Serial Programming/Modems and AT Commands

1 Introduction

1.1 General

This content is part of the Serial Programming book. It covers the programming of Hayes and Hayes-compatible telephone modems. Such types of modems are the norm in consumer applications, as well as many professional applications - basically, wherever modems are still used.

Modem programming is slowly becoming a lost art, particularly with the widespread migration of users from dial-up lines to DSL for very obvious performance reasons. Nevertheless, modems are used for many applications. In recent times, modems can be found in new areas where they were previously not seen. For example, embedded modems in machines are used to automatically "call home" to the manufacturer in case the machine is in need of some service. Often this is done via a wireless phone system, where the wireless module still provides a Hayes-compatible interface for dialing and data transmission.

The original Hayes modem command set is exclusively used as a reference in this module. **Vendor specific extensions are not covered, and do not belong into this module**. The module explains the origin of the term *Hayes*, and the related *AT commands*. It also includes some principal information about what a *modem* is, and how the signaling with a modem happens. The module then continues with a description of the basics of modem programming, including the set-up of a development environment.

Further, the module provides detailed (but *incomplete*) programming information, and an *incomplete* reference of the original Hayes command set and registers.

1.2 Administrative Information

This section particularly addresses potential authors. Please note:

- This module **is not** a dumping ground for random modem programming information and folklore.
- This module is **operating system agnostic**. The Programming Serial Data Communications book provides other modules for such information.
- This module deals with **generic Hayes modems**, not with any vendor specific extensions. If you really want to see your particular love-child covered, provide an Appendix with that vendor/brand specific information.
- Do not assume that just because something works on you particular modem it is the standard and other modems do it the same way. If you have no first hand experience that something is done the same way on "almost" all Hayes-compatible modems, then leave it out, or mark it at least as doubtful.

The reason why this module sticks with the original Hayes command set is to have a defined boundary. This module is not intended as a reference manual. Once someone has mastered the basic set, and implemented the code, it is rather straight forward to deal with vendor-specific extensions. Other extensions, e.g. the very rough and basic FAX extensions require some deep insight into the involved protocols (e.g. in the case of FAX the detailed encoding, compression and timing of fax data on the phone line). This is out of the scope of this book. If you know how to handle the FAX extensions, write your own book.

1.3 What is Hayes?

Hayes Microcomputer Products, Inc. was a modem manufacturer from the beginning of the 1980s until the end of the 1990s, with its heyday in the early '90s. The name *Hayes* still exists as a brand name, owned by *Zoom Telephonics, Inc.* (as of Fall 2004).

In 1981, Hayes developed the **Hayes Smartmodem**. This was a unique product at the time, because this modem was no longer simply a "dumb" device blindly converting serial data to and from audio tones, but contained some “intelligence”. It was possible to send commands to the modem to configure it, to execute certain operations (such as dialling a number, quieting the speaker, hanging up, etc.), and to read the current status of the connection. Hayes developed and published a command set to control the modem over a serial line. This command set became popular among consumer modem manufacturers, and was cloned a thousand times. Known as both the “Hayes command set” and the “AT command set”, it has long been the de-facto standard for controlling consumer modems and also many professional modems. Modems which support this command set are called *Hayes-compatible*.

The commands were standardised at some point in time, however, as it is typical with standards, there are several standards. Plus, of course, there are still vendor-specific extensions and implementations in different modems vary slightly. Some of these enhancements were required to support at that time emerging features, such as data compression and FAX support. As a result, the command sets of modern modems are not fully compatible with each other. The original Hayes commands, however, should still work, and still form the core of almost all consumer modem command sets.

The basic set of commands was at some point in time standardised as TIA/EIA-602 and the syntax as EIA/TIA-615. But as already mentioned, modem manufacturers added their extensions. A larger extended set, particular under the pressure from cell phone manufacturers, was standardised as ITU V.250 (old name V.25ter). That one usually forms the base for professional Hayes-compatible modems, and cell phones with build in data modems. ITU V.250 further referes to a bunch of other standards (e.g. V.251, V.252, V.253) for particular applications and extensions, and also has some supplements. Plus, of course there are the many standards defining other aspects of a modem, like compression and transmission.

See Also:

- Wikipedia:Hayes Communications
- Transferring Data between Standard Dial-Up Modems

1.4 What are AT Commands?

Almost all of the Hayes modem commands start with the two letter sequence AT - for getting the modem’s *attention*. Because of this, modem commands are often called *AT Commands*. This still holds for many of the manufacturer specific command set extensions. Most of them also start with AT, and are called *AT Commands*, too. Please note, that just because an AT command contains a *&* does not make it an extensions. *&* commands were already part of the original Hayes command set.

The exact usage of the term *AT command set* slightly varies from manufacturer to manufacturer, often subject to marketing blurbs. In general, it can be assumed that a modem with an *AT command set*

- uses commands mostly starting with AT,
- uses the original Hayes way of separating data and commands, and
- supports the original Hayes commands and register settings as a subset.

1.5 What is a Modem?

A modem in the classic sense is a **mo**dulator/**dem**odulator for transmitting digital information over analog wires, such as the analog telephone system’s two-wire or four-wire lines. The term has come to be used as acceptable slang for many communication devices used to link a computer to either another computer, or a wide-area network (Wikipedia:WAN). For example, the Ricochet radio data transceivers were commonly known as “Ricochet modems”.

This module deals with the classic type of *smart* modems, designed to convert data from/to a serial interface to/from an analog line. The module also applies to modems which provide the classic serial interface but connect over a different physical layer, such as a digital line, as well as devices providing a serial modem-like interface for other purposes. For our purpose, the modem is a classic DCE (data communications equipment) device, controlled via serial line by a classic DTE (data terminal equipment) device (such as a computer).

Depending on the type of modem, the modem can use a number of different technologies and speeds to transmit the data over the analog line. The details of these technologies are of no particular interest here, other than to note that it is possible with most modems to specify these communication parameters (for example, to disable compression, or to change modulation techniques). The data this module deals with is not the data on the analog line, but the data as it appears on the serial interface between the DTE and DCE. I.e. the data as read and written by a device like a computer.

(*Smart*) Modems also provide auxiliary services, such as dialling a particular number to set up a connection. As a consequence, a modem can be in a number of different states and modes, which are not always orthogonal. It is possible, for example, for a modem to be in the command mode while still keeping a connection (see the +++ sequence for details).

Non-smart modems had to rely on other equipment like an ACU (automatic call unit) to provide these auxiliary services, but they are practically extinct today.

1.6 Inband Signalling

The original RS232C/V.24 specification contained a TX wire for transmitting data and a RX wire for receiving data, and other completely separate wires for transmitting control information between the DTE and DCE, the idea being to separate data and control information. In telecommunication jargon this is called **outband signalling**.

Hayes-compatible modems use almost none of these RS232C/V.24 features. Instead, communication with the modem is done almost exclusively via the same RX/TX lines which are used for transferring the data. This mechanism is called **inband signalling**.

Inband signalling has significant disadvantages. At any point in time, both the DTE and DCE must know if information sent or received via the TX and RX lines is for signalling purposes, or if it is data, which should be handled transparently. Therefore, the DTE and DCE must operate in sync. If they get out of sync, either data will be lost, data will be incorrectly interpreted as commands, or signalling information will be interpreted as data, effectively destroying the original data.

Inband signalling has the advantage that the wiring between the DTE and DCE is simpler, and also that, at least at first glance, the communication software in the DTE is simpler.

As it has been said, Hayes-compatible modems use almost none of the RS232 control lines. But only almost. For example, they often drive DCD (data carrier detect). This, however creates the situation that modem-driving software now has to take care not only of the inband, but also the outband signalling with a modem. This slightly complicates the communication software's state machine.

Further, especially with the rise of cell phone modems, manufacturers have again started to introduce more outband signaling. Such modems provide multiple virtual serial interfaces. Some of these interfaces are exclusively dedicated to data transport, controlled by another serial interface which is either used exclusively for signalling (i.e. outband signalling) or can still also be used in the more conventional inband signalling scenario. In such cases the communication software needs to manage even more complex states.

1.7 Command State / On-line State

With respect of controlling the modem a Hayes-compatible modem is one of two main states:

Command State The modem interprets data from the DTE as modem commands. The modem can be in command state while still keeping a connection with a remote party.

On-line State The modem interprets data from the DTE as payload and transmits it to the other party. This state requires that a connection to the remote site has been established.

Inside these main states are a number of sub states. Also, with respect to other issues a modem has a number of communication states, e.g. if a remote carrier has been detected or not.

1.8 Originating Mode / Answer Mode

Originating mode A modem in originating mode is a modem which is setting up a connection, e.g., by dialing the number of a remote station and initiating the negotiation of protocols.

Answer Mode A modem in answer mode is a modem waiting to be contacted and ready to "answer the phone".

1.9 Command Responses

A modem is supposed to send a response for almost all commands it receives. These responses can either be in the form of ASCII strings, or numeric values. The response type can be switched with a command, but it is typical to use the ASCII responses.

Responses need to be tracked by the DTE with great care. Among other things they inform the DTE if the dialling of the remote site was successful or not, and if the modem switches from command state to on-line state or not.

Unfortunately, the set of response messages has been greatly enhanced since the original Hayes modems and are often configurable via additional AT commands. It is suggested to not strictly parse response messages but to forgivingly check if they contain interesting keywords, like CONNECT. It is also suggested to study the manual of a particular modem very carefully.

1.10 S-Registers

The so called S-registers are also a Hayes heritage which all Hayes-compatible modems support. They are registers in the modem which contain various settings. And like the AT commands, they have been extensively enhanced by different modem manufacturers.

The reason why they are called **S**-Registers is a little bit unclear. Some say the **S** stands for modem *settings*. Some say they are just called like this, because they are set and read with ATS... commands. In the common vernacular they were usually termed *storage* registers because they permanently stored the values even through power-off.

Several of the other AT commands also change values of particular S-Registers. There is usually no difference in setting a value directly via an S-Register or via another AT command. It depends on the particular situation which way of setting a register is better.

2 Modem Programming Basics

2.1 Command Reference

In order to program for an actual modem it is a rather good idea to obtain the command reference for that particular modem. Unfortunately, it has become quite common for no-name modems to ship without any kind of usable command reference. Thanks to Windows' Plug & Play feature it is no longer necessary on Windows to know the individual commands. Instead, all that is needed for a modem to run on Windows is to be shipped with the necessary .inf files (often hidden inside some "installer" software, and called a "driver" which is technically not the case, Windows already contains the necessary drivers).

If the modem doesn't come with a command reference the next logical step is to search the web. However, unfortunately, a lot of modem information has vanished from the surface of the earth and the web in recent years. With the rise of broadband Internet connections, modems have become old fashioned devices and many sources are no longer available. It has become more and more difficult to find basic information about particular modem types. Even for modern modems like cell phone modems it can be difficult to find the necessary information.

There are a number of alternatives to obtain a command reference if one doesn't come with the modem:

- Maybe the distributor provides one on its website
- Maybe the OEM manufacturer provides one.
  This requires to identify the OEM manufacturer. A possible way is to use the FCC number of the device, and then looking the original manufacturer up on the FCC web site.
- Maybe the chipset manufacturer provides one.
  Consumer modems are often just build around "off-the-shelf" modem chipsets from larger hardware manufacturers. The cheaper the modem, the more likely it is that the modem manufacturer didn't change anything in the firmware and is using the original example software from the chipset manufacturer. Some chipset vendors provide command references for their modems.
- By looking into the corresponding Windows .inf files it is possible to at least obtain the basic commands
- By using the generic Hayes command reference in this Wikibook module.
- Obtaining the previously mentioned standard documents if there is an indication a particular modem complies to such a command standard.
- Using some kind of *sniffer* program to monitor the communication between the modem and the DTE and reverse engineering the commands using the obtained information. This requires that (a) reverse engineering is legal in your justification and (b) that there is some DTE communication software available that handles the particular modem so there is some valid communication to sniff.

2.2 Setting up a Development Environment

It is highly recommended to spend some preparation time setting up a suitable development environment before starting to write drivers or software for a modem. Most of this consists of hardware set-up.

It is suggested to set up a small network with a "remote" computer and a second modem in answer mode. "Remote" computer in this case means a computer sitting right next to the development machine, but connected via the modems. If a **terminal program** is being developed, the "remote" computer should run some small BBS software (for example), so there is always someone ready to answer, and/or protocol analysis/data dump software. Developing modem software without such a setup can be extremely frustrating. Such a set-up pays off a hundred times in reduced development time and lower stress. Likewise, the modems used should have real speakers, and support ATM*n* commands well enough that you can leave the speaker on for the entire connection process (and ideally have the option to leave it on, period). "Debugging by ear" can be a reality with modems, particularly during compatibility testing.

If possible, a hardware protocol analyser, or at least an RS-232 breakout box, should be obtained. These can be placed between the computers and modems, if needed, to troubleshoot the serial link and ensure that data is, in fact, being transferred between the modem and the computer -- a sanity check which comes in handy far more often than you might expect. Actual hardware protocol analysers are surprisingly expensive, however; old Wyse terminals are not, and are almost as useful for this purpose. If you find one, pick it up. Terminals that support automatic baud-rate detection are particularly useful.

If dialing with the modem also needs to be tested, a small analog PABX for home usage is needed. These PABX units are dirt cheap; an analog PABX for four internal lines and one external line should cost no more than US$50. If dialing is not needed, then the modems should be capable of directly driving a two-wire or four-wire line in **leased-line** mode; otherwise, the PABX is still needed.

Possible setups are for example:

a) Leased-Line Mode

+-------------+ serial +---------+ 2-wire +----------+ serial +----------+ | Development |----------| Modem A |----\/----| Modem B |----------| BBS | | Computer |----------| |----/\---- | (answer) |----------| Computer | +-------------+ +---------+ +----------+ +----------+

or

b) With PABX

+-------------+ serial +---------+ phone wire +------+ phone wire +----------+ serial +----------+ | Development |----------| Modem A |--------------| PABX |--------------| Modem B |----------| BBS | | Computer |----------| |-------------| X |--------------| (answer) |----------| Computer | +-------------+ +---------+ +------+ +----------+ +----------+

or

c) Leased-Line Mode with Protocol Analyser

+-------------+ serial +---------+ serial +---------+ 2-wire +----------+ serial +----------+ | Development |----------| Y Cable |----------| Modem A |----\/----| Modem B |----------| BBS | | Computer |----------| Breakout|----------| |----/\----| (answer) |----------| Computer | +-------------+ +---------+ +---------+ +----------+ +----------+ || || || +----------+ | Protocol | | Analyser | +----------+

Other combinations are of course also useful. And being able to easily reconnect the protocol analyser, e.g. between Modem B and the BBS Computer is helpful, too.

2.3 Operating System, Programming Language & Communication Basics

Before dealing with the details of handling a modem, a few basics should be in place. First of all, the communication with the serial interface should be in place. This includes that the APIs as provided by the particular operating system for serial communication - if any - should be understood. If the operating system doesn't provide such APIs, then it is recommended to first implement the UART access and wrap it into a library, if the serial UART in some hardware is supposed to be programmed directly. Alternatively, a programming language which provides convenient access to a serial interface can be used.

Whatever is used, it should be tested before starting to program for the modem. There is nothing more annoying than not knowing if a particular misbehaviour is caused by a failure in the serial communication with the modem, or is a problem with the modem (usually with the commands sent to it).

Unless in the most simple case, it is suggested to use hardware handshaking with the modem - particularly for speeds greater then 2400 bps or 9600 bps. Therefore, the used low-level serial communication software and hardware should support hardware handshake. If the UART supports some FIFO, like the 16550 UART, the FIFO should be enabled (both for sending and receiving data).

It is undecided if data reception via polling or via interrupts is better. If every incoming byte raises an interrupt there are many interrupts at high communication speeds, and, as surprising as it might sound, polling the UART might be more efficient in such cases.

Communication as supported by a modem is usually half-duplex. Either the DTE or the DCE talks, the other side is supposed to listen. The communication with the modem should best be done with

- 8 Bit
- No parity
- 1 Stop bit

See the next section for speed information.

2.4 Line Speed is not DTE/DCE Speed

+-------------+ DTE/DCE speed +---------+ line speed | DTE / |----------------| Modem / |-------------- | Computer |----------------| DCE |-------------- +-------------+ +---------+

An issue which can be very confusing is the difference between the line speed (the data transfer speed on the telephone line) and the speed on the serial line between the DTE (computer) and the DCE (modem).

First, there is always some general confusion about the line speed, because some line speed is given with taking compression into account, while other data is given without taking compression into account. Also, there is a difference between *bps* and *Baud* due to the modulation schema used on the line. In addition, marketing blurbs obscure the picture. We will not make any attempt to clean up the long-standing Baud vs. bps confusion here (it is hopeless :-)). It is just recommended that whenever the modem returns information about line speed the above mentioned differences are taken into account to avoid any misinterpretation.

Second, the speed on the telephone line does not necessarily have to be the same as the speed on the serial line. In fact, it usually isn't on modern modems. It is recommended to set the DTE/DCE speed to a fixed speed instead of following the line speed. Logically, the fixed DTE/DCE speed should be large enough to cope with the highest expected line speed. V.90 modems should e.g. be accessed via 115200 bps or higher on the serial interface.

Setting the DTE/DCE speed on modern modems is quite simple. They all use autosensing on the serial interface. That is, they themselves detect the speed of data as received from the DTE and use the same speed to return data to the computer. They usually also autosense the parity, and 7 bit / 8 bit data length. Usually modems assume one stop bit when autosensing the serial interface. Therefore it is enought to just configure the serial interface on the DTE to the desired DTE/DCE communication parameters and let the modem figure it out on its own.

Autosensing can fail in rare cases and some modems might have broken autosensing. If a modem tends to fail autosensing it can help to start the initial communication after the DTE is configured with one or more *nop* AT commands

AT<*CR*>

repeated a limited number of times until the modem starts to return

OK

for the *nop* commands.

When a modem sets up a connection with a remote party it can report the used speed. In fact, it can report the line speed or just the DTE speed (some modems can report both). The end user is most probably interested in the line speed, and not the DTE/DCE speed. So from this point of view, it is best to set the modem to report the line speed, and e.g. write the received information to a log file. However, some old communication software or modem drivers interpret the response from the modem as a request to change the DTE/DCE speed. In such cases the modem must be set to always return the DTE/DCE speed. Since this DTE/DCE speed will be the same as detected via autosensing there will be no speed change.

In the rare case that the DTE/DCE speed should indeed follow the line speed, the responses from the modem should of course be set to return the line speed. Then the DTE software has to evaluate the response, and change the DTE/DCE speed accordingly. This is really not recommended these days.

See the #W: Negotiation Progress Message Selection command for details on how to set which response to get.

2.5 Character Set and Character Case

Commands sent to the modem, and textual responses are supposed to be in the ISO 646 character set. ISO 646 is just another name for the familiar 7-bit ASCII character set. Typically, modems chop off any 8th bit in commands they receive anyhow. They interpret the result as if the command has been sent using only 7-bit characters. However, it is not recommended to rely on this, but instead ensure that commands are only sent using 7-bit characters.

Commands are not case sensitive, assuming a modern modem. Some early modems insisted on uppercase-only commands. Still, a generic driver could do worse than ensuring that all commands are sent in uppercase, and all responses are interpreted case-independent. Typically, both letters of the AT command prefix must be of the same case. So AT and at are acceptable, while At and aT are not.

2.6 Welcome to the World of State-Machines

Modem programming means to tap into the world of telecommunications. This is an unknown field for most amateur, as well as professional programmers. Telecommunication is heavily centered around state-machines. And in fact, it is rather difficult or impossible to program a modem without using a state-machine. The modem is at any time in a particular state, and any DTE software which tries to control and use the modem needs to track the state of the modem - in an own state machine. This is necessary, because a Hayes-compatible modem can only do certain things when it is in a certain state. E.g. it can only dial out if it is not already connected to some remote site.

Part of a modem's state can be tracked via particular RS-232 lines. E.g. DCD (data carrier detect) can be used to figure out if the modem has detected a remote modem's carrier signal. Other information is provided by the flow-control lines. However, some states, and associated data need to be tracked via interpreting the modem's result codes.

People unfamiliar with the theory and practice of state machines often try to circumvent the issue by "tough coding". Which means, they throw more and more code onto the problem (wrapped in a heap of if/the/else/otherwise/maybe/... statements), until things seem to work - sort of. If they are lucky they have implicitly managed to create a state machine which works. If they are unlucky, they end up with a partial state machine, which breaks down should something unusual happen in the communication. This usually comes with the problem that the software was not designed to recover if things break down. So such software tends to hang or crash.

It is much more efficient to first spend a few hours to to learn the basics of simple state machines, and then spending a few more hours to describe the communication with the modem as a state machine. The result of this planning serves as a nice template for implementing the DTE software.

3 Flow Control

A slow device needs a way to tell its peer that currently, it is busy, so further incoming data must be stopped until this slow device tells otherwise. This mechanism is provided by flow control. There are two ways of doing flow control: by hardware or software.

3.1 Hardware Flow Control

Hardware flow control is usually implemented using the CTS (Clear To Send) and RTS (Request To Send) lines, which needs separate hardware data lines between devices. This is allocated in the RS-232 cable specification.

Hardware flow control based on DSR (Data Set Ready) and DTR (Data Terminal Ready) is uncommon, particular for modems. It can usually be found at serial printers. Again, DSR/DTR hardware flow control requires additional hardware data lines between devices.

From a programming point of view there is usually not much difference in programming CTS/RTS or DSR/DTR hardware flow control. The hardware has to provide means to drive/read the corresponding signals in the serial interface. If the hardware supports both, CTS/RTS and DSR/DTR flow control, then it is recommended to support both and provide the user with a configuration option.

It should be noted that some hardware or operating system drivers do not provide means to drive/read the less common DSR/DTR combination. If the remote device insists on DTR/DSR flow control a common workaround is to use CTS/RTS in the software, but rewire the cabling so the CTS/RTS wires are in fact connected to DSR/CTS.

3.2 Software Flow Control

This kind of flow control doesn't need extra signal line(s) like hardware flow control, but instead uses special control characters within the data content. To stop further incoming data, the receiving device sends the XOFF character. To enable more data, an XON character will be sent.

However, since the data being sent cannot contain these characters (unless you know that the receiving device ignores such information), binary (non-ASCII) data cannot be transmitted this way. Software flow control is typically used for communications to terminals and other character-based devices. Binary data should not be sent this way as it could, randomly, contain these characters. Hardware flow control using RTS/CTS is usually used.

Helpful Hint: Realizing that the Control Key is a special “shift” key that chops off the 100 bit (octal), it is easy to remember that the ASCII character used for sending XOFF is a Control-S (23 Octal) while the character for XON is a Control-Q (21 Octal). [Think of “S” for Stop and “Q” for Qontinue... don't you spell it that way?]

4 Changing State

4.1 General

Changing the state from command state to on-line state or vice versa is either straightforward or a great mystery. This module covers the more obscure ways.

4.2 On-line State to Command State

It is of course possible to switch from on-line state to command state by dropping the connection (going on-hook in modem terminology). It is also possible to temporarily switch into command state while keeping the connection.

Going on-hook programmatically (and not via dropping a modem control line) requires to first switch into command state while keeping the connection, too.

Switching into command state, while in fact in the middle of transferring data (nothing else is meant with on-line state) requires to send a certain escape sequence as part of the data. This escape sequence is detected by the modem and the modem changes state. Since this character sequence might also be part of the normal data, an additional mechanism is needed to separate the escape sequence from normal data. This is the curse of inband signalling.

The separation of the escape sequence is done by using a so called guard time, which was once patented by Hayes. As a result, some modem manufacturers eliminated the guard time using an alternate escape sequence called the Time Independent Escape Sequence. Anyway, the escape sequence is only recognized by the modem when there was no other data from the DTE (terminal) for at least the duration of the guard time, and when there was no other data from the terminal after the escape sequence for at least the duration of the guard time, too.

An escape sequence consists of three times the same particular character. The character, as well as the guard time is configurable. By default, the character is +, and the guard time is one second. So, with the default configuration, a change to command state requires

<*1 sec. nothing*>+++<*1 sec. nothing*>

If the connection should be dropped, this escape sequence should be followed by the AT command to go on-hook, which is ATH0:

<*1 sec. nothing*>+++<*1 sec. nothing*>ATH0<*CR*>

4.3 Command State to On-line State

The usual way to go from command state to on-line state is via dialing the remote site (see D command). But if the connection already exists, and the modem has been switched to command mode via the escape sequence, the way is different.

If the connection should not be dropped, but instead data transmission should be continued, the ATO0 (letter o, digit zero) command is needed:

<1 sec. nothing>+++*<1 sec. nothing> send a few more modem commands, then go back on-line* ATO0*<CR>*

5 Sync. vs. Async. Interface

6 X.25 Interface

7 AT Commands

The following list is the list of the original Hayes commands. Different modems use slightly different commands. However, this list is supposed to be as "generic" as possible, and should not be extended with modem specific commands. Instead it is recommended to provide such command lists in an Appendix.

7.1 AT Command Format

Here is a summary of the format and syntax of AT commands. Please note that most of the control characters are configurable, and the summary only uses the default control characters.

- AT commands are accepted by the modem only when in command mode. The modem can be forced into command mode with the #+++: Escape Sequence.

- Commands are grouped in command lines.

- Each command line must start with the #AT: Command Prefix and terminated with #<CR>: End-of-line Character. The only exception is the #A/: Repeat Last Command command.

- The body of a command line consists of visible ASCII characters (ASCII code 32 to 126). Space (ASCII code 32) and ASCII control characters (ASCII code 0 to 31) are ignored, with the exception of #<BS>: Backspace Character, #<CAN>: Cancel Character, and #<CR>: End-of-line Character.

- All characters preceding the #AT: Command Prefix are ignored.

- Interpretation / execution of the command line starts with the reception of the first (and also command-line terminating) #<CR>: End-of-line Character.

- Characters after the initial #AT: Command Prefix and before the #<CR>: End-of-line Character are interpreted as commands. With some exceptions, there can be many commands in one command line.

- Each of the basic commands consists of a single ASCII letter, or a single ASCII letter with a &prefix, followed by a numeric value. Missing numeric values are interpreted as 0 (zero).

- The following commands can't be followed by more commands on the command line. They must always be the last commands in a command line. If they are followed by other commands, these other commands are ignored. However, some of these commands take command modifiers and it is possible that a following command is accidentally interpreted as a command modifier. Therefore, care should be taken to not follow these commands with any more commands on the same command line. Instead, they should be placed in an own command line.

  - #A: Answer Command
  - #D: Dial Command
  - #Z: Soft Reset Command

- A command line can be edited if the terminating #<CR>: End-of-line Character has not ben entered, using the #<BS>: Backspace Character to delete one command line character at a time. The initial #AT: Command Prefix can't be edited/deleted (it has already been processed, because upon reception of the #AT: Command Prefix the modem immediately starts command line parsing and editing, but not execution).

- The modem echoes command lines and edits when #E: Command State Character Echo Selection is on (surprise, surprise :-)).

- When echo is on, #<BS>: Backspace Characters are echoed with a sequence of <BS> <BS> (backspace, space, backspace) to erase the last character in e.g. a terminal program on the DTE.

- A command line can be cancelled at any time before the terminating #<CR>: End-of-line Character by sending the #<CAN>: Cancel Character. No command in the command line is executed in this case.

- The #A: Answer Command and #D: Dial Command can also be cancelled as long as the handshake with the remote site has not been completed. Cancellation is done by sending an additional character. In theory, it doesn't matter which character. But care has to be taken that cancellation is not attempted when the handshake has already completed.

In this case the modem has switched to on-line state (#Command State to On-line State) and the character will be send to the remote side. A save way to avoid this problem is to always use the #+++: Escape Sequence followed by going on-hock with the #H: Hook Command Options. If the modem is already in the on-line state, this will drop the connection. If the modem is still in the handshake phase the first character of the #+++: Escape Sequence will cancel the command (and the rest will be interpreted as a normal command line, doing no harm).

- Command line execution stops when the first command in the command line fails, or the whole command line has been executed. Every command before the failed command has been executed. Every command after the failed command and the failed command in the command line has not been executed.

- There is no particular indication which command in a command line failed, only that one failed. It is best to repeat the complete command line, or to first reset the modem to a defined state before recovering from a failure.

- A modem only accepts a new command line when the previous command line has been executed (half-duplex communication). Therefore, care should be taken to only send the next command line after the result code from the previous command line has been received.

7.2 *Command Description Template*

To be removed when all commands are documented.

Syntax:

<The syntax of the command, when necessary in EBNF>

Description:

<Description of the command, including information about the purpose and effects>

Result Codes:

Related Commands and Registers:

- *<Link list of related commands and registers>*

7.3 Special Commands and Character Sequences

See Special Commands and Character Sequences Reference

7.4 AT Commands A - M

See AT Commands A - M

7.5 AT Commands N - Z

See AT Commands N - Z

7.6 AT& Commands

See AT& Commands

8 Result Codes

See Result Codes

9 S-Registers

See S-Registers

10 Advanced Features

10.1 Introduction

Modern consumer modems provide a number of additional features which were originally uncommon for a modem, but became standard features over time. This section provides an overview about how to program these features.

10.2 Fax Class 1

10.3 Fax Class 2

10.4 Voice Services

Serial Programming: Introduction and OSI Network Model -- RS-232 Wiring and Connections -- Typical RS232 Hardware Configuration -- 8250 UART -- DOS -- MAX232 Driver/Receiver Family -- TAPI Communications In Windows -- Linux and Unix -- Java -- Hayes-compatible Modems and AT Commands -- Universal Serial Bus (USB) -- Forming Data Packets -- Error Correction Methods -- Two Way Communication -- Packet Recovery Methods -- Serial Data Networks -- Practical Application Development -- IP Over Serial Connections

11 Text and image sources, contributors, and licenses

11.1 Text

11.2 Images

11.3 Content license