infoComm INTERNATIONAL®

Roadmap to Control

Guidelines
for
Equipment Control Interface Development

Revision 1.1
May 2006

InfoComm International
11242 Waples Mill Road
Suite 200
Fairfax VA 22030 U.S.A.
800.659.7469
membership@infocomm.org
www.infocomm.org

Acknowledgments

The *Roadmap to Control* has been developed by the InfoComm International® Independent Programmers Council, which was created in 2004 in recognition of the critical role that programmers play in the design of AV systems.

The Independent Programmers Council takes its place as an integral part of the AV community represented through the InfoComm membership. Since its formation, it has interacted with other key member councils such as those representing AV systems integrators, independent design consultants, technology managers and manufacturers.

Through ongoing audioconferences and in-person meetings held at the annual InfoComm show and the annual Leadership Forum, the council has worked to identify topics and projects of importance to programmers everywhere.

The groundbreaking Roadmap to Control project follows in the footsteps of the Dashboard for Controls initiative developed through the Technology Managers Council.

The council especially thanks the following members for their contributions to this document:

Tom Barnett, MBA, CTS-D
Frank P. Damiano
Dave Johnsrud, CTS
Howard A. Nunes

Table of Contents

Table of Figures

1. Introduction

1.01 Audience and Purpose

The *Roadmap to Control* is about creating better, smarter audiovisual (AV) devices; devices that are easier to integrate, communicate with, and control. Its goal is to help manufacturers design better control interfaces for their products, which will make it easier for designers, integrators and programmers to select and integrate products.

Many professional control systems programmers would state that there are some devices that are feature-rich and relatively easy to communicate with and control. These are the devices that programmers and engineers recommend and specify often, and welcome as part of the projects in which we are involved. Unfortunately, most equipment does not fall into this category.

Programmers involved in the *Roadmap to Control* project are not trying to advocate *specific* protocols or standards, but simply see the benefit and promise of the predominant availability of “good” equipment.

Tangible benefits of better, smarter devices, which are easier to integrate, communicate with, and control include:

1) Lower design and integration costs
2) Faster and more efficient integration
3) Better features and controls
4) Better reliability, with fewer service calls and costs

These benefits translate into a better experience for all parties involved:

1) Consultants spend less time designing systems and selecting equipment
2) Integrators spend less time installing and configuring systems
3) Programmers spend less time programming
4) Consumers enjoy better, more functional, and more reliable systems
5) Manufacturers sell more products with fewer support calls

The *Roadmap to Control* is the result of the Independent Programmers Council’s initiative to catalog what works well. Members of the council hope to have a positive impact on the AV industry and the design of future equipment. Readers are strongly encouraged to distribute the Roadmap to those who are in a position to act on the topics covered.

1.02 Background

Many forms of audiovisual (AV) control interfaces have been used, and new forms are being implemented every year.[1] Relay control dominated the first generation of AV control systems in the 1970s, with infra-red control becoming main-stream in the 1980s. The Electronic Industries Association (EIA) RS-232 standard, and the closely related RS-422 and RS-485 standards became the interfaces of choice in the 1990s because of their bi-directional communication capability, which enabled true feedback to the user interface. Although some field experimentation with USB and Firewire control has been conducted, their adoption has not been widespread in the AV industry. Ethernet-based control is currently emerging as a way to handle the vast amount of information being transferred by media servers, conferencing equipment and other content-rich devices.

[1] For an excellent overview of the various control interface wiring and communications standards, see Chapter 11 of *The Basics of Audio Visual Systems Design*, Revised Edition, 2003, InfoComm International (formerly International Communications Industry Association, Inc.)

The desire to integrate a wider variety of systems and functions is leading to applications that require communication with a wider variety of equipment, including devices not traditionally used in audiovisual presentation or videoconferencing. Room temperature, occupancy status, access control, lighting, drapery, and many types of other controls and information are increasingly being incorporated into the AV system user interface.

Finally, the need to integrate a wide variety of equipment and sub-systems with centralized control has driven the demand for standardized communication and management methods. This demand is particularly acute in commercial, institutional and other non-residential markets, where professional IT managers are requesting standardization, but is no less relevant to today's automated home. It has resulted in the gradual adoption of TCP/IP over CAT5/6 cabling by the AV industry. Often called "convergence" by AV professionals, it is a natural progression, which has the potential to dramatically simplify installation, integration and control.

The purpose of an AV control system is to integrate multiple devices together to respond as a single unified system. The purpose of a control interface is to provide a point of control and feedback between controlled devices and the AV control system. Problems occur when the designer of a device being controlled does not adequately consider simplicity and usability in creating the control interface. This often results in a control interface and protocol that allow too little control to maximize a device's features, or are so complicated and arcane that they are difficult to implement.

1.03 Device Communication and Feedback

The variety of controllable AV equipment seems to have grown exponentially with each passing decade. Each device's control interface is a reflection of the features available in it. Simple transport controls (e.g., play, stop, and pause) are essentially one-way and can be accomplished with relays, or more typically with infrared (IR) emitters or one-way serial interfaces, such as Sony's Control-S interface. Although no status information is provided by one-way interfaces, in many instances none is required.

Some devices have only a single point of control, which allows for simplicity of operation, and reduces the need for a bi-directional control interface. For example, a projection screen whose motor controller is connected only to a control system processor does not require a bi-directional interface, because control system processor commands are the only way the screen could change position. This allows the control system to accurately reflect the screen's position without a direct status update from the motor controller.

The majority of AV equipment, however, can be controlled from at least *two* points of control, and often three or even four[2]. The same projection screen, with a wall switch installed in parallel with the control system, becomes a device with two points of control. In order for the control processor to now accurately reflect the screen's position, accommodations must be made for the control system to detect a closure at the wall switch. Devices more complicated than a motor controller, with two or more points of control, often need some form of bi-directional control interface so that their status can be tracked by the control system.

In many cases the equipment manufacturer has neglected to add true device feedback, or even account for third-party control, and the system integrator is required to utilize devices such as power current sensors or video sync sensors, just to understand the state of a device. This is often the case with devices that have one "toggle" command for power, rather than two discrete commands for power on and power off. Sending the power toggle command as part of a system start-up sequence would unintentionally power down a VCR which may have just powered up when a cassette was inserted.

Bi-directional communication may actually be *required* for safety and maintenance purposes. Projectors installed on a lift and recessed into an enclosed area must first be powered off and allowed to cool in

[2] Consider a device with external buttons on its case, an RS-232 control connection, a manufacturer's IR remote control, and a built-in web server. This situation is becoming more common for complex devices such as switchers, digital signal processors (DSP's) and digital displays.

order to avoid equipment damage and prevent a fire hazard. The portability of today's projectors has also made them a target for thieves. Maintaining a constant 2-way dialog with a projector enables security personnel to be notified in the event that the projector has been removed. Collateral benefits can result as well. For example, today's projectors record lamp hours, so many facility managers expect to be notified by the control system when the lamp's life expectancy is near its end, so that they can replace the lamp before it fails.

Another interesting form of bi-directional control is currently being adopted, in which device-internal controls are called by the external control system. This method is popular with complex sub-systems (e.g., lighting, security, and HVAC). Digital signal processors (DSP's) now typically enable internal macros to be used for volume control, with the external control processor triggering the macro, and then polling the DSP to obtain the true state of the adjusted volume level. In some cases, the DSP would have automatically updated the control processor with the volume level change through unsolicited feedback. This method of control allows an audio specialist to tune multi-channel volume control step size and range to match the acoustic properties of the space, without requiring the control system to be reprogrammed.

Finally, the most information-rich interfaces are required for content-delivery devices, such as media servers, which may provide a wide variety of content and layers of metadata about the content, in addition to information about the device itself. The control system now needs to provide a convenient way to browse and search collections of content based on metadata. It is no wonder that the typical communications medium of choice in these systems is an Ethernet connection.

1.04 Comments and Suggestions

The Independent Programmers Council has developed the Roadmap to encourage all that read it to reach for a higher standard of control interface development and documentation. To that end, we welcome comments and suggestions that could improve this document. Send them to membership@InfoComm.org with a subject heading of "Roadmap to Control."

2. Recommendations for Interface

2.01 Simple ASCII Commands and Replies

The AV industry is international. Often, the individual developing a control interface speaks a different language from the people designing systems and implementing controls that use them. The language barriers often result in confusion, frustration, and loss of valuable time.

One solution has been to write the control interface in a language that neither party speaks: hexadecimal[3]. This requires the engineer developing the protocol to translate all commands into hexadecimal, and write a document in his or her native language describing the hexadecimal commands. This document is then translated into the native language of the programmer who will be implementing controls that use the interface. The end result is rarely desirable, as many layers of translation separate the two parties.

The Roadmap recommends that control interfaces be moved into the plain-text domain, eliminating at least one layer of translation. Given its large-scale acceptance, American Standard Code for Information Interchange (ASCII)[4] makes the most sense as the base language for these communications. ASCII has the following advantages.

[3] Hexadecimal – often referred to as "hex" – is a base-16 numbering system that uses a combination of digits and letters. It is primarily used as convenient and compact shorthand for binary numbers.
[4] ASCII is also permanently embedded in Unicode as the lowest 128 characters. This range of characters in Unicode UTF-8 may be used interchangeably. For readability, this document will refer to these characters as ASCII. For more information on ASCII, please refer to ANSI X3.4-1986 or ISO/IEC 646:1991.

- ASCII is a seven-bit code, allowing any character to be transmitted in a single byte
- Various international standards bodies have ratified ASCII (or issued a nearly identical standard), including:
  - European Computer Manufacturers Association (ECMA)
  - American National Standards Institute (ANSI)
  - International Organization for Standardization (ISO)
  - International Telecommunication Union (ITU)
  - German Institute for Standardization (DIN)
  - Internet Engineering Task Force (IETF)
- AV control systems manufacturers heavily base their programming languages on both English and ASCII.

With an ASCII-based approach, all control communication should also be transmittable and receivable by simple ASCII communication programs, such as HyperTerminal or Telnet, without the use of complex function key combinations. Both applications are available on a personal computer running Microsoft Windows, and support the transmission and receipt of ASCII characters from 0 to 9, A to Z, simple punctuation, and the basic control characters, carriage return and line feed.

The following examples illustrate the difference between Hex and ASCII control commands.

Example 1

```
0x02 0x50 0x4F 0x4E 0x03
```

Example 2

```
PWR ON<CR>
```

In example 1, the equipment designer saved a little time by not translating the “power on” command from a string of hex characters. However, this time is lost every time a programmer has to reference a protocol manual just to turn on the device. In example 2, the equipment designer invested the time to write the string in plain text up front, saving time during each implementation of the product.

Early instances of serial interfaces were often direct translations of hex IR codes to RS-232. At the time the IR codes were developed, they were very advanced in terms of condensed memory space and easy-to-interpret parsing by the 8-bit-or-less processors embedded into the devices at that time. For example, consider the compact and well-defined device and command representations of the Sony SIRCS codes and the Philips RC-5 codes. Converting these pulse code modulation (PCM) IR codes into ASCII pseudo-hex representations of hex strings was easy for the manufacturer, but a continual effort and drain of resources on the part of the AV control system programmer to decipher.

Imagine an installation technician or control system programmer debugging the control system. To verify that the control system and device are exchanging the correct data, a simple communications program such as Hyperterminal could be used to check either side of the interface using quick ASCII commands. To use hex strings would require complicated multiple keystrokes or writing macros that can send multiple character hex strings from a single keystroke. This is not within the skill set that one would typically find or want to train in a skilled AV technician, who is merely attempting to verify a cable connection. Simple ASCII protocols would not require advanced serial or Ethernet troubleshooting programs, which monitor both sides of a communications link, to verify or program a basic system.

If a manufacturer wants to implement a complex control interface, which consists of hexadecimal strings, check-sums and/or multiple transmit/receive strings for each control function, then it should provide a simple personal computer-compatible program to control the device. Such a program would enable proof that the wiring to the device is correct, and it would also allow the programmer to see a working example of the control interface to emulate in the control system code.

2.02 Plaintext Commands and Replies

Retrieving a parameter should be as simple as possible. For instance,

```
T?<CR>
```

should cause the thermostat to send back the current temperature as

```
T=072 F<CR>
```

Single letter commands (T versus TEMP, S versus SET, etc.) preserve code space over full word commands while still allowing easy controller programming and simple on-site troubleshooting. Better yet, the thermostat should send the current temperature automatically whenever it changes, without having to be asked (known as unsolicited feedback, see 2.08).

Padding the value with “leading zeros” for the longest expected data set allows for simple parsing of the reply, as the reply will always be the same length. For example, most of today’s video projectors will never see a lamp last for over 9999 hours; therefore, four digits is the largest value that should be required to reply to a lamp life query. To allow for the simplest parsing of the reply by the control system, responses to the lamp life query should always contain all four digits. This is accomplished by padding the value with leading zeros. For example, if the lamp has been in use for 32 hours, the reply might be:

```
LAMP=0032<CR>
```

2.03 Simple Help Menu

Using H<CR> or ?<CR> should provide a list of commands recognized by the device. Providing parameter ranges and/or number of bytes has a manual-on-a-chip effect that provides very tangible benefits both to consumers of a device and the manufacturer’s technical support personnel. Chip upgrades effectively contain their own manual updates. The code-space used by the help code is seldom needed by the original design engineers, but becomes more important than the latest device features if it makes the device implementation more successful in the field.

2.04 Firmware Version and Contact Information

Every device should have some means of reporting the current version of firmware loaded on it, along with the effective date of the firmware and contact information to verify if this is the latest version. Although it might seem beneficial to release minor changes to firmware as bugs are discovered and fixed, it is sometimes better to consolidate large revisions as major firmware releases. Doing so would allow enough time and performance of case studies to effectively test the revisions. Obviously each case is unique, and should be addressed according to the complexity of the device, the control interface, and the scope of the changes. Firmware should not be modified in a way that disables commands from a previous version, or changes the formatting of replies to queries.

While it may be convenient to create “one-off” versions of firmware for special project requirements, care must be taken to avoid releasing these into general circulation. While such versions may help expedite a project, releasing the “one-off” version may lead to confusion, which would require extra technical support time and unique training.

2.05 Support for Multiple Session Types

Control interfaces have different requirements depending on their intended use. The control interface for a device may provide control of almost every variable available within the device. This is often beyond what is needed for effective control and feedback in an AV control system.

Some manufacturers have developed the concept of offering “full sessions” and “limited sessions.” The limited session can be pre-defined or tailored according to the control system requirements. Some systems allow both types of sessions to run simultaneously (i.e., full session via Telnet and limited session via RS-232). This gives tremendous power in terms of debugging the interface between a controller and a controlled device.

However, supporting too many configurations can make it difficult to provide a simple programming interface. Ideally, setting the configuration parameters should be as simple as the basic serial interface. For instance, setting the current temperature options could be

```
!T,F,T30S<CR>
```

for a thirty second timer update in Fahrenheit. Checking the outdoor sensor settings could be:

```
?OT<CR>
```

The system response, for polled Fahrenheit, would be:

```
!OT,F,P<CR>
```

Similar control strings could be developed for schedules, heat/cool separation, fan purge, etc.

Some systems require custom configuration software to set them up. Some limit the available configurations to avoid providing programming hooks into the system. Neither of these two situations provides the user with the greatest value. Simple ASCII, plaintext commands to reconfigure a system can help make a complex device simple to use.

2.06 Fully Detailed Replies That Assume Nothing

When replying to a query, the controlled device should assume that the control system does not know to which query the reply is responding. It may seem that replying without a header or ID is sufficient, but if the device responds with only the data, the control system has to keep track of every query it makes to that device, a process that is difficult and unnecessarily complicated, and that does not scale to high volume usage or multiple control points.

For example:

```
02_VOL_I_1?<CR>
```

should return the following full string of data, which includes all parameters from the query:

```
02_VOL_I_1=066<CR>
```

Rather than the following abbreviated string, which may not be properly recognized by the control system:

```
66<CR>
```

The practice of returning a full string of data, including all parameters, has the additional benefit of enabling unsolicited feedback, which would allow a system to be programmed so that the control system can be aware of device changes without having to ask continually if any changes have occurred.

2.07 Verbose Error Messages

When a command cannot be executed, or the piece of equipment is in a fault state, a detailed error message should be provided. This should include as much information as possible, rather than a generic and non-specific error message such as “ERR”.

2.08 Status Update Modes

There are four prevalent status update modes:

1. **Unsolicited Verbose:** This reports every change (e.g., button press, temp change, mode) with the full system status. Verbose status updates are good for designing and troubleshooting a system, but the amount of information transmitted can overload the control system.
2. **Unsolicited Subscription:** This reports status only for specific selected functions for which you “subscribe” to receive unsolicited feedback, and only when the state of such a function changes. The advantage is that the control system receives only the information that it needs, only when there is a change in state, thereby preventing information overload.
3. **Polled:** This reports status only when polled from an external source, such as the control system, and allows for total system polling and individual function polling. Polling provides a precise way of keeping track of device status, but requires more activity and intelligence on the controller's side.
4. **Timed:** This reports status at specific pre-determined intervals, of both total system and individual status points. Timer updates can be used to communicate system values directly to the appropriate points in the control system with little or no intervention.

Sometimes a combination of polled and timed status works well, so that specific status points such as an HVAC system’s mode and active schedule can be polled, but real-time information such as temperature and system status can be updated upon specific intervals.

The AV industry trend, and generally preferred way to accomplish status updates, however, is through unsolicited subscription feedback. This presents the advantage of reporting any change to the state of a system, so the control system is kept current, without overwhelming the control system with unwanted data.

Unsolicited subscription feedback can provide benefits similar to the full session/limited session implementation, but at a much simpler level. Some events in a controlled device are required to provide effective feedback to the users of a control system. A simple example would be an audioconferencing system that annunciates an incoming phone call by giving users the choice between “Answer the Call” and “Do Not Disturb.” It is possible to poll the audioconferencing system for incoming calls, but it is much more practical and efficient to be able to configure the unit to send an unsolicited notification to the control system when an incoming call is detected.

Consider a device equipped with front panel controls, which have the same functionality -- such as volume control -- as the control system’s user interface. The front panel volume control may be used infrequently, but unless it is constantly polled by the control system, the user interface may reflect an inaccurate volume level. With unsolicited subscription feedback, however, the device can send the volume change when it occurs, keeping the two synchronized. Unsolicited subscription feedback should be supported by the control interface of most products.

2.09 Timing

Timing becomes critical for systems that control perceptible elements directly: there should be minimal time lag between the transmission and execution of commands. Imagine a volume control that continues to ramp up after the “volume up” button has been released. In a moment, it can surpass the desired level, and reached an uncomfortable, or in some cases, unsafe, level. Such behavior can be confusing and frustrating to users.

Another subtle, and often overlooked aspect of timing, is in the parsing of feedback from a controlled device. When large strings of feedback can instantaneously be sent by a controlled device, using flow-control such as hardware hand-shaking can improve the parsing of long, concatenated serial strings. Flow control simplifies the timing requirements of both the control system and the controlled device, as the hand-shaking can pace the feedback responses to a single string at a time. Software-based flow control can accomplish a similar result.

2.10 Physical Details for Serial Control

In the EIA-recommended RS-232 standard, two types of connections are mentioned that date back at least half a century. The first, Data Terminal Equipment (DTE), was originally implemented for teletype machines or dumb ASCII terminals that once were the user interfaces to computing devices. The second, Data Communication Equipment (DCE), was typically for modems[5] and other equipment used to communicate between the user interface DTE equipment to the mainframe computer.

Signal	Name	DE-9P	DB-25P	Direction
DCD	Data Carrier Detect	1	8	To DTE
RD	Received Data	2	3	To DTE
TD	Transmitted Data	3	2	From DTE
DTR	Data Terminal Ready	4	20	From DTE
GND	Signal Ground	5	7	Bi-directional
DSR	Data Set Ready	6	6	To DTE
RTS	Request To Send	7	4	From DTE
CTS	Clear To Send	8	5	To DTE
RI	Ring Indicator	9	22	To DTE

Figure 1: DTE/DCE Cable Pin Assignment Table

The major control system manufacturers use the standard DB-9 male DTE connector, popular on personal computers since the mid-1980s. The easiest and most trouble-free approach to developing a serial interface is to adopt the complementary DB-9 female DCE configuration. The cabling pin-out is a straight-through one-to-one relationship. Controlled devices that adopt the DB-9 male DTE configuration, on the other hand, require a null-modem cable or adapter in order to make each device appear as a DCE device to the other. A null modem normally reverses the TD and RD pins (2 & 3). Failing to reverse these pins when required is one of the most common wiring mistakes in the field.

Another common wiring error is the failure to reverse the hand-shaking flow control RTS and CTS pins (7 & 8), which is also accomplished by a null modem, or the failure to terminate pins 7 & 8 at all. The simplest bi-directional serial interface (i.e., no hardware handshaking) can be accomplished with three wires: TD, RD and GND (pin 5). By default, many installers only terminate for a three-wire connection in the field. If hardware hand-shaking is required in these cases, two-way communication will not be established. To help avoid such issues, manufacturers should carefully consider the requirement for hardware handshaking, and only use it where absolutely required.

If a large volume of communication traffic is expected, Ethernet-based control may be preferable to serial control. If serial control must be implemented in such an environment, RTS/CTS hand-shaking should be utilized. DSR/DTR signals are typically relegated to modem communication and are not generally used for hand-shaking on AV Devices.

For an RS-232 connection, only pins 2,3,5,7 and 8 should be utilized by a controlled device within the AV marketplace. The other pins are often reserved by control systems manufacturers for other control formats.

Note that if a non-standard pin-out is used for a serial connection, the pin-out should be silk-screened or otherwise obviously labeled directly on the device near the serial port.

[5] Modem is actually an acronym for MODulate and DEModulate, and refers to the conversion of digital data into a signal that can be transmitted over the public switched telephone network (PSTN), which can only handle analog signals which are within the frequency range of voice communication.

Multiple baud rates and protocols are seldom required by control systems. For most devices, a baud rate of 9600 8N1[6] is appropriate for RS-232 control. In some instances 38400 8N1 may be appropriate for devices that send and receive large amounts of data. Speeds in excess of 38400 are not recommended, as they can be problematic with longer unbalanced cable runs. Non-standard parity, stop bits, or data bits are not recommended, as their usage is so rare that many programmers and technicians may not immediately think to try a non-default setting.

2.11 Physical Details for Ethernet Control

One of the major advantages of Ethernet control is the simplification of the physical connection. Volumes of specifications have been written for Ethernet, eliminating the need here for much discussion about the physical connection. A standard RJ-45 Ethernet connection should be used on the device, rather than relying on adaptors such as serial-to-Ethernet, USB-to-Ethernet, or Wi-Fi[7] bridge of some sort.

Devices with Ethernet control should either default to a specific well-documented IP address when shipped from the factory, or be configured to accept a DHCP address. It should have an ARP-assigned address[8], with documentation.

2.12 Standardized Checksums

Standardized checksums have been developed over the years as well as cyclic redundancy checks (CRC's) for ensuring that data which has been corrupted by communication errors can be corrected or discarded. In a typical AV control system there is limited need for checksums or CRCs, which are used in places where the communication channel can be corrupted by electrical interference or communication errors and the data is mission-critical. In a typical control system, the communication channels are hard-wired RS-232 connections with short lengths of cable, usually indoors, and often in conduit, thereby limiting exposure to potential interference. The nature of an AV control system, moreover, is often to *display* mission-critical data, but seldom to actually *transmit* it. Because they are burdensome to implement, therefore, checksums should be implemented on AV devices only when absolutely necessary.

In the rare cases that checksums are truly required, the serial interface will need to be able to generate and parse hexadecimal values. The simplest checksums are byte summing or XOR-ing of the data characters in the string. The checksum is typically the last byte of data truncated to hex 0xFF or less. Typically checksums are used in protocols that specify a start character, such as STX, and a termination character, like ETX, along with a data character count byte following the data string. As mentioned above, this would be overkill in most AV devices and precludes the simple ASCII commands and replies recommended in this document.

Checksums at the control interface level are unnecessary in an Ethernet control environment. The Ethernet protocol itself performs CRC's at the data link layer, and TCP/IP or UDP/IP handle any bad packets.

[6] Serial communication shorthand for: eight data bits, no parity, one stop bit.
[7] Wi-Fi is a trademarked expression of the Wi-Fi Alliance, formerly known as the Wireless Ethernet Compatibility Alliance (WECA), and refers to its Standards for Wireless Fidelity brand. The standard applies to wireless local area networks (WLANs) based on the IEEE 802.11 specifications.
[8] Defined in Ethernet Request for Comments (RFC) 826, Address Resolution Protocol (ARP) is a method for resolving a device's physical Media Access Control (MAC) hardware address from its IP address.

3. Recommendations for Documentation

3.01 Accessibility of Documentation

The control interface documentation should be easily accessible to all users of a given device, including installation personnel, engineers, programmers, service technicians, end users, technology managers, and sales staff. This helps ensure that the correct product is specified for a particular application, on the basis of the availability of all required commands or queries. This practice will also lead to a device being used to its highest potential in more projects, which will lead to satisfied customers and repeat purchases.

The preferred method of distributing control interface documentation is within the device's user manual itself. If that is not practicable, it should be published as a separate document. Either way, it should also be easily accessible for public download on the manufacturer's website as a Portable Document Format (PDF)[9] file. It should also be provided in print or on CD-ROM in the box with the device, since it is likely that the device will be installed in situations where Internet access may not available.

3.02 Summary Descriptions

The opening section of any control interface technical documentation should be a summary description of the capabilities of the control interface and the available commands/queries. It should use language that can be understood by non-technical readers, which will allow system designers and sales personnel to determine the applicability of the product to a particular situation, without engaging an engineer or programmer.

3.03 Contact Information

The documentation should include technical support contact information. This should include at least a telephone number, a web address (to allow the user to check for an updated copy of the document before calling for support), and an email address. If possible, these numbers and addresses should be for a support group, not an individual.

3.04 Numeric Scales & Presentation

All numeric values should be carefully defined in scale as well as representation. A valid range should be provided for each command or query that contains numeric data. Each numeric value should be explicitly defined as to representation. Most values should be decimal values expressed as ASCII (for example, 106 = one hundred and six). Should a legacy device require different representation, such as hexadecimal values expressed as hexadecimal, in which 6A (single byte 0x6A) = one hundred and six, or hexadecimal values expressed as ASCII in which 6A (two bytes 0x360x41) = one hundred and six, non-ASCII numeric values should be clearly defined throughout the document. For example, literal hexadecimal bytes should be identified with a leading 0x (i.e. 0x0D for a carriage return.)

[9] PDF – Portable Document Format – is a cross-platform document standard created by Adobe Systems. PDF files are converted from their original format and viewed via Adobe Acrobat products: www.adobe.com. The use of PDF preserves the original font, images, graphics, and layout of the source file, and can be viewed on or printed from any computer, regardless of operating system, font availability, or particular applications installed.

3.05 Visual Distinction of Codes Samples

Visually distinguish between the text of the interface document itself and sample code or strings that would be sent to or received from the device. Sample strings should be represented using a different font (Courier has been widely used for this in the computer world) and clearly indented from the body of the document itself. For example, the following could be used to clearly explain the operation of an image blank command for a video display:

> IMAGE BLANK COMMAND – BLANK
> Sending this command will blank or un-blank the image. Examples:
>
> > BLANK ON<CR>
> > BLANK OFF<CR>

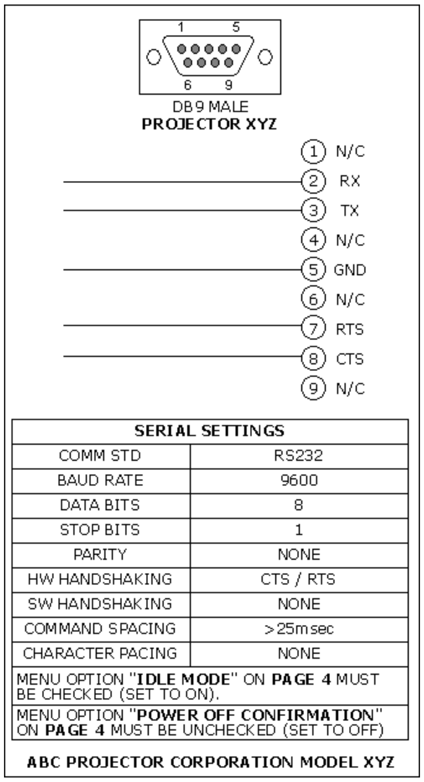
Be sure to clearly define line terminators. In most cases, a single carriage return should be all that is necessary. This can be represented in documentation as either <CR> or 0x0D. If a device requires a carriage return and a line feed, be sure to make this clear, such as <CR><LF> or 0x0D0x0A.

3.06 Inclusion of External Standards

Technical documentation should be fully detailed in every aspect of feature, functionality, command and query. Even if a control protocol is based on another standard that is detailed elsewhere, such as Telnet or XML, full details should still be provided, to ensure implementation of the product with minimal involvement of the technical support department.

3.07 Connection Details for Serial Control

Connection details should include all necessary information to establish a connection with the device. Nothing should be left to assumption. This section will differ significantly on the basis of the connection format. If the device control interface supports multiple physical connection methods (such as RS-232 and TCP/IP via Telnet), with the same interface protocol language, the documentation should include information for both. The documentation should also include any special preparations that must be made on the device in order to allow external control, such as specific menu settings.



SERIAL SETTINGS	
COMM STD	RS232
BAUD RATE	9600
DATA BITS	8
STOP BITS	1
PARITY	NONE
HW HANDSHAKING	CTS / RTS
SW HANDSHAKING	NONE
COMMAND SPACING	>25msec
CHARACTER PACING	NONE
MENU OPTION "**IDLE MODE**" ON **PAGE 4** MUST BE CHECKED (SET TO ON).	
MENU OPTION "**POWER OFF CONFIRMATION**" ON **PAGE 4** MUST BE UNCHECKED (SET TO OFF)	

ABC PROJECTOR CORPORATION MODEL XYZ

Figure 2: Recommended Format for Serial Connection Information

Serial connections should include specifications regarding the type and gender of connector used and the function of each pin used on the connector. It should also include the data transmission speed (baud rate), requirements for command spacing (time required between commands), character spacing (any pacing requirement between characters), character length (data bit), parity bit, and stop bit. Hardware or software flow control settings should be included, even if the requirement is NONE. See Figure 2 as an example for the format in which serial connection information can be presented.

3.08 Connection Details for Ethernet Control

Devices controlled via Ethernet require different connection information. There is normally no need to provide physical pin-out information for the RJ-45 connection, since it is highly standardized. As mentioned elsewhere, however, the default IP address, specification of TCP/IP or UDP/IP communications, and the port number on which the connection will take place, should be provided, as well as the default user name and password, if required. Instructions should also be provided about how to reset the IP address to the default if the address has been changed and forgotten. Any login, authentication or hand-shaking sequence required to open a session should also be outlined.

3.09 Command Formatting

The document should explain, in detail, the formatting of a valid command. This should outline each byte of the command and its function. A good method for accomplishing this is to take a commonly used string, such as power on or off, and thoroughly explain each character. Using such a common command allows the reader to use it as a test string to check the physical connection. It is a good idea to include the entire string in both ASCII and hexadecimal notation for maximum clarity as to data representation. The documentation should outline any preamble or header, addressing scheme, command, data, check sum, and delimiter or end byte. Figure 3 is an example of a good graphical presentation to explain the formatting of a command:

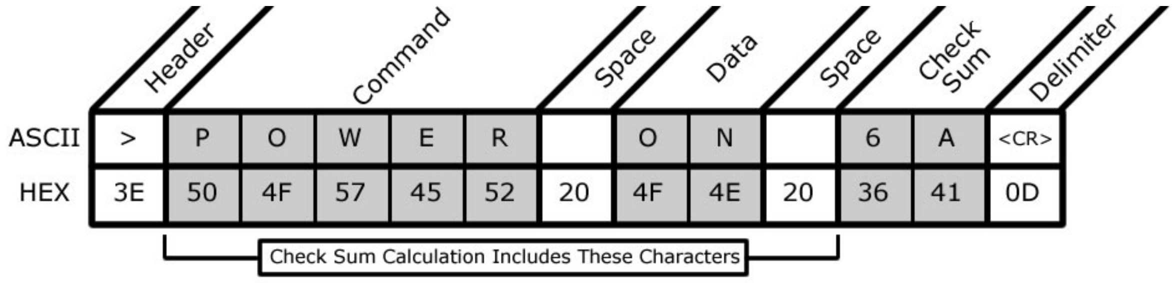


Figure 3: Recommended command formatting

Such a graphical depiction should also have a textual explanation of its details, such as this illustration:

> The above example command will power the device on. The following is an explanation of each byte in the command string:
>
> **Header** – Each command or query sent to this device must begin with the header character > (or 0x3E).
>
> **Command** – Locate the desired command or query in the following command list. The command for power is POWER.
>
> **Space** – A space (0x20) character is required between the command and the data.
>
> **Data** – Most commands require some data to be executed. In the case of the POWER command, the data will either be ON or OFF, depending on the desired function. In this example, we are turning the device on, so our data is ON.
>
> **Space** – A space (0x20) character is required between the data and the check sum.
>
> **Check Sum** – Each command or query must contain a valid check sum in order to be executed. The check sum is calculated by adding the hexadecimal values of the Command, both Spaces, and the Data. In this example, the addition is:
>
> 50 + 4F + 57 + 45 + 52 + 20 + 4F + 4E + 20 = 26A
>
> The last two digits of the result (in this case, 6A) are used in the checksum, represented in ASCII.
>
> **Delimiter** – Each command must end with a carriage return (0x0D)

The document should also include equally detailed information about responses received back from the device, such as acknowledgments or error messages.

3.10 Commands

A list of all available commands should be provided, including the applicable data for each command, a usage example, and expected replies. This does not need to be as detailed as the formatting section, since the formatting section will serve as the user's point of reference. If multiple formats are accepted, provide an example of each. If there are particular conditions that must be true in order for the command to be accepted, they should be listed. If the command will take time to execute, during which other commands should not be issued, this should be noted. For example:

Volume Control
The command VOLUME will set the main volume of the receiver. Valid data ranges from 0 to 100. Leading zeros are acceptable but not required. All values are decimal represented in ASCII. This command will be recognized only while the receiver is powered on. The receiver will reply with the new volume level.
Example:

```
Controller Sends:
        VOLUME 45<CR>
Receiver Responds:
        VOLUME=045<CR>

Controller Sends:
        VOLUME 008<CR>
Receiver Responds:
        VOLUME=008<CR>
```

3.11 Queries

All available queries should be outlined, carefully detailing the responses that can be expected from the device for each query. For example:

Lamp Hour Query
The query LAMP?<CR> will return the lamp hours for the projector. Example:

```
Controller Sends:
        LAMP?<CR>
Projector Responds:
        LAMP=0178<CR>
```

3.12 Errors

Error replies generated by the device when replying to commands or queries should be outlined in detail, defining each potential error code or condition. For example:

The following error codes will be returned by the mixer if a command cannot be executed:

```
ERROR01<CR> - Unknown Command
ERROR02<CR> - Input out of range
ERROR03<CR> - Output out of range
ERROR04<CR> - Invalid parameter
ERROR05<CR> - Bad checksum
ERROR06<CR> - Feature not supported by this model
```

3.13 Examples

Each command or query should be listed with a full example of both transmitted and received data, so a dedicated “example” section at the end of the document should not be required. Manufacturers may find it advantageous to have sample programs for popular control systems available for download from their websites. This provides a point of reference for a user to test communications with a device, as well as a source for “copying and pasting” desired commands into a program. If such example programs exist, the control interface documentation should state that and provide instructions for obtaining them. Major control system manufacturers are often willing to create sample programs or modules through their manufacturer liaison programs, and independent certified programming organizations are also available to provide this service.

4. Conclusion

In conclusion, manufacturers have the opportunity to create superior control interfaces for their products by establishing this as a priority. Products that are easy to control effectively will enjoy success in the marketplace, and provide benefits to all stakeholders. Consultants will get to know that a product can be easily integrated, and will specify it repeatedly in the future. Integrators and programmers will be able to complete systems more quickly. The manufacturer will sell additional product, and spend less time and money on technical support. And, most importantly, the consumer will enjoy a more functional and more reliable system. This will increase confidence in and comfort with AV systems in general, promoting interest in additional projects in the future.

The recommendations of the *Roadmap to Control* exist as a guideline for manufacturers in the development of simple and reliable control interfaces for their products. InfoComm International encourages comments and suggestions for potential improvements to this document, or questions about its application. Send them to membership@infocomm.org with a subject heading of "Roadmap to Control."