windows PCs and test equipment using RS-232 and Borland C++ Builder

AXEL MONTEIRO and TIMOTHY R. JORDAN
University of Nottingham, Nottingham, England

Modern experiments in the behavioral sciences frequently employ several items of electronic equipment such as computers, monitoring devices, stimulus presentation equipment, and response collection systems. In many cases it would be advantageous for these items to communicate directly with each other. Such communication may facilitate greater automation of experiments (i.e., reduced experimenter influences during the experiment), more precise experiment control (i.e., superior timing and synchronization capabilities of electronic devices), and greater accuracy of data collection (i.e., reduced ambiguity of participant responses). Many electronic experiment devices already provide external interfaces through which communication with other devices can be implemented. The most common is based on the RS-232 protocol, which is also found in all standard PCs. Therefore, Microsoft Windows based computers can be programmed to control experiments by communicating directly with electronic experiment devices. We show how to implement this RS-232 interconnection between devices and a Windows PC using currently available software tools.

Several computer-based systems for automatically controlling experiments have been presented by previous researchers (e.g., Cohen, MacWhinney, Flatt, & Provost, 1993; Dutta, 1995; Haxby, Parasuraman, Lalonde, & Aboud, 1993; Hunt, 1994; Schneider, 1988; see also Harley & Loftus, 2000; Vaughan & Yee, 1994). Many of these systems provide powerful environments for designing and automatically running entire experiments. However, in many cases, researchers may wish to include two or more computers in experiments and/or other electronic experiment devices (e.g., eye-tracking devices, VCRs, audio and visual data recording devices, etc.). To automatically control experiments of this kind, a way of implementing digital communication with experiment devices and/or multiple computers is required.

Modern Windows PCs can be programmed to communicate with other computers and other electronic devices using the RS-232 protocol. This communication protocol specifies the configuration of physical communication interfaces and parameters for digital serial communication and is incorporated into many electronic devices. All Intel PCs have at least one, and usually two, RS-232 communication ports located at the rear of the machine. They are typically referred to in the PC's hardware specifications as COM1 and COM2. The physical interface on the PC is also sometimes labeled with the

symbol 10101. The RS-232 protocol has been in use for several decades, and detailed information concerning its use is readily available in engineering textbooks and on the Internet. However, for the purposes of communicating with external electronic devices from within a Windows application, a basic understanding of RS-232 is sufficient (see Appendix A).

The Borland Delphi and C++ Builder Integrated Design Environments (IDEs)

In recent years, the development of computer software has been greatly facilitated by the increasing availability and sophistication of "visual" programming environments. These visual environments seek to simplify application development by (1) providing programming tools to aid design, development, and testing; (2) providing libraries of expertly designed and tested code modules (referred to as *components* in the Borland IDEs) that provide ready-made functionality for many common programming tasks; (3) encouraging support from third-party software developers in the form of additionally available components, providing additional pieces of functionality, often for less common (but equally useful) programming tasks.

A number of different visual programming environments are currently available. The most common include Microsoft's Visual Basic, Visual C++, and Visual J++ (based on BASIC, C++, and Java); REAL Software's REALBasic (based on BASIC); and Borland's Delphi, C++ Builder, and JBuilder (based on Pascal, C++, and Java; see Appendix B). Of these, the Borland C++ Builder IDE provides an ideal system for the development of ex-

Correspondence concerning this article should be sent to A. Monteiro, Royal Bank of Scotland Group, Clearing Live Support, A1 West Goodmans Fields, 74 Alie St., London E1 8HL, England (e-mail: axel.monteiro@rbs.co.uk).

periment control applications since it combines the power and efficiency of the C++ programming language with the ease-of-use benefits of visual programming. Borland C++ Builder is available for Windows XP, Windows 2000/ME/NT, and Windows 98.

Objects in the Borland C++ Builder IDE have two kinds of attributes that can be configured at design time: *properties* and *events*. Properties specify the visual appearance of the object when the application runs (if it is visible) and the functionality the object provides. Events define how the object will respond if certain actions occur. These can be user actions, such as a mouse click, or actions from other objects. The property values and event responses are specified in the Object Inspector window of the IDE. When an object is selected in the IDE, its properties and events are displayed in the Object Inspector window (Figure 1).

The Borland IDE comes with a wide selection of different components that can be used in applications. These are organized in *palettes* containing components of related properties and abilities (Figure 1). To include a component in the application, it is simply selected from the appropriate palette and placed on the form. If the component is to form part of the application user interface, then its size and location on the form are set by simply dragging with the mouse. Once placed on the form, the component's properties and its event behavior can be specified using the Object Inspector (Figure 1).

Essentially, the IDE provides a visual representation of the program code in the application. When an object is placed on a form in the application, that object's functionality can be accessed within the code of the application. For example, when a button object is placed on a form, its properties (such as its size, text, color, etc.) and its ability to respond to user actions (such as a mouse click) can be specified using the Object Inspector. C++ code defining what the button will actually do when it is clicked is then linked to the relevant event of the object.

With Borland C++ Builder, a basic experiment control application can be built relatively easily. Components supplied with the Borland IDE simplify the programming associated with collecting participant input, managing trial scheduling, and outputting data files for statistical analysis.

The TMS Async32 Communication Component

Access to the serial communications port of a PC running Microsoft Windows must be mediated by the operating system. Therefore, programming applications that involve serial communication necessarily involves interaction with the Windows Application Programming Interface (API). The Windows API is complex and best suited to programmers with substantial experience of Windows application development. However, the task of manipulating the Windows API for serial communication is abstracted to a much simpler visual interface by a freely

Figure 1. The Borland C++ Builder 5.0 Integrated Design Environment.

available Borland component. The TMS Async32 component is available for Borland Delphi and Borland C++ Builder and greatly simplifies the task of programming applications involving PC serial communications. Once downloaded from the TMS Web site and integrated into the C++ Builder IDE (or Delphi IDE), it can be used just like any of the standard Borland components.

The TMS Async32 component properties facilitate specification of communication parameters such as data transfer speed, buffer sizes, data word size, and so forth, and the object itself provides a set of routines for managing communication while the application is running (i.e., initializing and opening a communications port, reading from, and writing to the communication line, detecting when new data has arrived, etc.). Definitions of all of the properties and routines associated with the TMS Async32 component are provided in a help file that is supplied with the component download package.

An Automated Experiment Incorporating RS-232 Communication

We have used Borland C++ Builder and the TMS Async32 component to create automated experiment systems for several experiments involving communication between multiple PCs and electronic devices. However, to illustrate the implementation of RS-232 communication in the context of an experiment control application, let us consider a simple system architecture involving a single Windows PC and one external experiment device. In this system, stimuli are recorded in a random order onto digital videotape. The stimuli are presented in the experiment via a digital VCR that outputs to a standard television screen. The VCR is controlled by the Windows PC via RS-232 communication in such a way that one stimulus is played from the tape for each trial. The application that controls playing and stopping the tape for each stimulus also collects a response to each stimulus from the participant. Response collection is facilitated by the PC's monitor and keyboard. Hence, for each stimulus presentation, participants watch the television and then make a response on the PC's keyboard as they view the monitor. The control application waits for the participant to make a response, then records the response, and then plays the next stimulus from the tape.

Implementing RS-232 Communication Between a Windows PC and an External Device

Once the PC and external device have been connected correctly, parameters for communication must be set. Placing the TMS Async32 component on a form in the application facilitates the setting of communication parameters using the Object Inspector. Baud rate, parity, and so forth can be set by entering values in the appropriate fields of the Object Inspector. These are typically specific to the external device and should be listed in the device manual or obtained from the device manufacturer. If the communication is to occur between two PCs, these values can be chosen by the programmer. Pull-

down lists in the Object Inspector provide several of the most appropriate alternatives for each parameter.

The Device Name parameter denotes which serial port is to be used for the communication. The speed of the communication line is set using the baud rate parameter. Digital communication speed is defined in bits per second (bps). Electronic devices are typically preconfigured to use a specified baud rate. However, depending on the specific device, a range of baud rates may be available. Two Windows PCs, for example, can communicate via RS-232 at several possible baud rates. Choosing the fastest baud rate available to both the controlling PC and the external device is generally advisable. For Windows PCs, the maximum available baud rate is dependent on the specific serial communication logic incorporated into the system. For modern PCs, this is usually at least 128,000 bps. Hence, for communication between two modern Windows PCs, RS-232 communication can be achieved at this speed.

The Name parameter defines the name that will be used to reference the TMS Async32 component within the C++ code. The remaining parameters deal with error checking and the specifics of sending and receiving digital information. A detailed description of each parameter and how it is used is beyond the scope of this article. More information can easily be found on the Internet or in any basic networking textbook. However, it is not necessary to understand the functions of the communications parameters to develop a working system. What is vital is that all of the communication parameter settings (except Device Name and Name) are the same on the PC and the external device. For example, if the external device is configured to a baud rate of 9,600 bps, then the baud rate setting within the TMS Async32 component on the PC must also be set to 9,600 bps. If this is not the case, communications will be incorrectly interpreted, which will at best lead to "garbage" communications and at worst to unpredictable or erroneous device behavior.

The TMS Async32 component represents a C++ class. Placing the component on a form instantiates an object of the class that is referenced through the form. Member functions of the object can then be called to access the functionality of the object. By default, the instantiated object is named VaComm1, although the Object Inspector can be used to change this at design time to a more meaningful name. The default name for a form object is Form1. Therefore, the general form of syntax for accessing the TMS component functionality (with default object names) is

Form1->VaComm1->member-function.

Before the serial port can be accessed by the application, Windows must be notified that the port is to be used. This initialization is achieved through the TMS component member function Open [i.e., Form1->VaComm1->Open(); must be present in the code before any other calls to VaComm1 can be made]. Once the port has been specified as open, full duplex communication

with the external device is available to the application via the serial port. That is, messages can be sent and received simultaneously (see Appendix A).

In general, communication takes place one byte at a time. The manual for the device should specify (usually in hexadecimal) commands that the device can understand along with the messages that the device may send back. In our implementation, the *play* command for the digital tape deck is the hexadecimal number 2C. In a C++ string, hexadecimal numbers are denoted by a preceding \x. Hence, to instruct the tape deck to begin playing the tape, the value \x2C must be passed to the WriteChar member function of the TMS Async32 component. The TMS member functions WriteChar and ReadChar facilitate sending and accessing received data, respectively. The syntax for this command is therefore Form1->VaComm1->WriteChar("\x2C");. typically, the device will acknowledge receiving the command by returning a one byte OK message [which can be read with ch = Form1->VaComm1->ReadChar();, where ch is some suitably sized variable, such as a variable of type char]. In this way, a PC using single-byte messages can control an external device.

However, there may be situations where messages need to be greater than one byte. For example, in our implementation, the tape deck can be interrogated for its current time code (i.e., how many hours, minutes, seconds, and frames have elapsed since the beginning of the tape). Here a one-byte *request timecode* message (\x0F) is sent to the tape deck. The tape deck responds by sending back its current time code in an eight-byte message (rather than the usual single-byte response). These eight bytes represent eight ASCII characters (in the range 0 to 9), which in pairs specify the values for hours, minutes, seconds, and frames. This kind of multibyte response is handled by simply calling ch = Form1->VaComm1->ReadChar() multiple times and building up a string as each char is read. Note that the speed at which the bytes are read is not synchronized with the speed at which they arrive. It is therefore important to ensure that the VaComm1 object has an appropriately sized receive buffer. This can be set at design time using the Object Inspector. In addition, the task of receiving and evaluating multibyte responses is simplified if the sending device can structure the bytes into a null-terminated string (i.e., adding a \0 null character to the end of the byte sequence). If the multibyte message is structured in this way, the

TMS Async32 component member function ReadText can be used instead of calling ReadChar multiple times. ReadText gets multiple bytes from the buffer until a null character is encountered. Just as for read char, the bytes are returned as the return value from ReadText. Thus the syntax for ReadText (with default object names) is str = Form1->VaComm1->ReadText(), where str is a string object or a suitably sized array of char. For devices that can receive null-terminated string messages, TMS Async32 component provides a corresponding WriteText member function.

A simple program to control a Sony DSR-20 digital tape deck may be seen at the bottom of the page.

Here the port is first opened. The tape deck is then instructed to begin playing, and a while loop is used to continually check the current time of the playing tape until some predetermined stop time is reached [i.e., the definition of the function CheckStopTime(szStopTime) is not shown. However, it basically takes a pointer to a string as an input parameter and requests the tape deck's current time code. It checks the returned eight-byte time code against the input string and returns "true" if they match or false if not]. When bStopTimeReached becomes true, the while loop exits and the subsequent WriteChar command stops the tape deck. Note that, in a real implementation, it may be desirable to execute this procedure within its own thread (since the while loop prevents any other code from executing until it exits).

This particular implementation is concerned with controlling a digital VCR. As such, collecting and evaluating time codes is time-critical. However, since incoming messages are stored in the TMS receive buffer, there is no danger that a message will be lost as long as the buffer is emptied frequently enough (reading an item removes it from the buffer). Thus it is important that the buffer size be set sufficiently large to hold all incoming data between emptying. Since time codes change every 40 msec (i.e., at 25 frames per second), the code above is adequate as long as the while-loop execution time is below 40 msec.

Of course, the actual commands used to control the digital tape deck in our application are specific to the make and model of the tape deck that we used. However, the structure and management of communication using the TMS Async32 component within the application is typical for an automated experiment application involving at least one external device. The source code for the application may be of use to any researchers wishing to

```
Form1->VaComm1->Open(); // Mark the serial port as "open"
Form1->VaComm1->WriteChar("\x2C") // Tell the tape deck to play
bool bStopTimeReached = false; // Declare a stop flag
while(!StopTimeReached) // Keep checking time codes until it's time to stop
{
    StopTimeReached = CheckStopTime(szStopTime);
}
Form1->VaComm1->WriteChar("\x1A"); // 1A is the command for "stop"
```

implement an automated experiment system involving external devices or communication between two or more PCs. The source code for our application is therefore available free of charge via e-mail from the authors.

REFERENCES

- COHEN, J., MACWHINNEY, B., FLATT, M., & PROVOST, J. (1993). PsyScope: An interactive graphic system for designing and controlling experiments in the psychology laboratory using Macintosh computers. *Behavior Research Methods, Instruments, & Computers*, **25**, 257-271.
- DUTTA, A. (1995). Experimental RunTime System: Software for developing and running reaction time experiments on IBM-compatible PCs. *Behavior Research Methods, Instruments, & Computers*, **27**, 516-519.
- HARLEY, E. M., & LOFTUS, G. R. (2000). MATLAB and graphical user interfaces: Tools for experimental management. *Behavior Research Methods, Instruments, & Computers*, **32**, 290-296.
- HAXBY, J. V., PARASURAMAN, R., LALONDE, F., & ABBOD, H. (1993). SuperLab: General-purpose Macintosh software for human experimental psychology and psychological testing. *Behavior Research Methods, Instruments, & Computers*, **25**, 400-405.
- HUNT, S. M. J. (1994). MacProbe: A Macintosh-based experimenter's workstation for the cognitive sciences. *Behavior Research Methods, Instruments, & Computers*, **26**, 345-351.
- SCHNEIDER, W. (1988). Micro Experimental Laboratory: An integrated system for IBM PC compatibles. *Behavior Research Methods, Instruments, & Computers*, **20**, 206-217.
- VAUGHAN, J., & YEE, P. L. (1994). Using PsyScope for demonstrations and student-designed experiments in cognitive psychology courses. *Behavior Research Methods, Instruments, & Computers*, **26**, 142-147.

APPENDIX A

RS-232 Serial Communication

RS-232 is a full duplex (i.e., devices can transmit and receive data at the same time), asynchronous (i.e., the communication is not synchronized by a clock signal), communication standard. Devices that communicate using RS-232 are specified as either DCE or DTE equipment. Typically DTE devices (such as a PC) connect through DCE devices (such as a modem) to other DTE devices. The main difference between the two is largely in the direction of each "pin."

The physical configuration of the RS-232 interfaces can be either 25 pins or 9 pins wide.

Table A1
25-Pin DTE and DCE RS-232 Pin Assignments

DTE		DCE	
25-pin male pin out		25-pin female pin out	
Pin 1	Shield Ground	Pin 1	Shield Ground
Pin 2	Transmitted Data (TD) output	Pin 2	Transmitted Data (TD) input
Pin 3	Receive Data (RD) input	Pin 3	Receive Data (RD) output
Pin 4	Request To Send (RTS) output	Pin 4	Request To Send (RTS) input
Pin 5	Clear To Send (CTS) input	Pin 5	Clear To Send (CTS) output
Pin 6	Data Set Ready (DSR) input	Pin 6	Data Set Ready (DSR) output
Pin 7	Signal Ground	Pin 7	Signal Ground
Pin 8	Carrier Detect (CD) input	Pin 8	Carrier Detect (CD) output
Pin 20	Data Terminal Ready (DTR) output	Pin 20	Data Terminal Ready (DTR) input
Pin 22	Ring Indicator (RI) input	Pin 22	Ring Indicator (RI) output

Note that in the table above each pin is also labeled as input or output. This denotes the direction of communication through the pin connection and does *not* indicate whether the connection is a pin or a hole. DTE and DCE devices can be connected together using a *straight through* cable (i.e., each numbered pin at one end of the cable goes to the same numbered pin at the other end).

The following table depicts the 9-pin male connector found on PCs. Most equipment used in experiments is not communications equipment and it therefore usually configured as DTE.

(Continued on next page)

APPENDIX A (Continued)

Table A2
9-Pin DTE (IBM PC)
RS-232 Pin Assignments

DTE	
9-pin male pin out	
Pin 1	Carrier Detect (CD) input
Pin 2	Receive Data (RD) input
Pin 3	Transmitted Data (TD) output
Pin 4	Data Terminal Ready (DTR) output
Pin 5	Signal Ground
Pin 6	Data Set Ready (DSR) input
Pin 7	Request To Send (RTS) output
Pin 8	Clear To Send (CTS) input
Pin 9	Ring Indicator (RI) input

Connection between two DTE devices is achieved using a null-modem cable (i.e., crossed transmit and receive pins). 9-pin RS-232 devices can be connected to 25-pin RS-232 devices with the use of an appropriate interface converter.

APPENDIX B

Visual Programming Integrated Design Environments

Borland C++ Builder 5, Borland Delphi 6 and Borland JBuilder are available from Borland Software Corporation, 100 Enterprise Way, Scotts Valley, CA 95066-3249: <http://www.borland.com/products>.

Microsoft Visual Basic 6.0, Microsoft Visual C++ 6.0 and Microsoft Visual J++ 6.0 are available from Microsoft Corporation, One Microsoft Way, Redmond, WA 98052-6399: <http://www.microsoft.com/catalog>.

REALBasic is available from REAL Software Inc., One Westlake Plaza, 3rd Floor, Austin, TX 78746: <http://www.realsoftware.com>.

The TMS Async32 Communications Component

The TMS Async32 Communications Component is available for Borland C++ Builder and Borland Delphi. It is available free from <http://www.tmssoftware.com/downloads>.

The Sony DSR-20 Digital Audiovisual Tape Deck

The Sony DSR-20 tape deck is available from authorized resellers of Sony Broadcast and Professional equipment. More information about the DSR-20 (including the programmer's manual and communication protocols) is available from Sony Broadcast and Professional: <http://www.sonybpe.com>.

(Manuscript received October 18, 2001;
revision accepted for publication September 6, 2003.)