Odyssey Help - Table of Contents

Please select a topic from the list below.

Odyssey Help

Introduction

Select one of the introductory topics listed below:

Introduction

Introducing Odyssey

This introduction will go on to briefly introduce most of the features of Odyssey, however if you are a complete beginner to comms then you may want to read the next section "Introduction to Communications" before returning here. This will help you to better understand some of the terms used.

Odyssey is a comprehensive computer data communications package for the IBM AT and true compatibles running Microsoft Windows™ 3.1 or later, which as well as providing all the standard features of a powerful comms package, will also allow error free communications without requiring an expensive error correcting modem. The inclusion of MNP in Odyssey will allow noise free communications using an ordinary inexpensive modem, even set to its highest speed, with any other system, provided that it also supports MNP or V.42, whether in software like Odyssey, or in hardware. Although several other packages have now copied the Odyssey trick of implementing MNP in software, Odyssey was the first to do so, and in our (naturally unbiased) opinion, is still the best!

As in any Windows application, program functions in Odyssey accessed by means of pull down menus, however the software also provides shortcuts for the most common commands, through through the use of hotkeys and toolbar buttons. Full control over communication and file transfer settings is provided by means of simple configuration dialogs, and context sensitive help is available at all times. The configuration dialogs may also control default directories, colors used by terminal emulators etc.

Among the many features provided are:-

- **Text capture** to a disk file and/or printer (called text logging in Odyssey).
- **Host mode** with two levels of password protection allowing remote systems to dial into your PC.
- **Chat mode** for direct two-way conversations.
- User defined **macros**.
- **Keyboard remapping**.
- A **Command Line Recall** facility, to save typing when online.
- A **scrollback** feature which allows you to "look back in time" at text which has scrolled off the terminal display (called the "Review editor") in Odyssey.
- A **playback mode** which allows you to replay, offline, recorded portions of an online session.
- **Event logging**, which records details of calls, such as service name, date, time and duration. Other transactions recorded are file transfers, fax transmission and reception, script events, and so on.
- A **dialing directory** which allows an unlimited number of entries, and which supports both queued dialing of tagged entries, and automatic redial. You can see statistics on the numbers of calls to a particular service, see when you last called that service, how long you were on for, and so on.
- Several file transfer protocols built in, including **ASCII**, **Xmodem**, **Ymodem**,**SuperKermit**, **Zmodem** and **Compuserve B+**. Zmodem and Compuserve B+ are especially powerful and provide facilities for "no hands" automatic download of files. Both also provide the ability to resume a file transfer which was interrupted by a line drop or power failure; packages which do not offer this feature require you to start the download again from the beginning.
- A multi-window ASCII **text editor**, used for preparing messages offline, and also used to manipulate the contents of the scrollback buffer. Text editing can be performed using CUA commands, or can use

Wordstar™ compatible commands for those more used to the editor provided in Odyssey for DOS. The text editor in Windows Odyssey can handle files of any size - in particular, it does not have the 64k file size limits of the DOS version.

- A comprehensive Pascal-like **script language**, which among other things allows you to automate the dial and log on procedure for any service. This includes a special "**learn**" feature, which makes the script language facility more accessible to those with little or no prior experience of programming, and a **script compiler** utility which allows the use of precompiled scripts for faster loading or greater security (note that use of the compiler is optional).

- Accurate emulations of several popular terminal types, including **TTY**, **VT320**, **VT100**, **VT52**, **ANSI**, **DG200** and **Viewdata** (PRESTEL). Support for further terminal emulations is planned. Although not provided as a separate emulation, note that **VT220** is also supported as a subset of VT320.

- **FAX** modem support, including the ability to send, receive, view, and print FAX documents. You can send text, PCX, and TIFF files as FAXes, and you can export a FAX as PCX or TIFF. Both normal and high resolution FAX formats are supported. The Odyssey FAX server works with any modem which conforms to the EIA Class I or Class II specifications.

Minimum hardware requirements are a 386 based IBM PC or compatible with monochrome display, running Windows 3.1 or later in 386 enhanced mode, with 2Mb of random access memory, a hard disk with at least 3Mb free space, and (if dialup communications is required) an asynchronous modem either attached by cable to a serial port, or on an internal card. However, the ideal machine for Odyssey will have a hard disk with several megabytes of free space (giving room for file transfer downloads), a "Windows Accelerator" type SuperVGA card (for faster scrolling during terminal emulation), and a fast modem with V42bis error correction and data compression.

The serial cable used for connection of external modems to PC/AT or PS/2 machines is normally a simple ribbon cable. See cabling requirements.

Introduction

Introducing Communications

This help chapter is intended as an introduction to the subject of computer *data communications* for the first time user. It includes descriptions of the hardware and software required to allow computers to communicate and exchange data with each other, and information on how to make use of this equipment. It will also introduce some of the terms and ideas which will be discussed in the other chapters of this help document. Although not intended to be a definitive guide to data communications, it should provide the novice with a basic grounding in this exciting and expanding sector of the computing world.

Computer data communications, normally just abbreviated to '**comms**', is the exchange of information between computers which are linked together, either directly by cable, or by means of devices called modems which allow computers to communicate using the public telephone network. Data is exchanged in serial form, a concept which will be explained shortly.

This introduction has been broken down into several subtopics. Please choose from the list below. Alternatively, you may select the "Bits and Bytes" topic, and then follow the browse sequence.

Introducing Communications

Bits and Bytes

The smallest item of information that can be transmitted at one time is one character (the jargon term is 'one **byte**'). A byte is a computer code made up of eight binary digits (called '**bits**' for short) - a bit can be either a zero or a one. Bytes are transmitted to your modem which is connected by cable to the rear of your PC, or mounted on an internal card (whether the modem is internal or external makes no difference to how the computer sees it). Your modem transmits that byte to a remote modem, which passes it to its local computer.

Introducing Communications

Serial and Parallel

The connection between your computer and an attached device can be either '**serial**' or '**parallel**'. In a parallel connection, the byte would be fetched from the computer memory and transmitted in one go, with each of the eight bits going out on a separate wire, the presence or not of a voltage on each wire being used to represent whether that bit is a one or a zero. In a serial connection the procedure is much the same, except that only a single wire is used, so the bits must be transmitted one at a time in single file - in other words they are transmitted in serial. Parallel connections are usually faster, but begin to suffer problems as the distance between the two communicating devices increases - problems such as signals arriving at different times on each wire due to slight differences in their resistances. Serial connections are slower, but are simpler and more reliable over longer distances. This discussion will concentrate on serial connections.

You quite probably know that a serial cable actually uses more than one wire, however the other wires are used for other things. For example, a second wire is used to receive bits, two further wires are used to indicate when each side is ready to receive, another wires provides a reference or "ground", and so on. A standard PC modem cable can get by with about eight wires.

In order for a computer to use a serial connection, something must take on the work of converting parallel data bytes inside the PC into serial form, and vice versa when receiving. While your 80x86 CPU is quite capable of doing this job, it would be a waste of its resources, so instead the IBM PC makes use of a specialised chip called a UART (Universal Asynchronous Receiver/ Transmitter) which handles this job of assembling bits into bytes, and vice versa as its sole purpose in life. You will have noticed the word "asynchronous" pop in there, and we will return to that later.

Introducing Communications

The ASCII Code Standard

Let us leave modems for a moment, and think instead of a problem which is general in communications.

When you tell your computer to print a document from your word processor, you may never have thought about how the computer actually tells the printer what to print. The computer does not (normally) store a picture of each character in memory, instead it represents each character with a code, for example it might use the number 65 to represent the letter 'A'. So, when the computer wants the printer to print the letter 'A', it passes this instruction as a single byte, which contains the value 65, the code for 'A'.

Of course, for this to work, it is important that both computer and printer agree on the codes used to represent letters, otherwise the result will be total gibberish! This problem was apparent from the early days of computing, and so an official standard was born - the **ASCII** standard. This standard defines 128 codes which will be used to represent any letter which might appear in your document, as well as many *control codes* (eg. codes for moving the printer carriage to the left margin, codes for selecting a new page and so on).

ASCII stands for "*American Standard Codes for Information Interchange*", A name which highlights its origins, and also gives a clue to one problem that ASCII has, which is that this was an American standard which the pre-eminence of the American computing industry has forced the rest of the world to adopt, rather than being a truly international standard. This may be problem at certain times because there are characters which do not occur in American text which are sometimes required for other countries, for example the currency symbol of the UK, or non-Roman characters used in many European languages. Although you can often represent these characters on your own computer, and sometimes even on your printer (if it has been modified for the country of use), with communications it is a real problem, since the lack of an accepted standard for such characters makes it totally impossible to (for example) send an email message containing these characters to a computer which either uses different codes to represent those characters, or which might even fail to represent those characters at all.

Although new code standards have been defined by international organisations such as the **ISO**, these have not as yet achieved anything like the universality of ASCII, despite the problems of the latter.

In the previous section you were told that your computer stores one character as one byte, which is eight bits. However, if you know your binary arithmetic you will know that only seven bits are required to represent the 128 standard ASCII codes, so an eight bit byte has the potential to code another 128 characters on top of the standard ASCII character set. The IBM PC makes use of this fact to extend its character set to include ASCII, plus 128 other characters used for national character codes, line drawing and semi-graphic characters. This superset of the ASCII standard is sometimes used by bulletin board systems to create impressive log on banners. Note that Windows documentation refers to this PC extended ASCII character set as the "OEM character set", and uses a quite different eight bit character set (ANSI) for most purposes.

Notice that the ASCII standard is only required if you want to represent readable text on a computer or peripheral device. If the data does not need to be read by humans then the standard codes can be used for something else entirely. You may have come across this distinction with files stored on your computer. For example, if a file (document) is intended to be exchangeable between devices and interpreted according to ASCII then the file will often be referred to as an "**ASCII text file**", or simply as a "**text file**". If the file is not intended to be interpreted as ASCII then it will often be called a "**binary file**". These distinctions also crop up in file transfer, when descriptions talk about "**ASCII File Transfer**" or "**Binary File Transfer**". You should realise that there may be no physical difference in the data, the difference lies solely in how the data is interpreted.

Introducing Communications

Baud Rates

One of the properties of serial connections is the data rate, meaning the speed in bits per second at which data is transferred through the serial port. This bit rate is commonly referred to in the industry as the "**baud rate**", a term which is convenient, but technically inaccurate. However, to avoid confusion, we will adopt this misnomer throughout this document. When a modem is described as being "*capable of 2400 baud*", this means that it can receive and transmit up to 2400 bits per second.

In reality, one *baud* is one change in signal level on a telephone wire, but the level chosen could in fact represent any number of bits, depending solely on the number of different levels which can be distinguished.

For example, if your modem can recognise sixteen different tones, then each tone will represent four bits, and while the baud rate might be 600, the bit rate will be four times that, or 2400 bps. It is easy to see why this is so - receiving any of the sixteen tones can be imagined as selecting one of sixteen possibilities, numbered from 0 to 15 in decimal, or in binary as 0000 to 1111. In other words each tone can be thought of as a way of encoding any combination of four bits. Returning to the confusion of "bit rate" and "baud rate", one bit in fact equals one baud only when the modem distinguishes just two tones.

In real life, modems do use techniques other than simple level (amplitude) modulation to encode signals, but the principle is the same regardless.

Introducing Communications

Synchronous vs Asynchronous

Serial connections can be either "**synchronous**" or "**asynchronous**". A synchronous serial connection is one in which the two sides of the connection exchange timing information as well as data, with the arrival of bits on the interface being synchronised to the time signal. On the other hand, an asynchronous serial connection does not need a separate timing signal; instead the sender transmits an additional bit at the beginning of each byte (called the *start bit*), which alerts the receiver that more bits follow. The sender also transmits either one or two bits at the end of the byte to complete the transfer (called *stop bits*), and this allows the receiver to know that it has received all the intervening bits correctly (if its timing was off, it would probably miss the stop bits). Normally a single stop bit is sufficient. Using this method the bits within a byte must still be synchronised to the baud rate, but unlike a synchronous connection, the interval between bytes is not important. The standard serial adapter in your PC provides an asynchronous serial interface.

Note that an asynchronous serial connection actually sends ten bits (assuming one start and one stop bit) for each eight bit character transmitted, so for example, a 2400 bps connection would actually realise a maximum throughput of 2400 divided by 10, or 240 characters per second.

Introducing Communications

Parity

In many cases, communications will take place via the telephone network, where it is subject to electrical interference, crosstalk, temporary loss of carrier etc. To use the generic term, these communications are subjected to *line noise*. In the early days of communications a primitive error detection system called **Parity Checking** was invented in the hope of trapping some of these errors, so that data affected by noise could be discounted. Parity checking works by reserving one bit of each byte as the "*parity bit*", and this bit is set or reset according to rules which are known to both sides of the link. For example, the two sides might agree to use "**even parity**", which means that the eight bits of transmitted data (including the parity bit), must contain an even number of one bits. If the first seven data bits contain an even number of one bits then the parity bit is set to zero, otherwise it is set to one, to make an even number.

At the receiving end, the computer (in actual fact, the UART) checks that the byte it received contained an even number of one bits. If this is not the case, then one or more bits have been damaged, and the byte is discarded.

Even parity is the most common of parity check schemes, however variations include *odd parity* checking (where there must be an odd number of one bits), and *mark* or *space parity* checking, in which the parity bit is either always one, or always zero.

You may consider that parity checking sounds of limited use, and you may well be right. On the one hand, it does not guarantee to detect all errors. In fact every byte could be in error, but if they all happen to have the correct number of one bits then parity checking will see nothing amiss. On the other hand, even when it does detect an error, it just throws the data away. This data could have come from a space probe, gathered at enormous cost and transmitted the long distance to an earth receiver, only to be wastefully discarded when it arrives.

Parity checking should these days be considered archaic, yet it is still in common use on public communications networks. Where sense prevails, parity checking has been replaced by more sophisticated protocols which not only detect errors, but also correct them - the MNP protocol built into Odyssey is one of these more modern alternatives. Whereas parity checking would be lucky to catch 50% of those bytes in error, the modern protocols use an entirely different system of error detection (called CRC checking) which makes it highly unlikely that a block of data containing an error will pass undetected (less than one chance in 65,000 per block of 1024 bytes is typical).

Another problem with parity checking is inefficiency. This could be the worst sin of all, since as well as having the least impressive error detection capability, parity checking also has the highest overhead. CRC details vary, but the CRC mentioned above would reserve one bit for integrity checking to every 512 bits of data, which compares very well to the one check bit per seven data bits used by parity checking. The bit wasted by parity checks is most often a problem when you want to perform a file transfer, since if the data in the file uses the eighth bit (ie. when transferring binary data), then it is difficult to transfer the file using only the seven usable bits per byte of a parity checked link. This is not impossible to do, it just makes file transfers take longer - wasted time which the user pays for through inflated phone bills.

Introducing Communications

Flow Control

Having connected two systems together using our asynchronous serial interface, the next thing we need to learn about is **flow control**. This is required because the chosen baud rate will at times be too fast for the receiving side to handle. This requires that the receiving side have some agreed signal it can use to tell the transmitter to "hold it a moment!", until it is again ready to resume transfer at the full rate.

There are two basic types of flow control, referred to as "*hardware*" and "*software*" flow control. Strictly speaking the hardware flow control method described below is not standard as far as RS232 is concerned, it is however the de-facto standard implemented by nearly all PC modems and comms software.

The hardware flow control method reserves two of the wires on the serial interface (called **RTS** and **CTS**), RTS being the signal controlled by the computer, while CTS is controlled by the modem. Both signals are normally held high, but when the computer cannot receive more data it signals this to the modem by lowering RTS, and likewise when the modem is unable to receive it signals this to the computer by lowering CTS. When the computer or modem is once more able to receive, it raises its signal to inform the other.

Software flow control sets aside specific bytes which can only be used for stop/start signalling. The bytes used are called **XON** and **XOFF**. When the receiving side does not want to receive more data it transmits the XOFF byte to the sender, and when it is ready to receive again it transmits the XON byte. The characters normally reserved for this purpose are Control-S for XOFF, and Control-Q for XON. There are other variations on the software flow control theme, however these are unusual, and are not implemented in Odyssey.

Odyssey allows you to choose which flow control method you prefer, including no flow control at all. A disadvantage of software flow control is that it is possible for the XON and XOFF signals to be transmitted unintentionally, for example if they happen to occur in a file being transferred. For this reason it is suggested that where possible you use the hardware flow control method. You can also use no flow control at all, but then you must ensure that you never send data to the remote computer faster than it can be handled, otherwise some of the data will be lost.

Introducing Communications

Modems

As was mentioned previously, it is convenient to use the telephone network to connect computers together over large distances. Unfortunately, telephones have been around a good deal longer than computers, and the telephone network was clearly not designed with optimal computer communications in mind. In order to use the telephone system your computer therefore needs some assistance.

The phone system was actually designed to carry voice communications only, that is, sounds (or analogue data in the jargon). Computers on the other hand communicate using binary ones and zeros (or digital data). To use the telephone network you need some device which can convert a computer's ones and zeros into sounds, and at the other end a similar device to convert the sounds back into ones and zeros.

Enter the **MODEM**, a device which does exactly that, and whose name is derived from the two words **MO**dulate and **DEM**odulate, referring to the job it does of converting data from digital to analogue (modulation), and back again (demodulation).

There are many different types of modems available, and ways of connecting them to the telephone network. For most computers the modem is a peripheral piece of equipment, but for the IBM PC and compatibles, there is also the option of a card modem which fits into one of the expansion slots. Alternatively the modem may be built into the computer as standard, as in some laptop portables.

If the modem is stand-alone, then it will have to be connected to the computer through the serial connector (port) on the rear of your machine. If a serial port is not provided as standard in your PC then an adapter card will need to be purchased which contains one before the modem can be connected. Internal "card" modems are essentially a modem built onto a serial adapter card, so internal modems do not require a separate serial adapter.

These days the term "modem" covers a huge variety of widely different devices, from the simplest acoustic coupler, through medium speed interfaces, up to high speed controllers. Most Odyssey users will be interested in the medium speed devices, and we will also briefly describe the acoustic coupler.

The **Acoustic Coupler** is the oldest type of modem, which uses rubber or sponge cups or pads which fit over a standard telephone handset. Except in unusual circumstances these modems are now considered obsolete because of their susceptibility to external sounds and their normally lower data rates (300 baud is common). Nowadays, acoustic couplers are generally used only where a modem cannot easily be plugged directly in a telephone socket, for example when travelling abroad where standards for telephone sockets may differ, or where there are legal restrictions on direct electrical connection of devices not approved by the local telephone network authority.

A typical modern workhorse modem will accept serial data at speeds of 2400 baud or better, convert this data into a form acceptable to the telephone system, and transmit it to a similar modem attached at a remote point in the network. It will also receive data from its opposite number, and pass the reconstructed data to the local computer. Most modems these days offer "**Auto-Dial**" and "**Auto-Answer**" facilities (see the notes on Hayes compatibility in the next topic).

There have been enormous strides in modem technology since 1985. Whereas at one time 1200 baud was considered risky, and 2400 baud foolhardy, these days 2400 baud is commonplace, and better modems are communicating reliably at speeds of 28,800 baud or better. Not so long ago speeds such as these were not even possible on expensive leased telephone lines, designed specifically for computer use.

☞ For the sake of simplicity, the above discussion supposed that a telephone network communicates using sounds. Although this is apparently true from a human, functional point of view, in actual fact we know that the network really handles electrical signals. The microphone in your telephone handset converts sounds into an electrical signal and it is this which is communicated to the remote part of the network. The main difference between a simple acoustic coupler and other modems is that the former does not have a direct electrical connection to the telephone network, and so must use a speaker to convert digital data into sounds, which your telephone handset then converts (along with other sounds coming from the vicinity of the telephone) into electrical signals . In the case of modern modems, these have a direct connection to the telephone network and so simply convert directly from digital data to the electrical signals required by the network, without the problems of noise pollution (these can however suffer from electrical noise).

Introducing Communications

Hayes Compatibility

Some years ago the US manufacturer "**Hayes**" did the industry a service by implementing a standard set of functions within a modem which could be controlled by simple software commands, and which relieved the computer software of many of the fussy details formerly associated with control of the telephone connection. This was the first of the so-called **Smart Modems**. These modems are controlled by a number of commands all beginning with the sequence "AT" (hence the reference to "**AT commands**" you may have heard), and are much simpler to use for complicated tasks such as dialing and answering calls. Where once the computer would have to pulse the transmit line to dial, or monitor several of the wires on the modem interface when it wanted to answer, smart modems implement "**auto-dial**" and "**auto-answer**", which takes care of these tasks without help from the computer software, allowing the software to get on with the more important things.

NOTE: Odyssey does not require Hayes compatibility, although it **is** strongly recommended. Although Hayes compatibility is not mandatory, Odyssey *does* assume that the modem is controlled by means of a similar interface.

Introducing Communications

Remote Dialed Systems

Up to this point, only your own hardware and software has been discussed, yet it is obvious that the type of host system you intend to call should be considered. There are three types of system you might choose to call: another single user, a host computer running some sort of email, bulletin board or similar software, or one of several online database systems.

The single user is simply someone who, like yourself, has access to a computer and modem, and with whom you wish to communicate. Provided that the other user has the appropriate modem (preferably capable of auto-answer), and software, then there is nothing to prevent you from calling that person and transferring files. All that is required is that their computer is set up ready to receive a call (in Odyssey, that is done by placing the software in host mode, or by transmitting the "enable auto answer" command to the modem).

A **Bulletin Board System** (BBS) is really just an electronic meeting place where possibly thousands of people can ask and answer questions, exchange programs and ideas, and generally run up large phone bills. Lists of free and commercial systems in your area can be found in many good quality computer magazines. The commercial systems will normally require that you pay a subscription and/or connect charges to use their systems, and this is often worthwhile, considering the wider range of facilities made available. Most BBS systems assume a simple text terminal for displaying messages, although some support simple graphics which can be displayed if your software is capable of **ANSI** or **Videotex** terminal emulation (Odyssey provides both). Videotex (called **PRESTEL** emulation in Odyssey) displays block graphics similar to television "teletext" services, while ANSI BBS hosts generally assume IBM PC (OEM) compatible character sets and color modes.

Online database providers are all commercial, and these exist to supply you with information - at a price! A wide variety of business, financial, and professional information is available, but obtaining it will normally involve initial connect charges plus a separate charge for accessing the particular information which interests you.

With these different types of systems, running on different computers, you can naturally expect that there are a wide variety of requirements with regard to speed, terminal types, parity checks and so forth, and this is indeed the case. You can deal with most of this variety by entering the appropriate details into Odyssey's **dialing directory**, but the range of speeds you can use are dictated by the type of modem you have. These are some of the many modem standards you might meet, and the speeds they relate to:-

Standard	**Receive/Transmit**
V.21	300/300
V.22	1200/1200
V.23	1200/75
V.22bis	2400/2400
V.32	9600/9600
V.32bis	14400/14400
Bell 103A	300/300
Bell 212A	1200/1200

The latter two are US-only standards, while the remainder are the **CCITT** standards adopted by most of the rest of the world. The only Bell standard which will give you trouble is the 103A (300 baud) system, which is totally incompatible with CCITT V.21. Your modem will have to specifically support Bell 103A or CCITT V.21 if you wish to call such a service. The Bell 212A standard is more or less compatible with CCITT V.22, and many people find that they can call these systems using CCITT compatible modems without much difficulty.

Modern modems often support several of the above standards, and can be set to conform to one or another by means of switches or software commands.

Introducing Communications

Direct Connections

It is not always necessary to use a modem to connect computers together. If the two computers concerned are relatively close to each other (i.e. within 50 feet), then they can be connected using a correctly wired serial cable. Such a cable is sometimes called a "**null modem**", because the cable is wired in such a way that modems are not required. At Skyro Software however we consider the term "null modem" to be rather pretentious, so we will just refer to a "cable".

NOTE: Using a direct connection, speeds of up to 19,200 bits/sec or even more may be achieved, depending on the computers involved, and the software they use. You should be aware however that IBM do not recommend speeds greater than 9600 bps on PCs, or 19,200 on PS/2s.

Introducing Communications

File Transfer

No matter the type of remote system called, a common requirement is the ability to transfer stored information (files) between computers, whether directly connected or connected via the telephone network.

The methods used for **file transfer** are called "**Protocols**", and these are simply a set of rules understood by both sides which govern the process of transferring the data, and making sure that it is stored correctly on the receiving machine. In order to use a particular protocol, that protocol must be supported by both ends of the link, and any requirements of the protocol must be satisfied. For example, the **XMODEM** protocol will not work on parity checked links, or on links using XON/XOFF flow control.

The purpose of a file transfer protocol is to ensure that files are transferred, error free, from one system to another. File transfer protocols also handle pacing (ensuring that the sender is not transmitting data faster than the receiver can write it to disk), and they may also convert files into a form more suitable for the receiving computer, eg. by observing local conventions for end of line markers in text files.

Because of the multiplicity of file transfer protocols available, it is up to the user of the software to ensure that both sender and receiver use the same protocol. This is necessary for successful file transfer to be achieved.

As has been described above, file transfer protocols ensure that files can be transferred without corruption between computer systems, meaning that, during file transfer, the protocol will deal with any line noise errors that occur. However, when posting or reading normal text which is not stored as a file, the problem of line noise may still be apparent, however Odyssey's MNP support will take care of that.

Odyssey Help

Odyssey Configuration

Odyssey is configured by means of a set of dialogs available from the Setup menu, which is itself available on the main menu bar of all Odyssey document window types (see Odyssey and the MDI interface). Once you have made your changes to a dialog you can then save those changes by selecting the Setup|Save settings menu item, or just click the **Save** button available in any of the dialogs.

Setup|Communications
Setup|Modem...
Setup|General...
Setup|Editor
Setup|File transfer...
Setup|Terminal emulation...
Setup|FAX...
Setup|Host mode...
Setup|Keyboard macros...
Setup|Printer...
Setup|Save settings

☞ In modem control and init strings, as elsewhere in Odyssey, the vertical bar character '|' (ASCII 124) is used to represent carriage returns, ie. is equivalent to pressing the <Enter> key at that point if you were typing the modem command manually. Don't forget to add these, and don't delete them from existing strings! Another character which you can use in modem control strings is the tilde '~', which represents a delay of 0.5 seconds.

Odyssey Configuration

Setup|Communications Dialog

The **Setup|Communications** dialog is actually just one page of the Odyssey Configuration dialog, which is a multiple page tabbed dialog. The tabs are shown down the right hand side of the dialog - to change pages, just click on a white tab.

The **Setup|Communications** dialog is used to configure the baud rate, parity, flow control and other communication settings which Odyssey will use when the PC is *directly connected* to another PC or when Odyssey is being used for terminal emulation to a *directly connected* host device (see the note at the end of this topic for why *directly connected* is emphasised).

The fields on the dialog are as follows:-

Comm port: Selecting this field pulls down a listbox which offers you a choice of comm ports, including NUL0: (a null port) or real ports named COM1 to COM8. Not all of these ports are necessarily available for use on your machine - Odyssey will check that the serial port actually exists on your system, and is not in use by another application, before allowing you to use it. If Odyssey determines that the chosen comm port does not exist, then it will switch to the NUL0: port. The **NUL0**: port is an imaginary port which essentially just gives Odyssey something to attach itself to when it isn't attached to a real port. The NUL0 port is also sometimes useful to users who want to select a port after startup, eg. under script control.

Baud rate: This option has ten possible settings from 300 to 115200 bits per second and is the rate at which data is sent or received. You may use any setting which is acceptable to the device with which you wish to communicate. Notice that Odyssey does not offer a 1200/75 bps option (V.23). This is because Odyssey only supports V.23 modems which are speed buffered, and to use such a modem you should set the port for 1200 baud full duplex (and remember to enable RTS/CTS flow control).

Speeds above 19200 bps may not be reliable on your hardware. You can usually cure such problems by purchasing a replacement serial board for your machine which incorporates the more modern **NS16550AFN** UART. This device is capable of buffering interrupts for longer than the standard 8250 with which it is otherwise compatible, which makes it ideally suited to multitasking systems such as Windows.

Data bits: Different hosts will have different expectations regarding the number of significant data bits in each byte that you transmit to them. Some expect seven data bits per byte, and some expect eight - this option allows you to select which setting you wish to use. For most BBS systems eight data bits is normal. As an alternative to setting seven data bits, you might wish to consider leaving data bits set to eight, and enable the "**Strip parity**" option either in the Setup|Terminal dialog, or for individual services using the appropriate dialing directory entry.

Stop bits: In asynchronous communications, start and stop bits are added to each character transmitted (see "Introduction to Communications"). Some systems will use a single stop bit while others use two. Most systems use one stop bit.

Parity: This option has three settings (***None***, ***Even*** or ***Odd***). In most cases *None* will be used, ie. no parity, however some systems may require *Even* parity. Note that any parity setting other than *None* normally implies seven data bits. However, Odyssey does not enforce this, except when parity is set using the dialing directory.

Flow control: Odyssey supports both hardware and software flow control. Hardware flow control (Rts/Cts) is normally used with direct PC to PC file transfer or when connected to modems which provide internal buffering (which means nearly all modems these days). Software flow control (Xon/Xoff) is normal in situations such as a direct connection to a mainframe computer system. Avoid using software flow control unless strictly necessary, since that type of flow control upsets many file transfer protocols. In particular, you would not be able to use Xmodem or Ymodem file transfer over a link which uses Xon/Xoff flow control, and even when the protocol supposedly supports software flow control, (for example, Zmodem or Kermit) there is little point in giving the remote software the chance to mess things up unnecessarily.

Detect modem at startup: Odyssey normally attempts, during startup initialization, to confirm that a modem is switched on and connected to the port which Odyssey is configured to use, and will display an alert dialog if a modem is not detected. The check involves transmitting the init string to the modem, and looking for an echo. An echo normally means that a modem is connected, although it *is* sometimes possible to get an echo from an incorrectly wired serial cable (ie. one in which the receive and transmit wires are crossed). If you do not want Odyssey to carry out the "modem present" check (eg. because you are not using Odyssey with a modem) then you should disable this option. Odyssey will not carry out this check, regardless of the setting of this option, if your init string is blank, or if Odyssey is configured to use the NUL0 comm port.

Click on **OK** to confirm changes to this dialog, and any other dialog pages you have visited. Click on **Cancel** to cancel changes to all dialog pages you have visited. Click the **Save** button if you wish to make your changes last beyond the current Odyssey session. Clicking the **Help** button displays this help topic. Click on one of the tabs on the right side of the dialog to flip to another dialog page.

The Pick Modem button is available in all Odyssey configuration dialog pages, and is described separately.

Note that communications settings such as baud rate and parity are overridden when you connect to a service using the dialer. If you are intending to edit the communications settings for a particular dialed host, then you should edit the dialing directory entry for that service, rather than using the **Setup|Comms** dialog.

Odyssey Configuration

Setup|Modem (Dial Commands) Dialog

The **Setup|Modem (Dial Commands)** dialog is actually just one page of the Odyssey Configuration dialog, which is a multiple page tabbed dialog. The tabs are shown down the right hand side of the dialog; to change pages, just click on a white tab. Note that although there is only one entry point from the menu for modem configuration, modem configuration has too many options and so had to be spread over four dialog pages - note the **Dial cmds**, Connect/Fail, Init strings and Num prefixes tabs.

Tone dial prefix, Pulse dial prefix: The Dial Prefixes allow you to tell Odyssey what command it should transmit to the modem to make it dial a number. Odyssey will normally follow the selected prefix with the number to be dialed, and then the *Dial Suffix*. For example, on Hayes compatible modems the tone dial prefix is normally "**ATDT**", and the pulse dial prefix is normally "**ATDP**". The Dial Mode option (discussed below) selects which of these prefixes Odyssey actually sends to the modem. See also: How Odyssey constructs the dial command.

Dial Suffix: The Dial Suffix allows you to tell Odyssey what code or string to issue to the modem to complete a dialing command. Before transmitting the suffix Odyssey will have issued the appropriate dial prefix and the number to dial (see How Odyssey constructs the dial command). For Hayes modems all that is required for a suffix is the end of line marker ("|" in Odyssey). Beware of leaving this marker out, as that will almost certainly cause some very strange effects, such as the modem not executing the dial command until after the Odyssey dialer window has timed out.

Dial Mode: The Tone/Pulse dial prefix fields provided the modem command strings for tone and pulse dialing. However, Odyssey also needs to know which one it is supposed to use. If your phone uses tone dialing then set this option to *Tone*. If your phone uses pulse dialing then set this option to *Pulse*.

Hangup command: The Hangup command field tells Odyssey the modem command which should be used to hang up the line. For Hayes modems this is usually "**~~~+++~~~ATH0|**". Currently this command is used only by the Odyssey dialer, when it wishes to reset the telephone line after an unsuccessful dial attempt. The **ALT+H** command in Odyssey uses the RS-232 standard method of hanging up the line, that is, by dropping the DTR signal for roughly half a second. Note that this means that your modem must not be configured to ignore DTR. If you want Odyssey to *always* use DTR to hang up, then just delete all the characters in this string (don't just overtype them with spaces). This trick can be handy if your modem has the "BLACKLIST" feature required by some public telephone authorities, since most such modems provide an option to zap the blacklist when DTR is toggled.

Connect timeout: This is the number of seconds allowed following the issue of a dial command before Odyssey decides that the attempted connection has failed. This timeout will only be reached if Odyssey has not received either the connect message or one of the connect failure messages in the intervening period. The default timeout should be long enough for most national calls using pulse dialing. If you wish to make international calls then you may need to configure a longer timeout period, or if your local exchange supports tone dialing then you may use a shorter timeout. Remember that connection times vary, even on the same number. If you make the timeout too short then you may find Odyssey often fails to connect in time. You should be able to recognise this quite easily, if it happens.

Maximum Redials: If Odyssey fails to connect with a service at the first dial attempt then it will retry, up to the maximum number of retries entered here. Notice that this is a *RE*dial maximum. Odyssey will always attempt the number at least once, and then retry as many times as are specified here. If Odyssey is dialing several numbers from a dial queue then it will *not* persist with one number up to the retry limit, but will instead cycle around the items in the queue until each has been dialed the maximum number of times, or until a connection is established on one of the numbers.

Delay before redialing: Sometimes your modem needs a little time to recover between dial attempts. If so, then you can control how much breathing time Odyssey gives it by changing this setting. The default is one second.

Error correction done by: Odyssey provides an option in each dialing directory entry which enables or disables error correction for individual services. If you enable error correction for a service Odyssey then needs to know whether that error correction is to be carried out by your modem, or whether you want to use Odyssey's internal software MNP5 implementation. Check the 'Modem' option if your modem supports hardware MNP5 (or better) or V42bis - otherwise check the 'Software' option, which enables the use of Ody's software MNP. Note that if your modem supports an inferior MNP level (eg. MNP2) then you will still want to use Ody's software MNP, but you will need to give Odyssey the strings your modem uses for its "Disable Error Correction" command. That string is configured in the "Init Strings" page of this dialog.

Always init before dialing: When Odyssey is asked to dial several numbers in a round robin queue, it starts by sending an init string appropriate for the first service to be dialed. Thereafter (assuming it doesn't connect) it will only send a new init string to the modem if the requirements of the next service in the queue differ from the previous one in some way, eg. in error correction. However, for some users, this "smart init" can be somewhat *too* smart, because modem settings can also be changed using the "number prefix" feature, a change which will not be reset unless Odyssey sends another init string. However, sending an init string before each and every dial attempt will slow down the dialing routine, perhaps unnecessarily. To resolve this conflict you are given the choice: if you want Odyssey to send the init always, before every dial attempt, then enable this checkbox field. If you are not embedding modem configuration commands in telephone numbers in the dialing directory then leave this option disabled.

Click on **OK** to confirm changes to this dialog, and any other dialog pages you have visited. Click on **Cancel** to cancel changes to all dialog pages you have visited. Click the **Save** button if you wish to make your changes last beyond the current Odyssey session. Clicking the **Help** button displays this help topic. Click on one of the tabs on the right side of the dialog to flip to another dialog page.

The Pick Modem button is available in all Odyssey configuration dialog pages, and is described separately.

Odyssey Configuration

Setup|Modem (Connect Msgs) Dialog

The **Setup|Modem (Connect Messages)** dialog is actually just one page of the Odyssey Configuration dialog, which is a multiple page tabbed dialog. The tabs are shown down the right hand side of the dialog; to change pages, just click on a white tab. Note that although there is only one entry point from the menu for modem configuration, modem configuration has too many options and so had to be spread over four dialog pages - see the Dial cmds, **Connect/Fail**, Init strings and Num prefixes tabs.

Odyssey dials a number by transmitting a dial command to the modem (see How Odyssey constructs the dial command), and then waiting for some kind of reply, indicating either success or failure of the dial attempt. This dialog page allows you to tell Odyssey what strings might be expected in that reply, and whether it means success or not. Odyssey allows you to enter one "success" string, and up to six "failure" strings. If you don't need all six of the latter then just leave the unused fields blank.

Connect OK: Is the "success" response. For Hayes modems this is normally "**CONNECT**". Note that you should *not* specify a complete response line, eg. "CONNECT 2400" as this would prevent Odyssey from recognising other equally valid success messages, such as "CONNECT 1200". See notes elsewhere about baud rate detection in Odyssey.

Connect fail 1...6: are the failure messages which might be issued by the modem when a call fails to connect (eg. "BUSY" or "NO CARRIER"). When Odyssey is waiting for a connection to be established it ignores anything received from the modem unless it is either the connect string described above, or one of the failure strings.

Click on **OK** to confirm changes to this dialog, and any other dialog pages you have visited. Click on **Cancel** to cancel changes to all dialog pages you have visited. Click the **Save** button if you wish to make your changes last beyond the current Odyssey session. Clicking the **Help** button displays this help topic. Click on one of the tabs on the right side of the dialog to flip to another dialog page. The Pick Modem button is available in all Odyssey configuration dialog pages, and is described separately.

Beware of modems which issue a friendly message such as "Waiting for Connection" while it dials the number. The first seven letters of "Connection" would deceive Odyssey into thinking that a connection had been established, when in fact it hasn't. You can check this by issuing a dial command manually from the terminal keyboard ("ATDT"<number><enter>) if you think you may have this problem. All modems we have come across which display this message also have an option to disable it - an option which you should avail yourself of!

Odyssey Configuration

Setup|Modem (Init strings) Dialog

The **Setup|Modem (Init strings)** dialog is actually just one page of the Odyssey Configuration dialog, which is a multiple page tabbed dialog. The tabs are shown down the right hand side of the dialog; to change pages, just click on a white tab. Note that although there is only one entry point from the menu for modem configuration, modem configuration has too many options and so had to be spread over four dialog pages - see the Dial cmds, Connect/Fail, **Init strings** and Num prefixes tabs.

An **init string** is a command which Ody sends to the modem in order to ensure that the modem is configured as we require. Odyssey has two kinds of init string :-

- The main "Init string" is transmitted once, when Ody first runs, just to make sure that the modem is configured to Odyssey's expectations, and not those of some other package - you can also force this string to be transmitted by typing **ALT+J**, selecting the Command|Re-initialize modem menu item (when the terminal window is active), or by using the ModemInit() command from a script.
- Dialer init strings (including the error correction init strings) are transmitted just before Odyssey dials a number, and these configure the modem to the requirements of a particular service. Additionally, Odyssey uses the reply from the dialer init string as an indication that the modem is attached and working, since if the modem *isn't* working then there isn't any point in continuing with the dial attempt.

The fields on the **Setup|Modem (Init strings)** dialog page are described below. Note also the buttons on this dialog :- Click on **OK** to confirm changes to the dialog, and any other dialog pages you have visited. Click on **Cancel** to cancel changes to all dialog pages you have visited. Click the **Save** button if you wish to make your changes last beyond the current Odyssey session. Clicking the **Help** button displays this help topic. Click on one of the tabs on the right side of the dialog to flip to another dialog page. The Pick Modem button is available in all Odyssey configuration dialog pages, and is described separately.

Init string

This is the main init string mentioned above. It should contain any commands which set required defaults in the modem for the current session. Odyssey by default inserts whatever init string has been defined for your modem in the ODYSSEY.MDM file. Some users may prefer to have no init string at all, and instead depend on the settings stored in the modem non-volatile RAM. If your modem does not offer such a feature then you may be required to define an init string.

The exact meaning of "required defaults" depend on the modem. Assuming a Hayes compatible modem however, those defaults would include :-

- Verbal result codes
- Commands should be echoed
- DCD should represent the true state of the carrier
- DTR should not be ignored
- Internal modem error correction should be disabled if you wish to use Odyssey software MNP.

A typical init string for a vanilla Hayes compatible modem might be "**ATE1V1Q0X4S0=0|**".

Dialer init string

The dialer init string is a separate initialization string which is sent to the modem before each dial attempt. For Odyssey, the purpose of this string is mainly to elicit an "OK" response from the modem to confirm that it is alive and well before the dial command is sent. However, if you need or prefer some pre-dial initialization, then this is the place to put it.

This string is *not* optional. There must be something in here which will provoke a response from the modem, otherwise Odyssey will fail the dial attempt with a "Modem not responding" error message.

A typical pre-dial initialization would set a default operating mode in the modem prior to dialing a number, for example you may wish to use "ATZ|" to restore factory defaults changed during a previous call. The default dialer init string is "AT|", which does nothing except cause a Hayes compatible modem to return an "OK" in reponse.

Although still required, the old dialer init string has been superceded to a certain extent by the "error correction" strings which are described below. In previous Odyssey versions, the dialer init string would always be sent before dialing. However, in the current version, Odyssey *selects* an appropriate init string (either this one, or one of the error correction strings) and sends that instead. See How Odyssey Dials a Number.

Un-init string

This field is for future expansion - it is not currently used.

Auto-answer on/off

These fields let Odyssey know the correct strings to use when instructing the modem whether or not to auto-answer calls. The defaults...

Auto-answer on	-	**AT|~ATS0=2**
Auto-answer off	-	**AT|~ATS0=0**

...are correct for Hayes compatible modems - refer to your modem manual for the equivalent command if yours is not Hayes compatible. Odyssey sends the "Auto-answer on" string when entering host mode mode, and sends the "Auto-answer off" command when leaving host mode. Odyssey also sends these strings when a script uses the AutoAnswer(TRUE|FALSE) script command.

Disable Error correction
Enable error correction, disable compression
Enable error correction and data compression

For an explanation of why these fields exist, see Odyssey and Error Correction.

The "**Disable Error Correction**" string is the command which Odyssey sends to the modem when it wants to disable hardware error correction. Odyssey sends this string to the modem prior to dialing a service which (according to its dialing directory entry) does not support error correction. Odyssey also sends this string to the modem if the service *does* support error correction, but Odyssey software MNP is preferred (eg. to disable MNP in an old modem which only supports MNP2). If your modem has no error correction then this string may be left blank.

The "**Enable error correction, disable compression**" string is the command which Odyssey sends to the modem when it wants to enable error correction, but disable data compression in the modem - which it will do if the service to be called has "*On (no compression)*" as its Error Correction setting in the dialing directory. You might wish to disable compression in this way, for example, if you have an MNP5 modem, but intend to call a service to perform a long file transfer of compressed files (ARC, ZIP, LZH etc). MNP5 actually slows down the transfer of such files, whereas MNP4 (ie. error correction, but no compression), would not. If your modem supports V.42bis data compression then this distinction is less important, since

V.42bis is smart enough not to send data which has actually been enlarged by its "compression" routines. This string is not required, and may be left blank, if Odyssey software MNP is used or your modem does not have hardware error correction.

The "**Enable error correction and compression**" string is the command which is sent to the modem when both hardware error correction and data compression is required - which will be the case if the service has "*On (with compression)*" as its Error Correction setting in the dialing directory. This string is not required, and may be left blank if Odyssey software MNP is used or if your modem has no error correction of its own.

Odyssey Configuration

Setup|Modem (Number prefixes) Dialog

The **Setup|Modem (Number Prefixes)** dialog is actually just one page of the Odyssey Configuration dialog, which is a multiple page tabbed dialog. The tabs are shown down the right hand side of the dialog; to change pages, just click on a white tab. Note that although there is only one entry point from the menu for modem configuration, modem configuration has too many options and so had to be spread over four dialog pages - see the Dial cmds, Connect/Fail, Init strings and **Num prefixes** tabs.

Number prefixes allow Odyssey to handle very long (international) telephone numbers, or to handle special service specific modem initialization commands. A prefix is embedded in a dialing command whenever Odyssey meets a special symbol in the number it is asked to dial.

Up to ten prefixes (identified as A to J) are allowed, each up to 20 characters long. You use the prefix by embedding a symbol such as **@A** in the telephone number field of the dialing directory entry. The valid symbols are in the range **@A** to **@J**. For example you could have the following entry as a telephone number:

@B-5678

and when Odyssey dials, the **@B** symbol will be replaced in situ with the string you have defined as prefix B. For example if your dial prefix was "ATDT", your dial suffix "|", and prefix B was defined as "091-234", then the resulting dial command which is sent to the modem would be:-

ATDT091-234-5678|

Alternatively, you can use symbols **%A** to **%J** instead. Odyssey handles this slightly differently in that the prefix becomes a true prefix, ie. it is inserted at the start of the dial command rather than where the symbol occurred. For example, if the dial prefix and suffix were as above, and prefix A were defined as **ATZ|~~**, and the telephone number field was **%A091-234-5678** then the resulting modem dial command would be:-

ATZ|~~ATDT091-234-5678|

Using the second form of number prefix, it is not actually important where in the telephone number the prefix code appears, since it always results in the prefix being inserted at the beginning of the dial command string. When using this feature you should bear in mind that Odyssey examines the telephone number from left to right, and when it meets a '%' symbol the corresponding prefix is fetched and inserted at the start of the dial command - which means that if you have more than one '%' expansion symbol in the telephone number then the corresponding prefixes will appear in the dial command in reverse order of their appearance in the number.

Click on **OK** to confirm changes to this dialog, and any other dialog pages you have visited. Click on **Cancel** to cancel changes to all dialog pages you have visited. Click the **Save** button if you wish to make your changes last beyond the current Odyssey session. Clicking the **Help** button displays this help topic. Click on one of the tabs on the right side of the dialog to flip to another dialog page. The Pick Modem button is available in all Odyssey configuration dialog pages, and is described separately.

Odyssey Configuration

Setup|General Dialog

The **Setup|General** dialog is actually just one page of the Odyssey Configuration dialog, which is a multiple page tabbed dialog. The tabs are shown down the right hand side of the dialog - to change pages, just click on a white tab.

The fields on the **Setup|General** dialog are as follows:-

"Directories for..." panel

The fields in this panel tell Odyssey where to find, or where to put various files.

Directory for Downloads: You can instruct Odyssey to always store downloaded files in a particular directory, the one set up using this option. If this entry is blank then downloaded files are stored in the Odyssey directory instead. Note that if you nominate a directory then this directory *must* exist - Odyssey will not create it for you.

Directory for Uploads: If you ask Odyssey to upload a file, and do not specify a path, then Odyssey will look for the file in the directory you specify here. If you do not nominate a directory here then the Odyssey home directory is assumed. Again, Odyssey will not automatically create the upload directory.

Directory for Scripts: When you pull down the Terminal window Command menu, Odyssey displays a list of scripts you may run. When you dial a number from the dialing directory, Odyssey looks for a script whose name matches the directory entry key, and runs that script automatically. In both cases, Odyssey needs to know where to look for scripts. If you leave this field blank then Odyssey expects to find all scripts in the Odyssey home directory. Note that Odyssey will only search one directory at a time for scripts.

"Default Fonts" panel

The fields in this panel tell Odyssey which fonts you prefer it to use in various windows.

Default fixed width font: The contents of this field tells Odyssey which Windows font to use in any window which requires a fixed width font. Currently, the only applicable window type is an Odyssey text editor window.

Default variable width font: The contents of this field tells Odyssey which Windows font to use in any window which *can* use a variable pitch font - note that you can select a fixed width font for this field also. Amongst other window types, this setting affects the dialing directory "list of services" panel, dialer 'dial queue' dialog, file selector listbox, and Fax server listbox.

Note that Odyssey uses a special logic to select the specialized font used in the terminal window, and hence neither of the above settings affects that window. Likewise, for the status line Odyssey must select a font which will fit, and so the status line is also unaffected by these settings.

"Miscellaneous options" panel

MNP smoothing: This setting is ignored unless you are using the software MNP feature. MNP receives data in chunks called packets, and if the speed that the packets are displayed is significantly faster than they are received then there will be a pause between terminal updates which can make the display look jerky, and which some people find annoying. Odyssey can attempt to reduce this effect, however that sometimes means that throughput is not quite as high as it may have been without smoothing. Use this field to enable or disable the MNP smoothing feature.

Baud rate detection: This option affects the Odyssey dialer in situations when a dialed connection is established at a speed not equal to the current terminal rate (the speed between PC and modem). Enable this checkbox if you want Odyssey to adapt the terminal rate to the new connect rate, disable this checkbox if Odyssey should leave the terminal baud rate alone. The latter is normal for speed buffered modems, although the former may make file transfer more reliable on older modems if no flow control is enabled. This option applies both to outgoing calls in terminal mode, or incoming calls during host mode. See also: Baud rate detection.

Event logging: Event logging (sometimes known as "call logging" in other packages) is the recording of a history of activities which the user might wish to keep track of. For example, recording the duration of every phone call allows the user to check phone bills. The Odyssey event log is stored in a file called **ODYSSEY.REC**, and contains details of call establishment and termination, file transfer file names and throughputs, Fax transmissions and receptions, and so on. Since the call log grows indefinitely you may wish to maintain it at regular intervals, either by pruning, deleting or moving to external storage. Alternatively, you can disable event logging using this option.

Raw logging mode: Odyssey normally filters unusual control characters from logged text, because they may upset your offline text editor. However, it is sometimes useful to be able to produce an unfiltered log file when you want an exact copy of the received data, for example when logging the stream of terminal control sequences to an Odyssey terminal emulation.

If Raw Logging Mode is disabled, and Odyssey is emulating a non-TTY terminal, then Odyssey will attempt to strip out terminal control sequences, leaving readable ASCII text. When Raw Logging is enabled Odyssey will leave control characters intact, regardless of the emulation in use.

Warning: The Odyssey editors, in common with most other text editors, assume that the file you are editing consists entirely of ASCII text. Data received using Raw Logging mode may not be usable in the Odyssey text editor.

Enable file transfer sound: If enabled, this setting tells Odyssey to play an audible alarm whenever a file transfer completes or is terminated for some other reason. The alarm normally sounds for several seconds, but may be interrupted by pressing a key.

Enable keyboard bell: If this checkbox is enabled, then a received ASCII 7 character (BEL or Control-G) will cause a beep sound.

Confirm deletes in dialing directory: Odyssey allows you to delete dialing directory entries by pressing the delete key or by clicking the 'Cut' button. By default, Odyssey does not ask for confirmation before carrying out the deletion, since the deleted entry can always be pasted back. However, some users may prefer that confirmation is requested. If you are worried by the "cat sitting on the keyboard" scenario, then enable this checkbox.

Click on **OK** to confirm changes to this dialog, and any other dialog pages you have visited. Click on **Cancel** to cancel changes to all dialog pages you have visited. Click the **Save** button if you wish to make your changes last beyond the current Odyssey session. Clicking the **Help** button displays this help topic. Click on one of the tabs on the right side of the dialog to flip to another dialog page.

The Pick Modem button is available in all Odyssey configuration dialog pages, and is described separately.

Odyssey Configuration

Setup|Editor Dialog

The **Setup|Editor** dialog is actually just one page of the Odyssey Configuration dialog, which is a multiple page tabbed dialog. The tabs are shown down the right hand side of the dialog - to change pages, just click on a white tab.

The fields on the **Setup|Editor** dialog are described below. These settings control the initial settings of various text editor options - when a new text editor window opens its options will be set according to the value of the fields described here. Note that these options can also be changed dynamically, inside individual text editor windows.

Editing Commands: The Odyssey text editor can be operated using either conventional Windows™ (CUA) keyboard commands, or by WordStar™ compatible commands. Click the appropriate radio button to select the command set you prefer.

Keep backups: When saving a file, the Odyssey text editor first renames the existing file as *filename*.BAK, but only if that feature is enabled here.

Insert mode initially...: When you type a character in *insert mode*, existing characters on the line are moved right to make room for the new one. The opposite of insert mode is *overtype mode*, in which new characters replace existing ones. Click the appropriate radio button to control which mode Odyssey defaults to when a new text editor window opens.

Auto-indent initially...: The *auto-indent* feature in the Odyssey text editor applies when you press <enter> to begin a new line. In auto-indent mode the new line has the same indent as the line just completed, whereas when this mode is disabled, the new line will begin at column one. Check the 'On' radio button to enable this feature.

Word-wrap initially...: The Odyssey text editor supports *word wrap*, which means that a new line will automatically be started if you continue typing beyond the current right margin (the word being typed at the time is automatically moved to the new line). Also, the text editor "*Reformat Paragraph*" command is only enabled in word wrap mode, to avoid accidentally reformatting paragraphs on an inappropriate file, eg. an Odyssey script source file.

Right justify initially...: The *right justify* option applies in word wrap mode. If right justification is enabled then the Odyssey text editor will adjust the spacing between words in the line just completed so that the line exactly meets the right margin. The same thing applies when Odyssey executes the "Reformat Paragraph" command.

Tab fill character: This option controls the text editor "*hard tabs*" vs "*soft tabs*" feature. If the tab fill character is <space> then soft tabs are in use, and when you type the tab key the text editor fills the interval between the cursor position and the next tab stop with spaces. In hard tab mode (tab fill character is <Tab>) the text editor fills the interval with the optimal number of tabs and space characters.

Tab spacing: This option may be set to "*Smart*" or "*Fixed*". In smart tabbing mode the editor treats the column numbers of words on the previous line as the tab stops for the current line - a feature which is very useful when editing tables etc. Fixed tab mode is the conventional tabbing mode in which tab stops are set at regular intervals - the exact interval being defined in the Tab width field described below.

End of line marker: The Odyssey text editor can read ASCII text files whose lines end in CRLF (the normal DOS convention), or LF (the convention for files originating on Unix systems). It can even handle files which use a mixture of both (files captured by the text logging feature have a tendency to do this). However, if when editing a file you decide to insert a new line, the text editor needs to know whether you prefer that line to be terminated by LF or CRLF. The setting of this option decides that issue.

Tab width: This setting applies if *Tab Spacing* is set to *Fixed* (see above), and determines the interval of the fixed tab stops.

Right margin at col: This field sets the right margin column number, which is used when word wrap mode is enabled.

Click on **OK** to confirm changes to this dialog, and any other dialog pages you have visited. Click on **Cancel** to cancel changes to all dialog pages you have visited. Click the **Save** button if you wish to make your changes last beyond the current Odyssey session. Clicking the **Help** button displays this help topic. Click on one of the tabs on the right side of the dialog to flip to another dialog page. The Pick Modem button is available in all Odyssey configuration dialog pages, and is described separately.

Odyssey Configuration

Setup|File Transfer Dialog

The **Setup|File transfer** dialog is actually just one page of the Odyssey Configuration dialog, which is a multiple page tabbed dialog. The tabs are shown down the right hand side of the dialog - to change pages, just click on a white tab.

The **Setup|File transfer** dialog is used to set options supported by some of Odyssey's file transfer protocols. The dialog includes panes for ASCII, Zmodem and Compuserve B+ options, and one checkbox which applies to all protocols, including those protocols which do *not* have options in this dialog.

The one checkbox which applies to all protocols is the **View GIF/JPEG images while downloading** option. If checked, this option means that whenever Odyssey is downloading a .GIF or .JPG file, it will open a bitmap viewer window and show you the picture building up as the download progresses - this works especially well with interleaved GIF images. Although this feature works only with GIF and JPEG images at present, we may extend it to other image formats in future (note that this feature is only practical on image types which always specify required information such as color palette etc ahead of the bitmap data itself - so the feature will never work with formats such as TIFF or PCX).

Zmodem options...

This panel allows you to change some of the parameters which control Zmodem file transfer.

Auto-download: If enabled, then Odyssey will automatically recognise an incoming Zmodem header and invoke a Zmodem download without user intervention, thus saving the user the trouble of selecting file transfer at both ends of the serial link. If you intend to invoke a Zmodem download explicitly from a script then you should think about disabling this option, otherwise your script is likely to be pre-empted by the auto-download feature - this effect may be desirable, but you should be aware of the possibility of it happening. This option is enabled by default.

Full streaming: If enabled, the remote Zmodem system can send at full speed without waiting for Odyssey to write data to disk. If the disk drives on your PC are particularly slow, it would be best to disable this option - this is rarely required.

Escape control codes: If it is required that all control codes be escaped, for example when using a network which is not completely transparent to control codes, then enable this option. Zmodem always escapes the most commonly troublesome characters, such as Xon, Xoff etc, but this option tells Zmodem to escape *every* control code. This decreases throughput, so should only be used if truly necessary.

Compuserve B+ options...

This panel allows you to change some of the parameters which control Compuserve B+ file transfers.

Auto-start: This option is similar to the "Auto-download" option provided in the Zmodem panel. If enabled, Odyssey will automatically recognise when the Compuserve host wishes to begin a file transfer, and will invoke the Odyssey implementation of the CIS B+ file transfer protocol with no need for further intervention from the user.

☞ Compuserve uses a rather silly choice of character sequence to auto-invoke the B+ protocol; the sequence is a single ENQ character (ascii code 5). This character is commonly used by hosts and some intermediate networks to request an answerback message or terminal id from your terminal. If you have CIS B+ auto-start enabled all the time then this innocent, and perfectly normal request from the host would invoke a CIS B+ transfer! For this reason we recommend that you leave CIS B+ auto-start disabled until after you are connected to the Compuserve host, ie. after entering your ID and password. This is most conveniently done using a login script and the SetCISB() script command, but can also be done using this dialog. A sample script is shipped with Odyssey, showing how to use the **SetCISB()** command - see **CISV32.SCR**.

Interrogate Response: A Compuserve host sends out a sequence called the "Interrogation Sequence" when a user logs on. The response tells the host your machine type, screen dimensions and so forth. However, our tests have shown that it also causes the host to make other assumptions about the terminal, such as that it can be controlled by VT52 control sequences, which is certainly possible in Odyssey but is not always desirable if you want to be able to produce a pure ASCII log file. You may disable this option, and Odyssey will not respond to the interrogation sequence.

Send-ahead enable: If enabled, the remote Compuserve host can send at full speed without waiting for Odyssey to write data to disk. If the disk drives on your PC are particularly slow then it would be best to disable this option.

Escape control codes: If it is required that all control codes be escaped, for example when using a network which is not completely transparent to control codes, then enable this option. Notice that the Compuserve host normally escapes the most important control characters anyway, so enabling this option is only for the ultra-cautious, or for those who have a specific problem.

ASCII options...

This panel allows you to change some of the parameters which control ASCII uploads, and also affects transmission of macros, the script language "Transmit" command, and so forth.

Inter-Character delay (ms): The number entered in this field represents the delay in milliseconds between characters when uploading ASCII text, pasting messages, or transmitting a macro. The delay may be from 0 to 9999 milliseconds, the default is 20 ms. Some systems are upset if characters are sent too fast, mainly because the priority of their keyboard task has been chosen with average typing speeds in mind, far less than the 240 cps to 3840 cps your modem might deliver. If you encounter this problem then you can change the effective transmission speed of ASCII characters by changing this inter-character delay.

This setting does not affect protocol file transfers other than ASCII upload. Also note the script **Paste()** function is unaffected by this setting, although it *is* affected by the line delay setting which follows.

Inter-Line delay (ms): This can be changed to any number from 0-9999 (the default is 40), and represents the delay in milliseconds between text lines, when uploading ASCII text or pasting a message. Sending lines too fast can upset some systems, so if you think this is happening you should adjust the line delay accordingly.

Expand blank lines: When performing an ASCII upload, it is sometimes necessary to avoid sending blank lines, since these may be interpreted by the remote host as an instruction that the message has ended. Odyssey can avoid this problem by replacing blank lines during an ASCII upload with a line containing a single space character. If you want blank lines replaced in this way then enable this option. It is enabled by default.

Click on **OK** to confirm changes to this dialog, and any other dialog pages you have visited. Click on **Cancel** to cancel changes to all dialog pages you have visited. Click the **Save** button if you wish to make your changes last beyond the current Odyssey session. Clicking the **Help** button displays this help topic. Click on one of the tabs on the right side of the dialog to flip to another dialog page. The Pick Modem button is available in all Odyssey configuration dialog pages, and is described separately.

Odyssey Configuration

Setup|Terminal Emulation Dialog

The **Setup|Terminal emulation** dialog is actually just one page of the Odyssey Configuration dialog, which is a multiple page tabbed dialog. The tabs are shown down the right hand side of the dialog - to change pages, just click on a white tab.

This dialog contains many options which affect how Odyssey controls the terminal window, including the choice of terminal emulations and terminal colors.

Strip Parity: This checkbox enables or disables Odyssey's Strip Parity Bit feature.

Auto line-wrap: If Auto Line Wrap is enabled, then Odyssey will insert a carriage return into any received line which is longer than 80 characters. If line wrap is disabled then Odyssey will allow the line to be of any length, however only the first 80 characters are actually displayed.

Local echo: When you type characters at the keyboard, they are not displayed on the screen (echoed) by magic - instead, either the comms software or the remote host has to take responsibility for echoing the keys you type. If you enable this option then you are instructing Odyssey that it must handle the echoing of characters itself. If disabled (the normal case) then you are telling Odyssey that the remote computer will echo the characters it receives. Local Echo can be set either here, or for individual services using the dialing directory. There are two common mistakes which novices make where local echo is concerned. One is to turn local echo on when it is not required (this results in characters appearing ttwwiiccee on the display). The other mistake is of course to leave it off when it is required, which results in no echo at all.

BackSpace key sends: This option tells Odyssey whether it should generate either backspace (ASCII 8) or DEL (ASCII 127) when you press the backspace key. This option applies only when certain terminal emulations (such as VT100) are used, and is ignored elsewhere.

CR-In translation: When text is received from a remote system any incoming carriage return can be treated in one of two ways. If this option is set to ***CR*** then incoming carriage returns are left unchanged. If set to ***CRLF*** then line feed characters are inserted after each incoming carriage return. Use the first setting if incoming text lines are displayed double spaced when they should not be. Use the second setting if incoming text lines overwrite each other.

CR-Out translation: When text is transmitted to a remote system outgoing carriage return characters can be treated in one of two ways. If this option is set to ***CR*** then carriage returns are transmitted unchanged. If set to ***CRLF*** then line feed characters are inserted after each outgoing carriage return. Most remote systems will insert their own linefeed character when you send them a carriage return, in which case you would use the CR only setting. Some (rare) systems however require that you send them a linefeed after every CR, which is when the second setting is used.

Terminal type: This field selects the terminal type which the Odyssey terminal window will emulate. Odyssey terminal emulations are separate files which are loaded into memory when required. The emulation files all have an extension of ".TRM". To choose a new emulation simply choose from the list presented which you click on the listbox "down arrow" icon. Note that no emulations will be displayed if there are no ".TRM" files present in the Odyssey home directory. Note that terminal emulations can also be selected automatically for a particular service by naming the required emulation in the dialing directory entry for that service. Emulations can also be selected using the **Emulate()** script command.

Terminal Colors: This pushbutton leads to a subsidiary dialog (see Terminal Colors dialog) which can be used to control the colors used by the Odyssey terminal window to represent monochrome attributes. You should see the
description of that dialog for an explanation of this feature.

The remaining three options on this dialog are intended to be used by those who need to improve the performance of the terminal window.

No 'blit' scrolling: The Odyssey terminal window emulates a scrolling text terminal, which sometimes needs to be updated at quite high speeds (eg. in excess of 3840 characters per second). When this volume of data is streamed the terminal is almost constantly scrolling, and performance is thus controlled by how fast your video board can scroll a window. Odyssey can scroll the window in one of two ways :-

Blit scrolling - means that the terminal window is treated like a bitmap, and so scrolling involves doing a bit block transfer. This is very fast if you own a "Windows accelerator" type board which supports BitBlt operations in hardware. However, it can be slow if Windows has to emulate this function in software. This is the default scrolling mode.

No Blit scrolling - Instead of scrolling the window as a bitmap, Odyssey will simply repaint the window. If your video board can display text a lot faster than it can scroll bitmaps, then this would be the option to use. However, it doesn't look as good as bitmap scrolling. You need to check the "**No blit scrolling**" button if you want Odyssey to scroll in this way.

Deferred updates: If your video board is relatively slow at displaying text, then you may want to enable this option. In 'deferred update' mode Odyssey will postpone physically updating the terminal window until it has collected a larger number of operations which it can then paint in a single "refresh" operation, reducing the overhead of painting characters one at a time. However, if you use this option and the "No blit scrolling" option, and you are receiving a constant stream of scrolling text then you may find that large chunks of text are skipped altogether, if it had been erased by later text prior to the refresh event.

Emulate 'Blink' attribute: The Odyssey terminal window emulates a DOS text mode display, including the mapping of bits in a character attribute. One of those attribute bits is treated by text mode hardware as either the "blink" attribute, or the "high intensity background color" attribute, depending on a BIOS setting - some ANSI-BBS hosts may depend on this feature. If you enable this option then Odyssey will emulate the BIOS feature, and will display characters with this attribute as blinking characters, otherwise it will display them as non-blinking characters with high intensity background colors.

Click on **OK** to confirm changes to this dialog, and any other dialog pages you have visited. Click on **Cancel** to cancel changes to all dialog pages you have visited. Click the **Save** button if you wish to make your changes last beyond the current Odyssey session. Clicking the **Help** button displays this help topic. Click on one of the tabs on the right side of the dialog to flip to another dialog page. The Pick Modem button is available in all Odyssey configuration dialog pages, and is described separately.

Odyssey Configuration

Setup|FAX Dialog

The **Setup|Fax** dialog is actually just one page of the Odyssey Configuration dialog, which is a multiple page tabbed dialog. The tabs are shown down the right hand side of the dialog - to change pages, just click on a white tab.

This dialog is used to configure options supported by the Odyssey Fax Server module.

Path and ID

Path for FAX files: The directory is the place where received Fax documents will be stored, and is also used for temporary files such as converted Fax documents during transmission. If you don't specify a path for Fax files then the Odyssey home directory will be used, which may make the Fax files rather hard to manage (they can be rather large, and will fill up your hard disk quite quickly). We recommend that you create a directory called "FAXRECV" in the Odyssey directory, then specify the full path to that directory using this dialog field. The Odyssey INSTALL program will normally have created this directory for you.

Local station ID: Every Fax station must have an ID - it allows one FAX station to identify itself to another when making or receiving a call (this is the ident you often see on the LCD display of a dedicated FAX machine while it is sending or transmitting a document).   The ID can be up to 20 characters long. According to the FAX standard, this should consist solely of the digits '0'-'9', or spaces, however we have found that all FAX machines we have tried calling so far are perfectly happy with the letters 'A'-'Z' also, so you can try entering your company name, provided it fits in 20 characters and doesn't require anything other than ASCII letters, digits or a space (it is ok to use lower case letters, because the FAX server will automatically force them to upper case before transmitting the ID). If in doubt, stick to your international phone number.

General options

Enable receive: If you would like to be able to receive Fax documents, then enable this option, and Odyssey will answer incoming Fax calls whenever the Fax server is active. Note that you should set the FAX receive directory first. Bear in mind that if you share your modem line with a phone, then it isn't really a good idea to enable this option all the time, since voice callers are going to be somewhat surprised by the Fax tones which greet them.

Track ring indicator (RI) signal: If you enable the Odyssey FAX receive feature then Odyssey must be able to recognise when a call needs to be answered (note that your FAX "auto-answer" feature applies to incoming data mode calls, *not* FAX calls). Odyssey has two ways of recognising that the phone is ringing - one is to watch for "RING" messages coming from a Hayes compatible modem, and the other is to monitor the ring indicator (RI) signal on the serial port. If you enable this option then the latter method is used. The first method may be unreliable due to bugs in the modem firmware, however, the latter method requires that your modem cable connects the RI signal to the PC.

Operating mode: Odyssey supports EIA Class I and Class II Fax modems. Normally Odyssey will check your modem and decide for itself which controlling protocol is appropriate - Class I for class I-only modems, Class II for class II modems and for modems which support both interface types. However, you may prefer to force Odyssey to use one of the other protocols, eg. because your modem supports both protocols and there is a modem firmware bug involved in the protocol which Odyssey would normally use. Note that if you select a protocol which your modem does not support then Odyssey will ignore you, and operate as if you had left this setting left on *Auto*.

Viewer resolution: Standard FAX resolution is 200 dpi horizontally by 100 dpi vertically. If you select

"Detail" on your Fax machine then that resolution is changed to 200 dpi both horizontally and vertically. However, Odyssey normally displays either type of FAX on screen at 100 dpi in both dimensions, since this allows you to see more of the Fax page at once; you would only get the full resolution when the page is printed. You can use this field to force Odyssey to display the Fax at 200 dpi horizontally and vertically.

Class 1 options...

Reverse Tx bit order and **Reverse Rx bit order** reverse the order of bits in FAX bitmap bytes transmitted and received from the modem. For an explanation of these options please read Fax Bit Order Options. Do not modify these options unless you know what they do.

Class 2 options...

Reverse Tx bit order and **Reverse Rx bit order** reverse the order of bits in FAX bitmap bytes transmitted and received from the modem. For an explanation of these options please read Fax Bit Order Options. Do not modify these options unless you know what they do.

Rx trigger: In class II FAX modems, the receiver sends a flow-on signal to the modem, which tells it that the PC is ready to begin receiving streamed image data. Unfortunately, different modem manufacturers have implemented code according to different drafts of the proposed class II standards, and use different characters for this flow-on signal. Most class II modems use DC2, but a few are known to require DC1. You should not change this setting unless your modem manufacturer documents which signal is needed, or unless you already know which signal is needed.

Click on **OK** to confirm changes to this dialog, and any other dialog pages you have visited. Click on **Cancel** to cancel changes to all dialog pages you have visited. Click the **Save** button if you wish to make your changes last beyond the current Odyssey session. Clicking the **Help** button displays this help topic. Click on one of the tabs on the right side of the dialog to flip to another dialog page. The Pick Modem button is available in all Odyssey configuration dialog pages, and is described separately.

Odyssey Configuration

Setup|Host Mode Dialog

The **Setup|Host mode** dialog is actually just one page of the Odyssey Configuration dialog, which is a multiple page tabbed dialog. The tabs are shown down the right hand side of the dialog - to change pages, just click on a white tab.

This setup section is used to tailor the Host Mode feature of Odyssey, which is necessary if you wish to have Odyssey answer calls as well as make them.

Normal user password: The password to be used by normal (ie. unprivileged callers). A string of up to 20 characters can be entered. This level of protection allows users who know the password to gain access to files and information stored in the default host mode directory, but does not allow access to any other directory.

Privileged user password: The password to be used by privileged callers to your computer system. Again, a string of up to 20 characters is expected. Privileged users are allowed to change directory, in other words they have access to any file held on your system. You should therefore be extremely careful to whom you grant privileged access, since even a well intentioned caller might damage vital system files on your hard disk. Normally you yourself would be the only privileged caller.

Welcome message: This string is one of the first messages that a remote user calling the system will see. The message may be up to 65 characters long.

Default directory for callers: This field holds the name of the host mode default directory. Normal callers will have access to this directory only. If no directory is specified then the Odyssey directory is used instead.

Enable Error correction for incoming calls: The fields in this panel control whether or not to enable error correction for callers to the host mode, and if so, whether that means error correction only, or both error correction and data compression. The options in this panel are relevant whether the error correction is done by the modem, or by Odyssey with its internal software MNP feature. If error correction is enabled then the WaitForCall() script command will enable the appropriate level of error correction for callers. The panel contains three radio buttons, one of which may be selected. The radio buttons are:-

- *Disabled* - Error correction will be be disabled when Odyssey enters host mode.
- *Enabled, but data compression disabled* - callers to the Odyssey host will be permitted to use error correction, but data compression will be disabled.
- *Enable error correction and data compression* - callers to the Odyssey host will be permitted to use both error correction and data compression.

Click on **OK** to confirm changes to this dialog, and any other dialog pages you have visited. Click on **Cancel** to cancel changes to all dialog pages you have visited. Click the **Save** button if you wish to make your changes last beyond the current Odyssey session. Clicking the **Help** button displays this help topic. Click on one of the tabs on the right side of the dialog to flip to another dialog page. The Pick Modem button is available in all Odyssey configuration dialog pages, and is described separately.

Odyssey Configuration

Setup|Keyboard Macros Dialog

The **Setup|Keyboard macros** dialog is actually just one page of the Odyssey Configuration dialog, which is a multiple page tabbed dialog. The tabs are shown down the right hand side of the dialog - to change pages, just click on a white tab.

A ***macro*** is a stored string of characters which can be transmitted to the remote system with a single command. They are a shorthand way of sending commonly used text strings. Transmitting a macro should have exactly the same effect as typing each character individually from the keyboard except for the interpretation of special characters within macros which is described below.

Odyssey allows up to ten macros to be stored for later use. You must however remember to save your setup if you want stored macros to be remembered between sessions.

Not all characters within a macro will be transmitted literally, some have special meaning:-

|	means send an End Of Line (EOL) sequence.
~	means "pause here for half a second".
^x	With a letter this instructs Odyssey to send a control character, eg. ^H sends Ctrl+H or ASCII 8. With any other char the literal character is sent - use this to send special characters, eg. ^^, ^|, ^~.
^nnn	nnn is a decimal ASCII code for a character to be transmitted. Always use exactly three digits. For example, ^255 sends CHR(255), ^008 sends a backspace.

Use the Save button to make any changes permanent.

Once the macro has been created, you can then transmit it by typing **ALT+*x***, where x is the number of the macro. For example, **ALT+1** would transmit macro one.

If you need to attach a macro to a specific key, or need more than ten macros, then you should look at the keyboard definition feature of Odyssey (see the discussion of the Keyboard Remapping feature). The macro feature just described is intended for simpler applications.

Click on **OK** to confirm changes to this dialog, and any other dialog pages you have visited. Click on **Cancel** to cancel changes to all dialog pages you have visited. Click the **Save** button if you wish to make your changes last beyond the current Odyssey session. Clicking the **Help** button displays this help topic. Click on one of the tabs on the right side of the dialog to flip to another dialog page. The Pick Modem button is available in all Odyssey configuration dialog pages, and is described separately.

Odyssey Configuration

Setup|Printer Dialog

The **Setup|Printer** dialog is actually just one page of the Odyssey Configuration dialog, which is a multiple page tabbed dialog. The tabs are shown down the right hand side of the dialog - to change pages, just click on a white tab.

The printer selection dialog allows the user to nominate the printer to be used by Odyssey, and also provides access to the printer setup dialog provided by the printer device driver.

Printer selection listbox: Pull down the listbox and select the appropriate printer from the list presented. The printers on the list are those defined in your WINDOWS.INI file.

Convert screen dumps to monochome: When you ask Odyssey to print the contents of the terminal screen, Odyssey can do a color print (leaving it to your printer device driver to translate colors if and as necessary), or Odyssey can convert the screen display to monochome itself before passing it to the printer. Odyssey knows what should be readable inside this "terminal display bitmap", and so can normally do a better job of monochrome conversion - if that is necessary. Enable this option if you want Odyssey to do the conversion as described.

Click on **OK** to confirm changes to this dialog, and any other dialog pages you have visited. Click on **Cancel** to cancel changes to all dialog pages you have visited. Click the **Save** button if you wish to make your changes last beyond the current Odyssey session. Clicking the **Help** button displays this help topic. Click on one of the tabs on the right side of the dialog to flip to another dialog page. The Pick Modem button is available in all Odyssey configuration dialog pages, and is described separately.

Odyssey Dialogs

Terminal colors

One of the main tasks Odyssey has to perform is terminal emulation, ie. emulating the features of particular text terminals. Most such terminals display monochrome text, and allow attribute effects to be applied to text, such as *underlining*, *reverse video*, *bold* and *blinking* characters.

The original version of Odyssey was developed for DOS color text systems, in which such monochrome attributes were not supported. Therefore, the DOS version of Odyssey mapped every possible combination of the four attributes mentioned onto a different displayable color, and allowed the user to edit that color mapping. So, when the host asked the terminal to display "bold" text, it might have appeared on an Odyssey VT100 display as yellow on a blue background - and so on.

However, Windows™ has no such limitations; we could display bold and underlined text if we wanted to. On the other hand, what was originally designed as a slightly dubious workaround has now become a feature - many users have taken advantage of the DOS Odyssey behaviour to provide themselves with colorful alternatives to the boring VT100 monochrome display they would otherwise be using when connected to a local mini. So, even though it is no longer mandated by the hardware, the Windows version of Odyssey still maps monochrome attribute combinations onto colors, and this dialog exists to allow you to edit that color mapping.

The **Terminal Colors dialog** has three main elements.

- The **Element** listbox on the left displays the list of the sixteen possible combinations of the four recognised monochrome text attributes. You can select any element of this list by clicking on it.

- The **Colors** panel on the right displays the complete range of possible color attributes to which an element may be mapped. A box is drawn around the color which is mapped to the currently selected **Element**. To change the color mapping for the selected element, simply click on a different color combination in the Colors panel.

- The '**Sample Text**' panel near the bottom of the dialog shows how a larger piece of text would look with that foreground/background color combination.

Click on **OK** to confirm changes in this dialog, or click on **Cancel** to abandon any changes you have already made. Clicking the **Help** button displays this help topic.

Odyssey Dialogs

Pick Modem

The **Pick Modem** dialog appears when you click on the **Pick Modem** button in any of the Odyssey configuration dialog pages.

When you first installed Odyssey you were given a pick list of modems, and allowed to select the entry corresponding to your particular modem brand. This allowed Odyssey to automatically configure itself with the correct strings which control your modem.

Which is fine of course, until the day you come home with your shiny new high speed V62ter/ISDN modem, and find that Odyssey is now configured with precisely the *wrong* strings for it.

This isn't really a problem. Just access this dialog and you will be given that list of modems again. Choose the entry which corresponds to your new modem. It is likely that this list will become rather dated over time, so look out for more up to date lists on your favorite BBS (the Odyssey file containing the list is an editable text file called ODYSSEY.MDM), or keep in touch with your Odyssey dealer for updates.

Click on **OK** to accept the modem you have selected, or click on **Cancel** to retain your existing configuration. Clicking the **Help** button displays this help topic. If you click on OK then many changes will be made to fields throughout the Odyssey configuration dialog pages.

Odyssey Help

What is a Tabbed Dialog?

Odyssey has very many configuration options for you to manipulate - *far* too many options to fit on one simple dialog.

The traditional way for a Windows application to behave, when it needs lots of data entered that won't fit on one dialog, is to have a parent menu or dialog, and lots of nested dialogs. That sort of system is extremely cumbersome to use, since it involves traversing up and down the dialog/menu hierarchy, passing through dialogs you don't want to see, in order to get to the dialog containing the variable you want to change. Even the sheer number of dialog windows appearing on the screen at once can lead to confusion.

Odyssey adopts a strategy to replace this long winded interface which, although not yet traditional, is becoming increasingly common. The tabbed dialog can be seen in many popular windows apps, including the latest versions of Microsoft Word and Excel, and will be a standard feature of Window 95 (alias Chicago).

The idea is for the parent dialog to have "tabs", like the thumb tabs in a personal organiser or in a dictionary. To flip to a different page in the dialog you simply click on the appropriate tab. You can flip between tabbed pages entirely randomly; eg. if you want to flip from the first tab to the third you just click on the third tab - no need to visit the second tab at all, unless you particularly want to.

The Odyssey configuration dialog has its tab buttons down the right hand side. The gray tab indicates the current page, just click on a white tab to move to a different page.

Odyssey Help

How Odyssey constructs the Dial Command

When you tell Odyssey to dial a number, Odyssey takes the **dial prefix**, appends the phone number from the selected dialing directory entry, appends the **dial suffix**, checks for and expands any number prefix codes, then sends the resulting dial command to the modem. For example:-

Dial Prefix	=	**ATDT**	(select tone dialing)
Number	=	**012-345-6789**	
Dial Suffix	=	**|**	(Ody symbol for newline)

so the resulting dial command is:-

ATDT012-345-6789|

which should cause a Hayes compatible modem to dial that number.

☞ Those using Odyssey to dial out through a private telephone exchange normally have to prefix every telephone number with a digit such as '9', which selects an outside line. Rather than inserting this digit in front of every number in the dialing directory, you can put it in the dial prefix, eg. "**ATDT9,**". The comma on the end tells a Hayes compatible modem to pause for half a second, which gives a typical private exchange time to select the outside line.

Baud Rate Detection

When Odyssey sees the connect string from the modem (eg "CONNECT 1200"), it recognises that a connection has been made, however it does not stop reading characters from the modem until it gets a carriage return. While it does this, Odyssey is actually looking for the connect rate (eg. "1200", as in the example shown).

If Odyssey knows the rate the modem actually connected at, as opposed to the rate at which Odyssey is currently communicating with the modem, then it is possible for Odyssey to adapt to the actual connect rate on each call, provided you enable the "*Baud rate detection*" option in the Setup|General dialog. The purpose of this feature is to avoid any flow control problems which could arise if the terminal baud rate is different from the modem baud rate. However, not all modems are happy to have the speed of the terminal switch suddenly, *after* a connection has been established. The best way to find out is to try it. Some modems do not support this feature at all, while others support it as an option which must be enabled via an init string or by via switches on the modem itself.

☞ Most modern high speed modems these days are designed to work best with a fixed data rate to the terminal - if your modem supports hardware error correction and/or data compression then this is almost certainly the case and you should disable baud rate detection, both in Odyssey, and in the modem.

Odyssey Help

Odyssey and Error Correction

This help topic is here to explain why Odyssey needs error correction control strings in its Setup|Modem Init Strings dialog.

The **Setup|Modem Init String dialog** allows you to define the strings used to control hardware error correction in your modem. If your modem does not support any error correction then these strings should be left blank, and software MNP should be enabled.

Some users who have evaluated Odyssey without the manual have been puzzled by these options - they don't understand why it exists, particularly since other packages don't seem to have it; so a more in-depth explanation seems to be required...

The reason these options exist in Odyssey is partly historical: Odyssey was one of the first comms packages to implement **MNP** error correction in software. Being implemented in software gave the user complete control over error correction, such as the ability to enable or disable error correction for individual services, or to control the *level* of error correction/data compression to use with particular services, or even with individual calls. This amount of control was, and is, *very* convenient for most users, once they had discovered it was possible.

Then the market changed, and modems which offered hardware error correction and data compression began to get *much* cheaper. Users could get a 20% speed advantage by moving to hardware error correction, but on the other hand they lost the level of control they had got used to when error correction was software only.

Naturally, Odyssey users wanted the best of both worlds - the performance of hardware error correction with all the control afforded by error correction in software. However, to do this, Odyssey has to know what strings to send to the modem to enable and disable error correction, or to select error correction with or without data compression. Unfortunately, the de-facto standard Hayes modem control language does not extend to error correction, so Odyssey must be told the correct strings for each modem. A modem pick list feature was added to Odyssey so that the correct strings could be supplied for most popular modems, but if your modem was not on that list, and you would like the control described, then you will need understand what Odyssey requires, examine your modem manual, and provide the appropriate command strings.

The dialing directory was also changed at the same time. There used to be a "Software MNP" field in the dialing directory, which you ignored if you used hardware error correction. Now that field has become the "Error Correction" field, which you should set appropriately regardless of who handles the error correction. When it comes time to establish a connection, and error correction is requested, the Odyssey dialer will look at the configuration settings here to see whether software MNP is to be used, or whether hardware error correction is available and preferred.

Odyssey Help

The Strip Parity Bit Feature

When a particular remote service is dialed directly, it is normally easy to find out what parity setting that service requires. However, sometimes a remote service is accessed *indirectly* using a packet switching or other low cost network (such as Tymnet or DialPlus), and this network may require a different parity setting than the service itself. For example it is common for these networks to require a seven bit even parity setting, while most host services work best with an eight bit no parity setting.

You could easily configure Odyssey for seven bits even parity using the Setup|Comms dialog or the dialing directory, and this would work perfectly well - except for file transfer, since most file transfer protocols expect a full eight bits per byte of data. On the other hand, if you configure Odyssey for eight bits no parity then you will see spurious semi-graphic characters on the terminal screen while connected to the network, because some of the incoming characters will have their eighth (parity) bit set.

You can resolve this dilemma on many systems by setting Odyssey to eight bits no parity so that file transfer can work, and also enabling the Strip Parity option in the Setup|Terminal dialog, which ensures that you do not see the funny graphic characters. Alternatively you can enable or disable parity bit stripping for individual services using the dialing directory.

Note that when this option is enabled, Odyssey will only strip parity bits from characters displayed on the terminal screen, so this will *not* upset file transfer.

The only drawback to using this feature is that deliberate use of IBM semigraphic characters is also prevented (characters with codes greater than 127). This mostly means that you cannot view certain BBS logo displays in all their glory, which is not usually a damaging limitation. In any case, these fancy logins tend to occur on small bulletin board systems, where the packet switching network problem is not an issue.

The Odyssey ANSI emulation works best when parity bit stripping is *not* enabled, since this emulation is actually designed to make use of the full eight bit PC (OEM) character set. If you find this to be a problem then you will also likely find that you have no need of ANSI emulation with that service anyway. In that case you should either switch to VT100 emulation, or even better use TTY emulation, since there is little point in having Odyssey emulate a terminal whose features are not required.

Odyssey Help

Fax Bit Order Options

You will notice that many of the options in the Setup|Fax dialog have to do with "bit order": The EIA Class I FAX specification fails to specify whether data bytes transferred from PC to modem are transmitted to the remote modem least significant or most significant bit first - and since the FAX protocol (CCITT T.30) is bit oriented this is a rather unfortunate omission. We expect most manufacturers will assume the norm for asynchronous modems, which is that least significant bits are transmitted first. However, since it is unspecified, the modem manufacturer would be quite entitled to assume the opposite case. The class I bit order options allow you to adapt the FAX server for your modem - however the defaults should be correct in most cases.

Bit order also rears its head in class II, but for a different reason. We know of at least one version of the Rockwell FAX modem chipset which uses the wrong bit order in class II receive mode - the opposite bit order to that documented by the Rockwell data sheet and which is used in class II send mode. This problem can again be fixed by setting the appopriate bit order option to "Reversed".

DO *NOT* CHANGE THE BIT ORDER OPTIONS UNLESS YOU KNOW *EXACTLY* WHAT YOU ARE DOING. These options affect *only* what a FAX looks like when it is viewed - it has no affect whatever on the FAX send/receive protocol, and so is *not* a general panacea with which to fiddle whenever you have problems receiving or sending a FAX.

Odyssey Help

Using Odyssey Windows

Select one of the user guide topics listed below:

Using Odyssey

Odyssey and the Windows MDI interface

Odyssey is a Windows ***Multiple Document Interface*** (MDI) application. Instead of displaying all kinds of things at different times in one window, it displays different things in different windows, and you can have several such windows active at the same time. In order to work with the contents of a particular window you first have to select it (the currently selected - or active - window is easily identified because it is painted using a blue caption, while inactive windows are painted with a white caption). You can select a window by clicking it with the mouse, or you can select the window by choosing it from the list given in the Window menu.

Odyssey supports the Windows MDI keyboard conventions for switching between windows (eg. Ctrl+F6), and also implements its own conventions inherited from Odyssey for DOS, so that users moving to Odyssey from the DOS version can continue to use the keys they are familiar with. The following keys may be used in any Odyssey MDI child window :-

F5	- Maximizes (zooms) the currently active window, or restores the window if it was already maximized.
F6	- Switches to the next window (same as Ctrl-F6).
Alt+F3	- Closes the current window (same as Ctrl+F4).

Although this type of program interface is called MDI (where D means Document), Odyssey does not use child windows solely for displaying documents, at least not in any conventional sense of the word. Instead, Odyssey is divided into multiple subsystems, each of which performs a separate but useful function, and each of which uses a specialised MDI child window to display information relevant to that function.

The following is a brief overview of the different types (classes) of document window which Odyssey currently uses. Later sections of this user guide give more detailed information about the commands and features of specific window types. :-

Terminal window	- Unlike some other packages which dedicate the entire main application window, Odyssey directs all terminal emulation output to a child window, not to the full screen or main application window. This allows you to move the "terminal" display around the screen, just like any other window. You can of course maximize the terminal window if you actually want it to occupy the whole screen. There is only ever one terminal window in existance, and it cannot be closed (though it *can* be minimized). The terminal window is created automatically when Odyssey starts up. The position and minimized/maximized/normal state of the terminal window is saved between Odyssey sessions.
Text editor window	- You can have several text editing windows open at any time - open a text editor window by choosing **File|Open** or by clicking the edit toolbar button. The text editor in Odyssey can handle text files of any size (ie. there is no 64k limit). Cutting and pasting of text between editors is permitted, as is cutting and pasting text to and from other applications via the Windows clipboard. The text editor may be configured to use either CUA or "native" Wordstar™ compatible commands (the latter for the benefit of former users of Odyssey for DOS). The editor supports normal editing operations, block commands, search and replace etc, plus of course Windows style text selection using the mouse or shift-<movement key>.
Directory viewer	- A window of this type looks very much like a directory view in the standard Windows *File Manager* application. A directory tree appears on the left, a list of files on the right, and a row of drive buttons at the top. Why not just use file

manager you may ask?. Well, the Odyssey directory view exists in order to allow you to perform Odyssey specific functions on the list of selected files - such as view selected text files with the Odyssey text editor, view GIF™ or JPEG files with the Odyssey bitmap viewer, or upload a batch of selected files using Odyssey Zmodem file transfer. While all of this could have been implemented by making Odyssey a drag and drop client (in fact, we allow that too!), it turns out to be more convenient to have the feature built directly into Odyssey, at least in our opinion.

Bitmap viewer - If you double-click on a GIF,BMP or JPEG file while using the directory viewer, Odyssey loads the image and displays it using a bitmap viewer window. This window allows you to print the bitmap, copy it to the clipboard, and so on. Odyssey also displays a GIF or JPEG in a bitmap viewer if you are downloading the GIF/JPEG file from a BBS, and you have enabled the "*View GIFs/JPEGs while downloading*" option in Setup|File transfer. The Odyssey FAX server also uses a bitmap window to preview FAX images you are about to send, and to display FAX files you have received. (Note: We plan to add the ability to view other bitmap formats, such as PCX and TIFF, using our bitmap viewer. This may have been added by the time you read this - please check the README.TXT file in the Odyssey directory to make sure).

FAX server window - The FAX server window is used when the FAX server is active - it displays a list of FAXes in your FAX-receive directory. You can view or delete a FAX in the list by clicking on it, and then selecting the appropriate menu option.

File transfer progress - A file transfer progress window looks like a dialog box, and is only displayed during a file transfer, to show you the progress of that file transfer. The fact that it *isn't* an ordinary dialog is important, since it means that you can switch to other child windows while a file transfer is in progress (eg. to a text editor or directory viewer). If it was a normal modal dialog you would be frozen out of other windows until the transfer was completed.

Using Odyssey

The Terminal window

The ***Terminal window*** is where all characters received from the serial port are displayed. It is in this window that terminal emulation takes place, and it is this window which must be active if you intend characters that you type to be transmitted through the serial port.

The terminal window is unusual in a number of ways .

First, no matter what size you make the terminal window, it always contains the same number of rows and columns. If you drag the window to a new size (or maximize it), the visible contents of the window do not change, because Odyssey has adjusted the font size in order to keep the number of rows and columns constant. Obviously, this is done because all the terminals being emulated also have a constant number of rows and columns.

Second, Odyssey does not allow you to make the terminal window any old size you like. This is because of the font sizing mentioned in the previous paragraph. When you change the terminal window size Odyssey changes the font size to keep the rows/columns the same. However, a font of precisely the right size is not always available, so Odyssey chooses the closest sized font available, calculates the dimensions of the window, assuming 80 columns by 25 rows with the new font, and then "snaps" the window to that new size.

In fact, Odyssey does a little more than that. If it simply snapped to the closest font size, you may have found that it snapped back to the terminal size you were using previously, which would have been rather annoying. Instead, Odyssey looks at what you tried to do, and then tries its best to satify you. If you made the terminal window a *little* larger, then Odyssey looks for a font which is a little larger than the font it is already using, and snaps the window to the appropriate size for that font. Likewise, if you made the terminal window a *little* smaller then Odyssey looks for a slightly smaller font and changes the window size to suit. If you make the window a *lot* larger or smaller, then Odyssey chooses the largest available font, or the smallest (as appropriate), and again changes the terminal window size. Maximizing the terminal window always causes the terminal window to use the largest sized font it has available, of those which are suitable for your display resolution.

The final special feature of the terminal window is again related to the font sizing problem. Odyssey has very specific requirements for the font which it uses in the terminal window. First, it must be a fixed width font (all the text terminals Odyssey emulates use fixed width fonts). Second, the font must use the OEM (IBM PC text mode) character set, mostly because ANSI emulation requires the OEM character set. In PRESTEL emulation mode the font is even stranger - it must use the PRESTEL character set, including those funny mosaic characters, which it certainly isn't going to find in any standard Windows font, even a Windows OEM font.

Furthermore, Odyssey requires that the font used must be available in a nice range of sizes, otherwise the font sizing feature described above will not work very well.

Odyssey cannot depend on a standard Windows installation satisfying these specialised font requirements, which means that we had to bundle a suitable font selection with Odyssey. These fonts are stored in the file ODYPC8.FON (PRESTERM.FON when PRESTEL emulation is in use), and these font files must reside in your Odyssey directory. Note that the Odyssey terminal window *always* uses one of these specialised fonts - you can override the default font selection in other Odyssey windows, but not in the terminal window.

The Odyssey terminal window is used to emulate a scrolling text terminal, thus its performance is heavily dependant on the scrolling performance of your video display card. A "Windows accelerator" type card is very much recommended for this application. However, Odyssey provides a number of options in Setup| Terminal emulation which may be used to improve performance, if necessary. The same configuration dialog can be used to control other terminal window parameters, such as the colors used when mapping monochrome video attributes to the color display.

Notes: The **Review editor** and **text logging** features record text which has been displayed in the terminal window. Also, if an Odyssey script uses the **Write()** command, that output goes to the terminal window. This means that scripts can send configuration control sequences to terminal emulations via the Write() command, and also supplement text capture files etc.

Using Odyssey

The Dialing Directory

The Dialing Directory is a central repository where all the details are kept about host services you might want to call - what modem speed to use, what parity,terminal type, what logon script (if any) to run, whether to turn on text logging - and so on. The dialing directory is also your starting point when you want to make a call. Having entered these details about a service you can make a call by simply highlighting that services entry, and clicking the **Dial** button (or double-clicking the entry).

Now, there are some macho types around who don't see the point of a dialing directory. A user can get the modem to dial manually any time he likes by typing a simple **AT**<*phone number*> command, so why doesn't he? Why do we need some complex dialog to take place before we can dial a number?

The best way to understand clearly why the Dialing Directory exists, is to imagine how Odyssey would be used if the dialing directory *didn't* exist! (in fact, the very first DOS version of Odyssey didn't have a dialing directory, and was used in precisely the following way).

How do you make a call when you don't have a dialing directory? Well, as mentioned, the usual way is to simply type a modem dial command directly at the terminal keyboard, for example on a Hayes compatible modem you would type **ATDT12345678**<enter>. Your modem will dial, make a connection, and suddenly the terminal screen will be filled with scrolling text.

So far so good...

...except that you should have set the comm port speed to 19200 bps before the call, and you also forgot to change the comm port settings to eight bits no parity (it was set to seven bits even parity after the previous call). Also, didn't you intend to turn text logging on for this service? - and what about terminal emulation - doesn't this service expect you to be using VT100? - and anyway, are you *sure* you dialed the right number?

It really is a lot of hassle to remember all this stuff, and a real pain having to go through this checklist manually before we can finally make a call - so why don't we write a script to take care of it? (this was the recommended procedure in that very first version of Odyssey). Well, writing that script will certainly do the trick, *if* you are confident enough in your programming skills with the script language - a big "if" for most novice users.

So now you see; the dialing directory is nothing more or less than a handy place to jot down information on how to set up Odyssey correctly for a bunch of different host services - a place where that information won't get lost. And the real bonus is that when it comes time to call the service, Odyssey does all the work for you. As soon as you select a service from the directory, Ody looks at the settings in the dialing directory entry, dials the number, *and* automatically makes the appropriate changes to the setup menu - and it does so at the best moment for each change (some before the number is dialed, some after the connection is made).

So, you should not be confused by the fact that the options in a dialing directory entry often appear to duplicate similar looking items in the setup menu. There is only one "Baud Rate" setting in Odyssey - the one in the setup menu. The "Baud rate" field in the dialing directory entry for a service simply says what the setup menu baud rate should be changed to when you ask Odyssey to call that service. The same goes for all the other settings you find in a dialing directory entry.

See also, the Dialing Directory Dialog.

Using Odyssey

The Text Editor

See also: Text Editing Commands

Odyssey provides a powerful built in text editor which is *not* based on the standard windows edit control. The main differences between the Odyssey editor and an edit control are:

- The Odyssey editor has no 64k limit on the size of a file which can be edited. Thus, you can use the Odyssey editor to examined a captured text log file, no matter how large that file is.

- The Odyssey editor gives you the choice of whether you want to use CUA editing commands, or "native" Wordstar™ compatible commands, which might be preferred by those moving to Windows Odyssey from the DOS version.

You should also find the Odyssey editor to be much faster than the standard edit control, especially where text searches are concerned, and when processing larger files.

Although we have made every effort to make text editing with the Odyssey editors familiar to those used to normal windows controls, there are inevitably some minor differences. These differences are deliberate, where we have decided that the standard operation represents a serious design flaw:-

- In a standard Windows edit control, if you use a scroll bar to skim over the text, the caret does not move to remain within the visible region of the file. Thus, if you then decide to do some editing at the point you've skimmed to, and you start by touching any of the arrow keys, then the edit control will jump back to where the caret was, losing your place in the file where you had scrolled to. Odyssey does not follow this example - like any normal text editor, the caret in the Odyssey editor moves to remain in view as you scroll, no matter how you scroll (ie. regardless of whether you use the arrow keys or scroll bars).

- In a standard Windows edit control, if you have marked a section of text and then move the cursor, the selection is lost. Odyssey does not follow this convention. A block of text stays selected until you tell the Odyssey editor to remove the selection.

- In a standard Windows edit control, if you have marked a section of text and then type any character, the selected text is deleted and replaced with the character you typed. This can come as rather a shock to someone who had selected the block in order to print it, and then noticed a minor typo to correct first. Needless to say, Odyssey does not follow the standard edit control convention. Text is deleted when you explicity enter a block delete command, and not otherwise.

You can have up to ten independant text editing windows opened at any time, plus one special window (the Review Editor window) which Odyssey uses, when so commanded, to display text which has scrolled off the terminal display. Each window can be sized independantly, dragged to new positions etc. Normal edit windows may be used to prepare text messages offline, or for editing script source files, or for viewing any text file you like.

Default modes for all editors can be controlled using the Setup|Editor configuration section. This allows you to set your preferred defaults for tab size, insert or overwrite mode, right margin position, and so on.

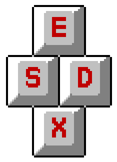
All edit windows are normally configured to use CUA compatible editing commands, supplemented by Wordstar keystrokes for those functions which CUA does not define. However, you can force the editor to use pure WordStar™ commands by setting the appropriate option in the Setup|Editor dialog.

To create text while in an editor you simply press any of the letter or number keys on the keyboard. Any letter you type will appear to be inserted into the text at the cursor position. When the cursor reaches the extreme right hand side of the text window then the text displayed will scroll sideways. Don't panic! Your text is not destroyed, it is merely not visible since it has passed outside the boundary of the text window. In a similar fashion, if you move the cursor down past the last line, or up past the first line in the text window then the text will be scrolled up or down as required to keep the cursor within the boundaries of the display.

Special editor functions are performed using the remaining keys on the keyboard or by means of control key combinations (commands which use a combination of the **Ctrl** key and another letter). The reason for the duplication of several commands is that some people who are touch typists prefer not to move their hands from the main keyboard area, while others prefer a simpler, single keystroke command where possible. The description of commands which follows will use abbreviations such as "**Ctrl+X**" for control commands, which means that you should hold down the **Ctrl** key while you press the X key.

As mentioned above, Odyssey by default uses CUA editing commands. However, the basic CUA command set is extended with a large number of WordStar compatible editing commands, so it will be useful to understand how WordStar style commands are structured (those already familiar with WordStar should skip this section).

WordStar commands are mnemonic because of their relative positions on the keyboard, and seldom because of the letter on the keytop itself. For example, the standard cursor movement commands use the E, S, D and X keys - hardly easy to memorise if you look at the letters, however when you look at how these keys are grouped on a standard PC keyboard:-



Then you may not be surprised to learn that **Ctrl+E** moves the cursor up, **Ctrl+S** moves the cursor left, **Ctrl+D** moves the cursor right, and so on.

NOTE for CUA fans: Don' t worry: these are standard Wordstar keys, supported for the benefit of Wordstar devotees, and provided *in addition* to the obvious choices for moving the cursor around on a PC - the arrow keys). Studying them (even briefly) does however lead to a better understanding of the overall structure of the WordStar editing commands, if you ever want to use them.

Other cursor movement commands can be arrived at by prefixing the appropriate letter with the **Ctrl+Q** (Quick) command. For example, **Ctrl+Q E** (type **Ctrl+Q,** release, press **E**), will move the cursor to the first line in the text window, and **Ctrl+Q X** moves the cursor to the last line. **Ctrl+Q S** moves the cursor to the first position on the current line, while **Ctrl+Q D** moves it to the last position. Other commands group in a similar fashion, and once you recognise these basics, most WordStar commands should be significantly easier to remember.

All edit windows show the name of the file being edited in the window caption. All editors also share a common "status line" at the bottom of the Odyssey MDI desktop area, which gives you some basic information about the file you are editing. For example it tells you the current line and column number of the current cursor position, as well as the status of the insert, indent and word wrap toggles. Insert mode means that letters typed are inserted into the text, with remaining characters on the line moved aside to make room. The opposite of insert mode is overtype mode, in which characters typed replace or "overtype" characters previously at that position.

One other feature of the status line is a "modified" legend which appears when the text has been altered. This tells you that the file will need to be saved before leaving Odyssey if you do not want the changes to be lost. Odyssey will of course remind you of this anyway when you close the edit window or decide to exit the program, if you have not saved the file in the meantime.

Odyssey Text Editor

Editing Commands

Please choose one of topics below for information on commands falling into the selected category.

See also: Text Editor Overview

☞ In the text editor documentation, "cursor" refers to the familiar text cursor (or caret), and not to the mouse pointer.

As text is received from the remote computer it is stored in the review editor buffer, and when the buffer is full the oldest 2000 characters are thrown away. The review buffer therefore represents a "window" into the current Odyssey session, covering the most recently received 30k of text. Since the Review editor is intended primarily for viewing purposes only, it does not feature commands for loading and saving files. You can however mark blocks and write them to disk, a feature which allows you to cut and paste sections of text between editors. You can also print a selected block using the "Print Block" command. All of these commands are described fully, later in this help topic.

Odyssey Text Editor

Movement commands

Commands are provided which allow you to move the cursor by one character position left, right, up or down. You can also move left or right one word at a time, move to the top or bottom of the current window or document, or you can move up or down the document one page at a time. At no time will you be able to move the cursor outside the boundary of the current display window, neither will you be able to move the cursor past the first or last lines in the document. The following paragraphs describe each of the cursor movement commands.

Character Left (Native)	**[Ctrl+S or ←]**
Character Left (CUA)	**[←]**

This command moves the cursor one character position to the left unless the cursor is already in column one, in which case this command will do nothing. The cursor will not wrap around to the preceding line.

Character Right (Native)	**[Ctrl+D or →]**
Character Right (CUA)	**[→]**

This command moves the cursor one character position to the right. If the cursor was already at the extreme right of the display window then the text will scroll left. The cursor does not wrap around to the following line.

Line Down (Native)	**[Ctrl+X or ↓]**
Line Down (CUA)	**[↓]**

Moves the cursor down one line in the current column, scrolling the window as necessary to keep the cursor within the display boundary. This command does nothing when the cursor is already at the last line in the document.

Line Up (Native)	**[Ctrl+E or]**
Line Up (CUA)	**[ ]**

Move the cursor up one line in the current column, scrolling the window as necessary to keep the cursor within the display boundary. This command does nothing when the cursor is already at the first line in the document.

Word Left (Native)	**[Ctrl+A or Ctrl+←]**
Word Left (CUA)	**[Ctrl+←]**

This command moves the cursor to the start of the nearest word to the left. If the cursor was on the first word of a line then the cursor moves to the beginning of the last word on the previous line (in other words the cursor wraps around). This command does nothing if the cursor was already on the first word in the document. As far as the editor is concerned, a "word" is any sequence of letters or digits separated by spaces or punctuation characters.

Word Right	**[Ctrl+F or Ctrl+→]**

This command moves the cursor to the start of the nearest word to the right. If the cursor was on the last word of a line then the cursor moves to the beginning of the first word on the following line. This command does nothing if the cursor was already on the last word in the document.

Left on Line	**[Ctrl+Q S or Home]**

Moves the cursor to the extreme left of the current line (column one), restoring the display window if it had previously been scrolled left.

Right on Line **[Ctrl+Q D or End]**

Moves the cursor to the extreme right of the current line (one character position beyond the last character on the line), for example, if the current line is 10 characters long, then the cursor moves to column 11, ready for new characters to be appended. If the line is longer than the display is wide, then the display window will scroll left as necessary to keep the cursor within the display boundary.

Page Up (Native) **[Ctrl+R or PgUp]**
Page Up (CUA) **[PgUp]**

Moves the cursor up one page in the current document, a page being defined by the current height of the display window. The text forming the previous page is displayed. The cursor position does not move unless there are no previous pages, in which case the cursor moves to column one, line one in the document.

Page Down (Native) **[Ctrl+C or PgDn]**
Page Down (CUA) **[PgDn]**

Moves the cursor down one page in the current document, a page being defined by the current height of the display window. The text forming the next page is displayed. The cursor position does not move unless there are no following pages, in which case the cursor moves to column one of the last line of the document.

First Line in Window (Native) **[Ctrl+Q E or Ctrl+Home]**
First Line in Window (CUA) **[Ctrl+Q E or Ctrl+E]**

Moves the cursor to column one of the first line in the display window.

Last Line in Window (Native) **[Ctrl+Q X or Ctrl-End]**
Last Line in Window (CUA) **[Ctrl+Q X]**

Moves the cursor to column one of the last visible line in the edit window.

First Line in File (Native) **[Ctrl+Q R or Ctrl+PgUp]**
First Line in File (CUA) **[Ctrl+Q R or Ctrl+Home]**

Moves the cursor to the column one of the first line in the document, and causes the text of the first page in the document to be displayed.

Last Line in File (Native) **[Ctrl+Q C or Ctrl+PgDn]**
Last Line in File (CUA) **[Ctrl+Q C or Ctrl+End]**

Moves the cursor to the column one of the last line in the document, and causes the text of the last page in the document to be displayed.

Beginning of Block **[Ctrl+Q B]**

Moves the cursor to the first character of a marked (selected) block of text. If no block is currently marked then this command does nothing. Other block commands are described in a later section.

End of Block **[Ctrl+Q K]**

Moves the cursor to one character position beyond the last character of a marked block. If no block is currently marked then this command does nothing.

Goto Line **[Ctrl+Q G]**

This command allows you to quickly jump to a specific line number within the current document. A dialog prompts you for the line number the editor should jump to.

Odyssey Text Editor

Insert and Delete commands

These commands allow you to control whether typed characters are inserted into the text or overwrite existing text, insert line breaks, and also allow to you delete characters, words, lines or blocks.

NOTE: Characters, lines or blocks once erased cannot be recovered. Be especially careful of holding any control key down for too long as the PC keyboard "auto-repeat" feature may result in you erasing more than you intended.

Insert/Overtype Toggle (Native)	**[Ctrl+V or Ins]**
Insert/Overtype Toggle (CUA)	**[Ins]**

This command is used to toggle the editor between insert and overtype modes. The legend "Insert" will appear on the editor status line when the editor is currently in insert mode, or else this word will be "Overtype" when you are in overtype mode.

Characters typed while the editor is in insert mode are inserted into the document at the current cursor position, with any characters on the line at and to the right of this position being shifted further right to make room. The cursor then advances to the right, ending up on the same character it was on before the insertion.

In Overtype mode the character typed replaces the character previously at that cursor position; the character overtyped is lost. The cursor then advances to the next column.

Insert Line (Native)	**[Ctrl+N]**
Insert Line (CUA)	**(not available)**

This command inserts a line break at the current cursor position, without moving the cursor. Another way to insert a line break is simply to press the <Enter> key, however that operation is different in that the cursor moves to the beginning of the new line.

Insert Tab	**[Ctrl+I or →|]**

This command moves the cursor to the next tab position. The editor supports several tabbing modes (the mode being selected in the Setup|Editor dialog. :-

The editor can use **Smart Tabs** or **Fixed Tabs**. Smart tabbing means that when you press the tab key, the editor examines the line above the current line, and aligns the cursor with the next word on that previous line. This is extremely useful when laying out tables. Fixed tabs are your usual fixed-interval tabs - the default interval being eight columns, though this can be changed in the **Setup|Editor** dialog.

The editor can also use **Hard Tabs** or **Soft Tabs** (note that this option is independant of the Smart or Fixed tabs feature). In Hard Tab mode tab intervals are filled with actual tab characters (ASCII 9), whereas in soft tab mode the tab interval is filled with spaces. Hard tabs make the text file smaller, but may create formatting problems if you import the text into an editor that assumes a different tab interval. The Hard/Soft tab option *only* affects what Odyssey does when you insert a tab in the Odyssey editor - selecting soft tabs does not prevent Odyssey interpreting tab characters correctly when it reads in a foreign ASCII file. Note: the **Setup|Editor** dialog refers to this feature as the "*Tab Fill Character*". If the fill character is Tab then you are using hard tabs, if it is space then you are using soft tabs. The editor options menu contains a checkable "Hard Tabs" item (if this item is not checked then you are using soft tabs).

If this command is used in insert mode then the text at and to the right of the current position will be shifted to the nearest tab stop to the right. In overtype mode the cursor is moved, but the line contents are not affected.

Insert Control Character [Ctrl+P]

If you need to insert a control character into the text (for example to embed a form feed command for the printer), then at first glance you may decide that this is impossible, because when you type **Ctrl+L** (the formfeed character), for example, then the editor interprets this key as an editor command instead of inserting it into the text. To get around this problem you first type **Ctrl+P**, which tells the editor that the next character you type is to be regarded as a character to be inserted, and not as an editor command. Therefore, to insert your formfeed character you would type **Ctrl+P** followed by **Ctrl+L**.

Delete Character [Ctrl+G or Del]

This command deletes the character at the current cursor position. Characters on the line to the right of this position are moved left to close the gap.

Delete Character Left [Ctrl+H or BackSpace]

This command deletes the character to the left of the current cursor position, most useful if you have just made a typing error. The keytop on the backspace key sometimes says BackSpace, and it sometimes just has the symbol ←. In the latter case you should not confuse this key with the left arrow key, which merely moves the cursor to the left (non-destructively). The backspace key is located on the main section of your keyboard, immediately above the <Enter> key (↵).

Delete Word [Ctrl+T]

This command deletes the word at the current cursor position. If the cursor is in the middle of a word then only those characters at and to the right of the cursor are erased. If the cursor is on a space between words then spaces are removed up to the next word or punctuation mark. If the cursor was on a punctuation character then only that character is removed. Characters remaining on the line are shifted left to close the gap.

Delete Line [Ctrl+Y]

This command causes the current line to be removed, with remaining lines in the document being moved up to close the gap. Lines once deleted are lost.

Delete Remainder of Line [Ctrl+Q Y]

Erases all characters from the current cursor position to the end of the current line. Characters deleted are lost.

Delete Block (Native) [Ctrl+K Y]
Delete Block (CUA) [Ctrl+K Y or Ctrl+Delete]

Deletes the currently marked block (if no block is marked then this command does nothing). Text following the deleted block is moved to close the gap. Block commands are described fully in a later section.

Copy to Clipboard (Native) [CTRL+Insert]
Copy to Clipboard (CUA) [Ctrl+C or Ctrl+Insert]

Copies the currently marked block to the Windows clipboard. This may then be pasted into another edit window inside Odyssey, or into an edit control in another application.

Cut to Clipboard (Native) [Shift+Delete]
Cut to Clipboard (CUA) [Ctrl+X or Shift+Delete]

Copies the currently marked block to the Windows clipboard, and then deletes the block. The cut block may then be pasted back into the current edit window, into another edit window inside Odyssey, or into an edit control in another application.

Paste from Clipboard (Native) [Shift+Insert]
Paste from Clipboard (CUA) [Ctrl+V or Shift+Insert]

If the clipboard contains text placed there by Odyssey or another application then this command allows you to paste that text into the current file at the current cursor position. The pasted block then becomes the new "current" marked block.

Odyssey Text Editor

Search and Replace Commands

These commands allow you to search for a given sequence of characters in the text, and optionally to replace it with another sequence of characters. Several control options are provided.

Find String [Ctrl+Q F]

This command allows you to search for any occurrence of a string of characters in the current document. The **Find String** dialog is displayed, prompting you for the string to find, the string you last searched for being offered as a default (blank if there was no previous search), and showing current values for several search options. See the description of the Find String dialog for more detailed information on these options. You should make any necessary changes to the dialog, then press <Enter> or click on OK to perform the search.

If the requested string was found, then the cursor will be moved to a point immediately after the string (if the search direction was forward), or on the start of the string (if the search direction was backwards).

Find and Replace [Ctrl+Q A]

This command is used to search for a string of characters and then replace it with another string. This command is very similar to the Find command described above, except for the additional prompt in the **Find and Replace** dialog for the replacement string. There are also a couple of extra options used when replacing text - see the description of the Find and Replace dialog for further details.

Repeat Last Find/Replace [Ctrl+L]

This command repeats the last Find String, or the last Find and Replace, whichever was done most recently. For example, if a Find and Replace was used last then another Find and Replace will be performed.

Odyssey Text Editor

Block Commands

Block commands allow you to mark out a segment of text in order to apply an operation to that entire segment (block). For example after marking a block you can then choose to delete, copy, move or print that block. You can also cut or copy the block to the clipboard (or write the block to disk) in order to paste it into another editor, or another application, and you can read a file from disk and merge it with the current file, in which case the merged text becomes a new marked block in the current document.

Marking Blocks the CUA Way

The Odyssey text editor supports the WordStar™ style of marking blocks. However, it also supports the CUA style of text selection which you may find more convenient in many ways. Note that the "CUA style" of block marking is available regardless of whether the editor is operating in CUA or Native command modes. The following is a list of the various CUA-standard ways to select (mark) a block of text.

- **By selecting with the mouse**. As in any text control in a Windows application, you can select text by pointing and clicking where you want the selected region to begin, and then dragging the mouse (while holding the left mouse button down) to the place where the selected region should end. The selected block grows in size as you move the mouse around. Note that the "end" of the selected text can be before or after the starting point.

  If you move the mouse outside the boundaries of the edit window while selecting text, the editor will scroll the text window in the direction this indicates. For example, if you move the mouse to below the text window then the text window will scroll up. Similar things happen if you move the mouse to the left, right or above the edit window boundary.

- **By "double-clicking" with the mouse**. If you double click on any word in a document, that word is selected (marked as the current block).

- **Using Shift+<movement key>**. Editor movement commands are described in another section of this help guide. However, you should know that if you hold down the **Shift** key while you type any of the listed movement commands, then the text cursor is moved as documented, *and* all the text from the starting point to the new cursor location is selected (or deselected if it was selected already). You can apply several of these **Shift+movement** commands in sequence, to "grow" or "shrink" the region of selected text.

  Note however that this feature only works with single-keystroke movement commands, ie. **Shift+Ctrl+Home** works (**Shift+Ctrl+PgUp** in WordStar mode), but holding down **Shift** while you type **Ctrl+Q R** does *not* work.

Every time you mark a new section of text, any previously selected text is automatically deselected. You can also deselect text by clicking with the mouse anywhere within the boundary of the edit window.

Other sections of this help topic describe the alternative, WordStar method of block marking.

Mark Start of Block **[Ctrl+K B or F7]**

This command is used to tell Odyssey where you want a new block to begin. No physical marker appears in the text, however the entire block will be highlighted once you have marked both beginning and end (in either order). Once a block is marked and highlighted it may be the subject of other block commands such as move, copy or delete block.

Mark End of Block **[Ctrl+K K or F8]**

This command is used to tell the editor where you want a new block to end. No physical marker appears in the text, however the entire block will be highlighted once you have marked both beginning and end (in either order).

Mark Word **[Ctrl+K T]**

This command may be used if you wish to mark a single word as a block so that it may be copied or moved. Alternatively, you can mark a word by double-clicking it with the mouse, or by typing **Shift+Ctrl+→** (although the latter command is slightly different - it marks from the cursor position to the end of the word).

Mark Line **[Ctrl+K L]**

This command may be used if you wish to mark the current line as a block so that it can be copied or moved.

Hide/Display Block **[Ctrl+K H]**

When a block is marked and highlighted, you may un-highlight it using this command. However, the editor does not forget where the block markers were, so typing this command again will cause the block to become visible once more. The hide block command is most commonly used after a move or copy block operation, as this leaves a highlighted block at the destination position. If you intend no further operations on that block then this command will un-highlight it.

Copy Block **[Ctrl+K C]**

Makes a copy of the currently marked and highlighted block at the current cursor position. The new block then becomes the currently marked block. Attempts to copy a block onto itself are ignored.

Move Block **[Ctrl+K V]**

Moves the currently marked and highlighted block to the current cursor position. The block remains highlighted at its new position. Attempts to move a marked block into the highlighted area (ie. onto itself) are ignored.

Delete Block (Native) **[Ctrl+K Y]**
Delete Block (CUA) **[Ctrl+K Y or Ctrl+Delete]**

Deletes the currently marked and highlighted block, moving remaining text in the document up to close the gap. A block once deleted is lost.

Write Block to Disk **[Ctrl+K W]**

Writes the currently marked and highlighted block to a file on disk. The editor will prompt for a name to give to the new file. If a file exists already with the same name then you will be asked whether the existing file should be erased.

Read Block from Disk **[Ctrl+K R]**

This command is used to read text from a file on disk and insert that text at the current cursor position, leaving it as the currently marked and highlighted block. The editor will prompt for the name of the file containing the text you want to read.

Append Block to Disk **[Ctrl+K A]**

Appends the currently marked and highlighted block to an existing file on disk. The editor will prompt for the name of the file to which the block is to be appended.

Print Block [Ctrl+K P]

This command may be used to copy the currently marked and highlighted block to the printer, which must be online and ready to receive data.

If no block is highlighted when this command is entered then the entire document is printed.

Paste Block [Available from Command menu only]

Transmits the currently marked and highlighted block through the serial port to a remote host, which must be ready to receive text at the time - note that Odyssey has no way of checking this first. The effective speed of transmission while pasting can be controlled by adjusting the ASCII character and line delays in the Setup|File transfer... configuration section.

Indent Block [Ctrl+K I]
Un-indent Block [Ctrl+K U]

The block indent/un-indent commands increase or decrease the left margin offset of a block of text. For example, if the Indent block command is used on a selected paragraph which was indented to column five, then the indent is increased to column six. The Un-indent block command would reverse this.

These commands are most useful when working with Odyssey script source files (or other program source files), in order to apply a uniform re-indentation on a bracketed section of program code.

Copy to Clipboard (Native) [CTRL+Insert]
Copy to Clipboard (CUA) [Ctrl+C or Ctrl+Insert]

Copies the currently marked block to the Windows clipboard. This may then be pasted into another edit window inside Odyssey, or into an edit control in another application.

Cut to Clipboard (Native) [Shift+Delete]
Cut to Clipboard (CUA) [Ctrl+X or Shift+Delete]

Copies the currently marked block to the Windows clipboard, and then deletes the block. The cut block may then be pasted back into the current edit window, into another edit window inside Odyssey, or into an edit control in another application.

Paste from Clipboard (Native) [Shift+Insert]
Paste from Clipboard (CUA) [Ctrl+V or Shift+Insert]

If the clipboard contains text placed there by Odyssey or another application then this command allows you to paste that text into the current file at the current cursor position. The pasted block then becomes the new "current" marked block.

Odyssey Text Editor

Text Formatting commands

In fact, there is only one text formatting command provided in the Odyssey text editor.

Reformat Paragraph [Ctrl+B]

This command causes a paragraph to be reformatted such that it will fit between the defined left and right margins. If "Right Justify" is enabled in the Setup|Editor configuration section then the editor will format the paragraph such that each line in the paragraph is the same length, exactly meeting the right margin.

While the editor has a command to set a right margin, there is no equivalent command to set a left margin. Instead the editor will format every line to have the same indentation as the first line to be formatted. For example, to reformat a paragraph with a different left margin, move to the first character of the first line in the paragraph, type the space bar or backspace to adjust its indentation, then type **Ctrl+B** to reformat. All remaining lines in the paragraph will be given the same indentation.

The editor considers a paragraph to be any sequence of lines ending in a blank line. Since this is a pure ASCII editor it does not use special control characters to mark paragraph ends.

The reformat paragraph command will not work unless word wrap mode is enabled.

This command leaves the cursor on the line FOLLOWING the blank line which ended the paragraph. This is hopefully where the next paragraph begins, and is intended to make it convenient to step through a document, reformatting each paragraph in turn.

Odyssey Text Editor

Mode Control Commands

Odyssey text editor optional modes are controlled via commands which are prefixed with **Ctrl+O**. Most option commands are toggles, ie. the command enables the feature if was previously disabled, otherwise the command is disabled.

Toggle Auto-Indent [Ctrl+O I]

Auto-indent refers to the editor feature whereby, when you type <Enter> to begin a new line, that new line is automatically given the same indentation as the line just completed, and thus the cursor is placed immediately below the first character of the previous line. When Auto-Indent is disabled the new line is not given any default indent at all, and the cursor therefore moves to column one in the new line.

Toggle Word wrap [Ctrl+O W]

This command toggles "Word Wrap" mode, ie. if the word wrap was initially disabled then this command will enable it, and vice versa. When word wrap mode is enabled a "Wordwrap" symbol will be visible on the editor status line.

Word wrap occurs when the cursor reaches the right margin as you type. If that happens then the word currently being typed is moved to the next line, with the cursor left positioned after the last character. If the "Justification" option is also enabled then the editor will format the line just completed so that it exactly fits the line between left and right margins.

Toggle Justification [Ctrl+O J]

This command toggles "Right Justify" mode. If enabled, the editor ensures that every line exactly meets the right margin whenever a line or paragraph is reformatted. If disabled line ends are allowed to remain ragged.

Toggle Backups [Ctrl+O B]

This function controls whether or not the Odyssey text editor writes a .BAK file containing the old file contents, every time you save the file. If you disable this function then no backups are kept.

Toggle Hard Tabs [Ctrl+O H]

The Odyssey text editor can use **Hard Tabs** or **Soft Tabs** (note that this option is independant of the Smart or Fixed tabs option described next). In Hard Tab mode tab intervals are filled with actual tab characters (ASCII 9), whereas in soft tab mode the tab interval is filled with spaces. Hard tabs make the text file smaller, but may create formatting problems if you import the text into an editor that assumes a different tab interval. The Hard/Soft tab option *only* affects what Odyssey does when you insert a tab in the Odyssey editor - selecting soft tabs does not prevent Odyssey interpreting tab characters correctly when it reads in a foreign ASCII file. Note: the Setup|Editor dialog refers to this feature as the "*Tab Fill Character*". If the fill character is Tab then you are using hard tabs, if it is space then you are using soft tabs. The editor options menu contains a checkable "Hard Tabs" item (if this item is not checked then you are using soft tabs).

Toggle Fixed Tabs [Ctrl+O F]

The editor can use **Smart Tabs** or **Fixed Tabs**. Smart tabbing means that when you press the tab key, the editor examines the line above the current line, and aligns the cursor with the next word on that previous line. This is extremely useful when laying out tables. Fixed tabs are your usual fixed-interval tabs - the default interval being eight columns, though this can be changed in the Setup|Editor dialog.

Toggle EOL marker [Ctrl+O L]

The Odyssey text editor can read files whose lines end in CRLF (the normal DOS convention), or which end in LF only (the convention for text files originating on Unix systems). Odyssey does not mind if the same file contains a mixture of different line end types, and you don't need to set any modes in order to be able to read these files.

However, when you insert new lines, the Odyssey text editor must know whether you want the new line to end in LF or CRLF, which is why this option exists. If this option is enabled, then new lines will end in CRLF, otherwise they will end with LF. The editor **Options** menu shows the current state of the EOL toggle.

Set Tab Width [Ctrl+O T]

Odyssey defaults to a tab interval of eight columns when fixed tabs are being used. You can change the tab interval using this command. A dialog prompts you for the new tab width.

This option has no effect if you are using smart tabs.

Set Right Margin [Ctrl+O M]

This command is used to set the right margin required for word wrap and paragraph reformatting operations. The right margin is set at the cursors column position - note that you are not asked for a column number. The right margin setting will be made permanent if you use the "Save Setup" option of the Setup menu.

Odyssey Text Editor

Miscellaneous commands

This section groups together remaining editor commands which don't obviously fall into one of the previously described command categories.

Help **[F1]**

Displays help topics which describe editor commands.

Keyword Help **[Ctrl+F1]**

Searches the Odyssey help data for topics which are indexed under the word currently under the text cursor. This is of most use when editing Odyssey scripts, since it allows you to display help specific to a particular script command or script language keyword.

Compile **[F9]**

If the file loaded in the current window is a source file for an Odyssey script (a .SRC or .SCR file), or a keyboard mapper definition source file (a .KDF file) then the **F9** key allows you to compile that script or keyboard template directly, from within the editor. In the case of .SRC files, the compiled .SCR file is written to disk, assuming that the compilation was successful. The same thing applies to .KDF sources, which generate .KEY files if the compilation is successful. In the case of .SCR files this command simply performs a syntax/semantics check of the script, but does not attempt to write a compiled version of the script to disk.

If a syntax error occurs while the source file is being compiled then the compilation aborts, an error message is displayed on the status line, and the editor text cursor is set to the position of the syntax error. The error message is removed from the status line at the first keypress (note: that key is not swallowed).

Set Marker at Cursor **[Ctrl+K n]**
Jump to Marker **[Ctrl+Q n]**

The **Set Marker n** command (where n is a digit from 0 to 9) records the file position in an internal, numbered marker variable. You can then later jump back to that marker by entering the **Jump to Marker n** command, where n is the same digit. A marker once set remains set (automatically anchored to the character at which it was set), until the Set Marker command is used elsewhere with the same n, or until the text containing the marker is deleted. A block copy or move of a text region containing markers does not copy the markers (a block move counts as a copy followed by a delete, so the markers are lost at the delete step).

Upper Case Word **[Ctrl+U]**

If the cursor is resting on a word, this command forces that word to be all capitals.

Correct Case Word **[Shift+F7]**

If the cursor is resting on a word, and that word has occurred previously in the file, then this command changes the case of that word so that it matches the most recent previous occurrence of that word. This command is most useful when editing Odyssey scripts or other program sources when you want to ensure that a consistant capitalisation is applied to a program symbol throughout.

Close Edit Window **[Ctrl+K Q or Alt+F3 or Ctrl+F4]**

Closes the current edit window. You will be prompted for confirmation if the file has been modified and not yet saved.

Open File	**[F3]**

The Load File command allows you to open a new editing window, prompting you for the name of the new file. This is a shortcut for the **File|Open** menu command.

Save File	**[Ctrl+K S or F2]**

Saves the current file to its current name. A .BAK copy of the old file is made, if backups are enabled in the editor Options menu. The **Save File** command is a shortcut for the **File|Save** menu command.

Scroll Down	**[Ctrl+Z]**
Scroll Up	**[Ctrl+W]**

These commands scroll the edit window down and up, respectively. These are keyboard alternatives to using the vertical scroll bar.

Using Odyssey

The Directory Viewer

A directory viewer window is opened when you select **File|View directory** (from the terminal mode main menu), or when you click on the directory viewer toolbar button.

Please select one of the topics below for further information .

Odyssey Directory Viewer

Introduction

A **Directory Viewer** window looks very much like a directory view in the standard Windows *File Manager* application. A directory tree appears on the left, a list of files on the right, and a row of drive buttons at the top. The Odyssey directory viewer feature exists in order to allow you to perform specific Odyssey functions on the list of selected files - such as view selected text files with the Odyssey text editor, view GIF™, BMP and JPEG files with the Odyssey bitmap viewer, or upload a batch of selected files using Odyssey Zmodem file transfer. While all of this could have been implemented by making Odyssey a drag and drop client (in fact, we allow that too!), it turns out to be more convenient to have the feature built directly into Odyssey.

The Directory Viewer window consists of three main elements :-

The **Drive Ribbon** near the top of the window shows a list of disk drives on your system. This should include network drives, if you have access to any. Each item on this list is a "drive button" - you can change drives by clicking on one of these buttons. A white focus rectangle around one of the drive buttons indicates the current drive. The current drive is also shown as part of the directory viewer window title.

The **Directory Tree** panel on the left is a hierarchical listbox which shows you the structure of the directories on the selected drive. This list starts with a drive letter at the top (which indicates the root directory of the drive), and below which are all the subdirectories on that drive. These subdirectories are all indented (moved further to the right) with respect to the root directory, to show that the subdirectories are *children* of the root directory. If subdirectories of the root themselves contain further subdirectories then these sub-subdirectories (if shown) will again be indented, relative to the parent directory item.

You may not be interested in seeing the entire directory tree for a drive; you may in fact be interested in seeing only the children of one directory. If you don't want to see the children of a directory tree item then simply double-click on that item and the directory viewer will "collapse the branch" (ie. hide the children away). If you want to expand the branch again in future then just double-click on that item again. Alternatively, you can use the **+** and **-** keys respectively to expand and collapse tree branches, or you can use the **Command|Expand branch** and **Command|Collapse branch** items from the directory viewer Command menu.

Note the little icons which are shown to the left of each directory name :

The icon (meant to look like a closed folder) indicates a directory which has no children. When this directory is open the icon changes to
.

The icon indicates a directory which *has* children, but these children are not currently displayed (the branch containing the children has been collapsed or has never been expanded). When this directory is open the icon changes to
.

Finally, the icon indicates a directory which has children, and those children are also displayed. When this directory is open the icon changes to
.

The **File List** panel to the right of the directory tree shows filenames in the current directory which match the current search pattern. You can choose whether to show only file names in this list, or you can show full file details (size in bytes, date and time of last modification, file attributes). Later sections of this help chapter will tell you how to make that choice.

Note that the area below the drive ribbon is a "split window", with the panel on the left side of the split containing the directory tree, and the right side of the split containing the list of files. You can change the relative sizes of these two panels by dragging the split bar to a new horizontal location (click on the dividing bar with the mouse, then hold the mouse button down while you move the mouse to the new location for the bar).

Odyssey Directory Viewer

Changing Drives

To change drives with the mouse: Simply click on one of the drive icons.

To change drives with the keyboard: Press the tab key until the drive ribbon has the keyboard focus (the solid white focus rectangle changes to a dotted rectangle). Then use the left and right arrow keys to move the focus rectangle to the drive you want, and finally press **Space** to change the current view to that drive.

☞ If you press **Enter** instead of space, Odyssey will open a second viewer window showing the contents of the selected drive.

Odyssey Directory Viewer

Changing Directories

With the mouse: click on a directory name in the directory tree, or double click on a directory name in the file list.

With the keyboard: When the directory tree has the focus you can use the arrow keys to move to other directories (the file list panel is updated as soon as the new directory tree item is highlighted). If the file list panel has the focus then you can change directories by using the arrow keys to highlight the directory you want to change to, and then pressing the **Enter** key.

Odyssey Directory Viewer

Selecting Files

Before you can carry out operations on files or directories you must first tell Odyssey which files and directories you want to work on. You do that by selecting them from the file list panel. Items in the file list can be selected using either mouse or keyboard, as described below.

Selecting file list items with the Mouse.

- Select a single file by clicking on it with the mouse (this also deselects any previously selected items).
- Select a group of files by clicking on the first item of the group, and then **Shift+Click** on the last file in the group. This selects the first item, the last item, plus everything in between.
- Toggle the selection state of an individual file by pressing **Ctrl+Click.**
- You can select several groups of files if you use **Ctrl+Click** to select the first item, and then use **Shift+Click** to select the last item.

Selecting items with the Keyboard.

- As in any Windows application, you can select an item or a group of items by moving to the first item you want to select, and pressing **Shift+<movement key>** to select all the files between the original and final location of the highlight bar. This action deselects any previously selected items.
- You can also select files individually with the keyboard by first pressing **Shift+F8** (the focus rectangle begins to flash), then you move around the file list pressing space when you reach items whose selection state you want to toggle. Finally press **Shift+F8** again to end the special selection mode.

☞ Selection states are lost if you change directories.

Odyssey Directory Viewer

Copy or Move Items

When you **copy** an item (a file or a directory), an exact duplicate of the item is made and placed wherever you tell Odyssey to put it; the original file is not affected by the copy operation. When you **move** an item a copy is made as before, but this time the original is deleted. Files (or directories) retain their original name after the move or copy; you should use **File|Rename** if you wish to rename a file or directory. If the source and destination drives for a **move file** operation are the same, then Odyssey does not physically copy the file data, instead it simply renames the file, giving it a new path. If the source and destination drives are different then Odyssey must physically copy the file data, which you will no doubt notice, because it takes longer to complete.

The destination directory for a move or copy must be different from the source directory - an item cannot be moved or copied onto itself, because DOS does not allow two files in the same directory to have the same name (copy), and moving a file to the same place it was moved *from* does not make sense.

Moving or Copying Files and Directories.

Using the File menu: Having selected the items you wish to copy, select **File|Copy** - a dialog will appear which will prompt you for the name of the destination directory. If you are moving items (rather than copying), then select **File|Move** instead.

Using the toolbar: Do as above, but instead of pulling down the file menu, just click on the **move** or **copy** toolbar button instead.

Using "Drag and Drop": Select the item(s) you wish to copy, then click on one of the selected files and, *while holding down the mouse button*, drag the selection to the destination directory where it appears in the directory tree. Note that the destination directory must be visible in the directory tree before you start dragging, for this to work as described. As you are dragging the files you will note that the mouse cursor changes to reflect a dragging operation. When you release the mouse button a dialog will appear, asking you to confirm the operation.

If you merely drag the items as above, Odyssey will assume that you mean to copy the selection. To perform a move operation you drag the items to the destination directory as described above, but this time press and hold down the **ALT** key, and keep it down until *after* you have released the mouse button over the destination directory.

Alternatively, you can drag the selected file(s) onto the **move** or **copy** toolbar buttons. In this case a dialog will appear which prompts you for the name of the destination directory.

Odyssey Directory Viewer

Deleting Files or Directories

For the sake of safety, Odyssey puts certain limitations on the things you can delete using the procedure described below. You are not permitted to delete the Odyssey home directory, and nor are you allowed to delete the root directory on any drive. This still leaves more than enough rope to hang yourself with, so do please be careful!

Deleting Files and Directories.

Using the File menu: Having selected the items you wish to delete, select **File|Delete...** - a dialog will appear, asking you to confirm that you really do want to delete the selected items.

Using the toolbar: Do as above, but instead of pulling down the file menu, just click on the **trashcan** toolbar button instead.

Using "Drag and Drop": Select the item(s) you wish to delete, then click on one of the selected files and, *while holding down the mouse button*, drag the selection to the "trashcan" toolbar button.

☞ Once deleted, files and directories are gone for good - you cannot undo the deletion. Thus the moral is: stop and think before you delete *anything*.

Odyssey Directory Viewer

Renaming Files or Directories

Odyssey does not allow you to rename the Odyssey home directory, but doesn't stop you renaming anything else. Be careful when renaming directories, since that may invalidate search paths and other path information stored in the configuration files of your applications.

To Rename Files or Directories

First select the item(s) you wish to rename, then select the **File|Rename...** menu item. You will be prompted for a new name for the file or directory.

If you are renaming a group of items, then the "new name" must be a wildcard.

Odyssey Directory Viewer

Viewing Files

To view a file or file(s), first select the file(s), then select the **File|Open...** menu item, or click on the "Open document" toolbar button.

The exact meaning of "view" depends on the file type. If the selected file is an application (.EXE file), then that application runs. If the selected file is a Windows help (.HLP) file then Odyssey asks the Windows help application to display its contents. If the file contains a bitmap, then that bitmap is read and a Bitmap Viewer is opened. Currently, Odyssey only recognises GIF,BMP,JPEG and it's own FAX format as bitmap files. If the file is none of the types mentioned then Odyssey assumes it to be a text file, and opens a text editor window.

To prevent accidentally opening a zillion selected files, Odyssey ignores anything after the first ten selections (an arbitrary limit).

Odyssey Directory Viewer

Listing Different File Types

Odyssey defaults to displaying, in the file list panel, all the files in the current directory which match the search pattern ***.***. However, you can change this by selecting the **File|Show Files of Type...** menu item, or by clicking the "New file type" toolbar button.

In either case, you will be prompted for the new wildcard, and the files listed will then change to show only files matching the new pattern. For example, changing the pattern to *.TXT will cause the file list panel to be repainted, only listing files in the current directory which have the .TXT extension.

☞ The file list always includes a list of the subdirectories in the current directory, regardless of the current file type selection.

Odyssey Directory Viewer

Listing Files in Detail

When Odyssey displays a list of files in the file list panel, it normally shows only the name of each file. You can change this by pulling down the Directory Viewer **Command** menu.

Note that the first two options on the menu are:-

√ **Show filenames only**
Show all file details

When the file listing contains only file names, the first of these menu items will be checked. If you select the **Show all file details** option then *it* will be checked, and the file listing will change to a single column format, containing all the details you would normally see in a DOS directory listing, ie. the file size in bytes, the date and time the file was last modified, plus file attributes as listed in the table below:-

A -	The file has the "archive" attribute bit set, meaning that it has been modified since the last time it was backed up.
H -	The file is hidden (it does not appear in a normal DOS directory listing).
S -	The file is a system file.
R -	The file is marked read-only.

Odyssey Directory Viewer

Changing the File Sort Order

By default, the Odyssey directory viewer lists files sorted in alphabetical order of their name ("a..." first). However, other sorting methods are supported. To change the sorting method, pull down the directory viewer Command menu, and choose one of the **Sort by xxxx** options - a check mark will be shown beside the menu item relating to the current sorting method. In greater detail, the sorting options are :-

Sort by name: Files are listed in alphabetical order of their name. This is the default sorting order, as described above.

Sort by type: Files are sorted first by their extension, and then (within a group of files with the same extension) by name. This has the effect of ensuring that files with the same extension are listed next to each other.

Sort by size: Files are sorted so that the largest files appear at the top of the list, and the smallest files at the bottom. If your disk is nearly full then finding the largest files you no longer need, and deleting them, has the quickest payoff in terms of recovering disk space.

Sort by date: The files are sorted by age, with the oldest files at the top of the list. You might use this to find files which you haven't needed in a long time, and liberating the disk space they occupy.

Odyssey Directory Viewer

Uploading Files

This feature is one of the main reasons Odyssey implements its **Directory Viewer** facility, since it gives you the power to select a list of files and upload them to a remote host, all in one batch (assuming a batch file transfer protocol is used).

Now, while it is true that the standard Upload command in terminal mode allows you to upload a batch of files, you normally have to indicate *which* batch of files by means of a single wildcard specification. That's fine if all the files in the batch can be covered by a single wildcard that doesn't also pull in unwanted files, but if they can't, then you need the directory viewer upload feature described here.

To upload a list of files, first select all the files you want, using methods described earlier (the files do all need to be in the same directory). Then pull down the **Upload** menu and select your upload protocol, or, if you intend to perform a Zmodem upload, then you can simply click on the "Zmodem Upload" toolbar button instead. You could also drag the selected files onto the "Zmodem Upload" toolbar button.

Note that you could, if you wished, preselect a list of files prior to connecting to a remote host, leaving the directory viewer open (though perhaps minimized) while you dial. Then once connected, switch to the open directory viewer and upload the selected files, as described above.

☞ If you try to upload a batch of files using a non-batch protocol, eg. Xmodem, Ymodem or ASCII, then Odyssey will only upload the first of the files in the batch. The feature described above is intended for use with a batch protocol, ie. Ymodem batch, Zmodem, SuperKermit or Compuserve B+.

Using Odyssey

The Bitmap Viewer

Odyssey includes a **Bitmap viewer** window class primarly in order to implement the **View GIF/JPEG while downloading** feature (selected in the Setup|File transfer... dialog), and of course for use in the FAX server for viewing received FAX documents and for previewing FAX documents you are about to transmit.

Since we must provide a bitmap viewer, we decided to also make it available for use in other situations, such as when you double click on a bitmap file in the Directory Viewer file list panel. The bitmap viewer itself is completely generic, however we currently only provide file *readers* for GIF,BMP,JPEG and FAX file formats - other file formats may be supported in due course, remembering of course that what we have here is a comms package, not an image processing utility.

For further information on the bitmap viewer, please choose a topic from the list below:

Viewing a Bitmap file
Printing a Bitmap
Copying Bitmaps to the Clipboard
Moving around a large Bitmap image
Deleting Bitmap Files

Odyssey Bitmap Viewer

Viewing a Bitmap file

If you enable the **View GIF/JPEG while downloading** option of the Setup|File transfer... dialog, then a Bitmap Viewer window will be opened automatically when you begin downloading a GIF or JPEG file; this happens regardless of the file transfer protocol you use.

Alternatively, you can manually open a bitmap viewer by double-clicking on a GIF,BMP,JPEG or FAX file in a Directory Viewer file list panel.

Finally, if you double-click on the "List of Received FAXes" displayed by the FAX Server then a bitmap viewer is opened, and the selected FAX is displayed inside it.

Odyssey Bitmap Viewer

Printing a Bitmap

You can print a bitmap by reading the bitmap file into an Odyssey bitmap viewer and then selecting **File|Print bitmap...** from the Bitmap Viewer main menu, or by clicking on the "print" toolbar button.

Odyssey Bitmap Viewer

Copying Bitmaps to the Clipboard

You can copy a bitmap image to the clipboard by first reading the bitmap file into an Odyssey bitmap viewer and then selecting **Edit|Copy...** from the Bitmap Viewer main menu.

The bitmap is always copied to the clipboard in DIB (CF_DIB) format, in order that palette information is preserved for the target application.

Odyssey Bitmap Viewer

Moving around a large Bitmap image

Many bitmap images you would like to view with the Odyssey bitmap viewer are too large to view in their entirety, even when you maximize the viewer window, unless you have a larger than average display resolution. Most current Windows users have 640x480 or 800x600 resolution displays - the largest images you are likely to want to view go up to 1024x768 or even (in the case of a full page 200 dpi FAX image), as large as 1720x2300 pixels). To handle these larger images you must be able to scroll (or *pan*) around inside the image. This section outlines the menu/keyboard commands which allow you to do just that. Note that you can shrink an image by clicking it with the right mouse button; you can also enlarge an image by clicking with the left mouse button.

The **arrow keys** and **page keys** can be used to roam over the image, as can the **scroll bars**.

The **Home** and **End** keys can be used to jump directly to the left or right sides of an image. The **Shift+Home** and **Shift+End** keys jump to the top or bottom of an image.

If the bitmap file contains more than one page (eg. a multi-page FAX file), then **Ctrl+Home** can be used to move to the first page, and **Ctrl+End** moves to the last. **Ctrl+PgDn** moves to the next page, **Ctrl+PgUp** moves to the previous page.

As in any Odyssey MDI child window, the **F5** key can be used to zoom and unzoom the bitmap viewer.

Many of the above commands are duplicated in the bitmap viewer **Command** menu.

Odyssey Bitmap Viewer

Deleting Bitmap Files

Sometimes, having viewed a bitmap image you have just downloaded, you will decide that you do not wish to keep that image. You can delete the bitmap file when the viewer is active by selecting **File|Delete bitmap...** or by clicking the "trashcan" toolbar button. You will be asked to confirm that you really want to delete the bitmap file.

You are not allowed to delete a FAX file if the bitmap viewer was opened in order to preview a FAX before transmission, because the FAX server always deletes such temporary fax files itself.

Odyssey Bitmap Viewer

The GIF File Format

The ***Graphics Interchange Format*** (GIF) was invented by Compuserve Inc. to allow Compuserve users to exchange graphics images using a standard format. GIF incorporates the following features, all of which are supported by Windows Odyssey:-

- ***LZW compression*** to reduce file size (and hence upload/download times). This is essentially the same compression algorithm used by the popular ARC and ZIP utility programs, and by V.42bis modems.

- Optional use of an ***interlaced*** image format, ie. instead of storing each successive scan line in order, the first part of the file contains every eighth scan line, then every fourth scan line, and so on until the complete image is filled in. This is most useful when viewing an image as it is being downloaded, since it allows the user to see the complete image (albeit at lower resolution) when the download is only partly complete; the user could then decide to cancel the download if the image turned out to be uninteresting. Note however that few GIF images actually make use of this encoding option.

- Support for monochrome images, and palette based images up to 256 colors.

- A *streamable* format, ie. all the information necessary to decode the image is provided in a header which precedes the image data, hence it is possible to display the image accurately as it is received. This feature would not be possible, for example, with a 256-color PCX image, which stores the palette at the end of the image, or with TIFF images of any resolution, which have no fixed order for image components.

Windows Odyssey can decode all GIF images conforming to the GIF89a and earlier specifications.

The GIF format has a number of limitations and/or weaknesses:-

- The lossless LZW compression method is only truly effective with particular types of images, ie. those in which repeated substrings occur frequently. An example of an image which would compress well is a simple line art or "cartoon" type image, consisting of large regions of the same color. Simple run-length encoding would also be highly effective with such images, so one is led to wonder whether the choice of LZW compression is a particularly good one. True life images (scanned photographs etc), will not achieve high compression ratios with the LZW scheme - while there *is* repetition in such images, simple LZW type algorithms will not detect it.

- The GIF format, currently, cannot handle 24bit images. Because of this, and because other (lossy) schemes get much better compression, many hosts are nowadays gradually moving towards JPEG as the standard exchange format for callers.

Odyssey Bitmap Viewer

The BMP File Format

The **BMP** file format is the native portable graphics format for Windows. It originated on OS/2, and has gradually been extended as hardware capabilities have increased. When loaded into memory, a BMP (minus header) is known as a DIB (*Device Independant Bitmap*), which distinguishes it from a DDB (*Device Dependant Bitmap*), which is the internal image format your video device driver uses. BMPs support bitmaps containing 1, 4, 8 and 24 bits per pixel. A new option introduced in *Windows NT* supports bitmaps containing 16 bits per pixel.

The BMP/DIB format has a number of peculiarities which make it awkward (and hence slower) to use, compared to DDBs. In particular, a BMP/DIB image is stored in the file upside down (the first scan line in the file is the last scan line of the image), which presents certain problems and overheads when reading and manipulating BMPs. Also, 24 bpp BMPs store the pixel components in Blue-Green-Red order, which is the reverse of the order used by most other formats (and incidentally, is the reverse of the order as documented by Microsoft).

It is known that future versions of Windows address some of these peculiarities, particularly there will be an optional variant of the format which stores images the right way up.

Odyssey Bitmap Viewer

The JPEG File Format

JPEG (*Joint Photographic Experts Group*), is actually a data stream format, not a file format. The JPEG standard leaves a number of factors unspecified, particularly the color model used (ie. RGB vs YIQ vs HLS etc). Therefore, most PC based JPEG readers/writers actually conform to the more specific "JPEG File Information Format" (JFIF) standard, which specifies the YCbCr color model for 24bpp images.

JPEG uses a lossy DCT based scheme for image compression, and hence gets the highest compression ratios of any popular image format, but only gets this performance when working from the original, unprocessed true color image. In particular you should note that both quality and compression ratio will be lower if you attempt to JPEG-compress an image which has been previously processed (eg. dithered and saved as a GIF).

Note that a JPEG image never uses a palette - the image is always either grayscale (8bpp) or truecolor (24bpp). This obviously has implications for the display of JPEG images on the 256 color displays used by typical Windows installations, since it means that an image must always be dithered before it can be viewed satisfactorally.

There are a number of options Odyssey could have used to dither a JPEG image - most readers use a simple ordered dither, which has the merit of being quite fast, but introduces unacceptable artifacts into the image. Instead, Odyssey does an "error diffusion" dither, which takes slightly longer, but does not introduce noticeable artifacts.

Odyssey uses a fixed palette for dithering all JPEG images, not an optimized palette. Optimizing the palette would produce *much* better results than a fixed palette dither, however it also takes much longer to do. We felt it was better to try and achieve decent quality very quickly, and leave you to use dedicated JPEG reader applications when you need superior quality.

Odyssey Bitmap Viewer

The FAX File Format

The .FAX format is a proprietory format used by Odyssey to store multi-page FAX documents. The format consists of a simple header, and then multiple images in CCITT Group III (T4) FAX format.

The information given below is provided for the benefit of programmers who wish to read Odyssey FAX files. Note however that we do ***not*** promise that this format will remain unchanged in future releases.

The following is the format of a .FAX file header (256 bytes) :-

```
typedef struct {   /* 256 byte header */
   /* 0*/ char Marker[4];
   /* 4*/ word Version;
   /* 6*/ word HeaderSize;
   /* 8*/ word HDPI,VDPI;
   /*12*/ word Pages;
   /*14*/ word PageIndex;
   /*16*/ word Date;
   /*18*/ word Time;
   /*20*/ char RemoteID[22];
   /*42*/ word RxError;
   /*44*/ byte Reserved[84];
  /*128*/ word Index[64];
} FAXHDR;
```

The fields in this header record are as follows:-

Marker - Provides a signature which can be used to verify that this is an Odyssey .FAX file. The signature is "ODFX" - note that there is no terminating NUL byte.

Version - identifies the sub-format of this particular FAX file. Windows Odyssey v2.00 writes the value 200h into this field.

HeaderSize - the size of the .FAX file header, in bytes. Image data for the first FAX page begins at offset *HeaderSize* from the beginning of the file.

HDPI - the horizontal resolution of each FAX image, in dots per inch. Currently, this value is always 200 decimal. Note that all FAX pages in a single file have the same resolution.

VDPI - the vertical resolution of each FAX image, in dots per inch. This value will be 100 for normal images, and 200 for "detail" images.

Pages - the number of pages in this FAX file.

PageIndex - the offset, in bytes, from the beginning of the file to the "page index", which contains the file address of each FAX page. In the current FAX format, this value is always 128.

Date - The date this FAX file was created/received, using the same format as a DOS file date/time stamp.

Time - The time this FAX file was created/received, using the same format as a DOS file date/time stamp.

RemoteID - This is the station ID of the remote FAX station which transmitted this FAX. This field is only valid for received FAX files (it will be zeroed in FAX images created locally).

RxError - The error code returned by the FAX receive protocol routines. This should be zero if the FAX was received without errors. Note that this field is of course valid only in FAX images received from a remote FAX station.

Reserved - fill bytes used to pad the header to 256 bytes, some of which may be used for other purposes in future Odyssey releases.

Index - The file offset of each page in the FAX document. Each offset uses units of 128 bytes; ie. a reader should multiply this value by 128 to obtain the byte offset of the beginning of a page. Note that the size of a page n can be calculated using the formula **(hdr.Index[n+1]-hdr.Index[n])*128**, provided that n does not identify the last page in the file (in which case the formula should be: **FileSize(fax) - hdr.Index[n]*128**).

The FAX pages themselves are encoded using raw Group **III** (T4) compression, ie. they are stored exactly as received. Each page is however padded up to the next multiple of 128 bytes.

Using Odyssey

The FAX server

The Odyssey **FAX Server** allows users to send text and graphics files as FAX documents, receive FAX documents, view FAXes, export FAXes to various standard graphics formats, print FAX documents, and so on. The FAX server module can be accessed via the Odyssey **Command** menu, or by pressing **ALT+V** while the terminal window is active. The FAX server can also be controlled entirely from a script.

For further information about the FAX server, please select one of the topics below:

☞ Note that the Odyssey FAX server is an integrated part of Odyssey, and is not intended to be used as a "printer driver" by other applications.

Odyssey FAX Server

Introduction

The **FAX Server** allows users to send text and graphics files as FAX documents, receive FAX documents, view FAXes, export FAXes to various standard graphics formats (eg, for processing by an OCR package), print FAX documents, and so on.

The FAX Server is activated by either selecting **Window|FAX server...** from the terminal window menu, or by pressing **ALT+V** when the terminal window is active. There will normally be a brief pause while the FAX server configures the modem for FAX operation (a dialog appears telling you that this is happening), and then the "Received FAXes" window will be displayed, which lists the names of any FAX files currently in the FAX receive directory.

If no FAX modem (or no modem at all) is connected and powered up, then the "brief pause" mentioned above may go on for quite some time (20 seconds or so), until Odyssey finally concedes that your modem just doesn't want to play.

The first time you invoke the FAX server you should check the "**Local station ID**" and "**Path for FAX files**" (both configured in the Setup|Fax... dialog), before doing anything else. These items are normally set up by the INSTALL program when you first installed Odyssey, but it will do no harm to check again now. Before changing the local ID you should read the documentation on the local ID option, in particular the parts about which characters are allowed. Note that if you enter a directory name in the "Path for FAX files" field then that directory *must* exist - Odyssey will not create it for you.

The FAX server always automatically saves between sessions the values you enter in the various menu options, including the local ID, receive directory, and "Send FAX" dialog options.

We feel we should warn you that many modems on the market today (especially class II modems) are buggy, ie. they do not conform properly to the specifications published by the EIA, which is the only interface specification programs like Odyssey have to work with. Because of this, it is ***impossible*** to guarantee that our FAX module will work with all FAX modems. Older FAX software will generally be more reliable, because the authors of FAX software which has been on the market for some time have had many modem bugs reported to them, and will have gradually collected and incorporated workarounds in their product so that, over time, fewer modems give problems as far as the end user is concerned - at least until the next batch of new modems appear. Since Odyssey FAX support is relatively new we do not have the benefit of this maturity; while we have tried very hard to fix all the problems we have discovered thus far during testing, it is inevitable that some modems somewhere will still cause problems. However, if you report such a problem to us we will make every effort to correct the problem as soon as possible.

Odyssey FAX Server

Fundamentals

The Rise of the FAX machine

Those of us who have been using modems for much of the last ten years have spent a great deal of that period evangelising about the technology to a dubious non-technical audience. To us it has seemed obvious that the ability to send and receive files by modem, or converse with other modem users using text messages was a wonderful thing, which was bound in the end to replace the infinitely inferior technology of the ordinary postal service; derisively referred to as "snailmail".

The letter post has indeed been replaced to a certain extent, but not by the technology we expected. Instead a subset technology has been developed based around a single unit consisting of a scanner, printer and modem, the so-called facsimile or FAX machine. Although the components of a FAX machine are familiar items to computer users, the combination has become a consumer item, infinitely easier for the naive user to cope with than the alternative we (as techies) might have preferred.

The success of the diminutive FAX machine cannot easily be exaggerated - in fact its use has grown so much that in the last couple of years it has threatened to expand into the techie domain, in the form of stand-alone dual purpose modems capable of both data and FAX operation - add a separate scanner and printer and your FAX modem becomes the equivalent of a FAX machine; in fact you don't even need the scanner, since it is possible to generate the FAX image of a document entirely inside the computer without ever producing a paper copy!

The only difference between a "data" or "FAX" modem is the signalling standards most commonly used - the signalling standard refers to the tones (or signal levels, or phase modulation, or whatever) used by the modem to represent binary data. For data modems V21 is the most common 300 bps standard, and other common data modem standards include V22 for 1200 bps, V22bis for 2400 bps, V32 for 9600, and V32bis for 14,400 bps. Data modems are also generally full duplex, ie. they have two separate communications channels making it possible to send data and receive data at the same time.

On the other hand, a FAX modem uses V.21 for 300 bps, V.27ter for 2400 or 4800 bps, V.29 for 7200 or 9600 bps, and V.17 for 14,400 bps. Also, FAX modems are generally only half-duplex, i.e. data can only be sent in one direction at a time. This means that it is generally not possible for a FAX modem to communicate with a data modem unless the latter includes specific FAX capability (many users have been fooled into buying a 2400 baud modem with "9600 baud FAX", believing that they would be able to send ordinary data at 9600. This is wrong of course - you need a V32 data modem to do that).

FAX Modem Interface Standards

Of course, the fact that a data modem has additional FAX capability is not the end of the matter. In order to use that facility you must also have software capable of operating the FAX part, rather than just the normal data part. Although FAX modems have been around for a few years, for the first couple of those years there was no standard way of controlling them, so the early FAX modem user generally had to either learn to live with the standard software bundled with the modem, or do without PC-FAX capability.

The old problem of lack of standards has now been more than solved, in fact as is often the case, we now suffer from a surplus of them. In 1990 the Electronic Industries Association (EIA) approved a FAX modem control standard commonly referred to as "EIA Class I". Class I simply extends the de-facto standard Hayes "AT" command set to include a new set of commands for controlling FAX modulation schemes, synchronous communications, and so forth.

Very recently (in late 1992) a new version of the EIA standard has appeared called "Class II". It still consists of a set of AT commands, but now the modem firmware contains much of the code required to implement the FAX protocol, which for class I modems was part of the task which the comms software had to manage. For users however the difference between class I and class II is irrelevant, provided that the communications software properly implements the remaining parts of the FAX protocol - whatever that might be.

A competing standard called "*Communicating Applications Specification*" (CAS) has also appeared. CAS, which is supported by DCA and formerly by Intel, is implemented at a much higher level than EIA Class I/II - in fact, it doesn't deal with the FAX modem at all, but instead specifies a callable program interface which would be implemented as a DOS device driver supplied with the modem, and which client software would use in much the same way it uses a memory manager, or a mouse driver. CAS does not appear to have been a great success so far - several PC applications have supported it, but few modem vendors have supplied compliant drivers to control their modem - others reasons for this lack of success might be that users are reluctant to install a memory hogging device driver if they can possibly avoid it, and also because of difficulties accessing the CAS driver software from protected mode environments such as Windows.

There is talk of other standards, for example there is FAXBIOS, supported by WordPerfect Corp and Everex (a CAS-like standard) and supposedly there is also a new CCITT standard called T.Applecon or T.611 which is due to appear (Intel are even rumoured to have switched allegiance from CAS to T.611).

Currently, the Odyssey FAX Server module supports EIA/TIA class I and II modems. Should support pick up for competing standards then we will update the package appropriately at that time.

Some things which may surprise you....

- *FAX machines are bitmap oriented*. The main thing you must remember about FAX is that it is a standard for transferring *image* data between facsimile machines. If you send a text file from your PC it doesn't arrive as a text file, it arrives as a series of images, each of which corresponds to a single imaged page. If you wanted to convert that to ASCII text you would need to run the images through an OCR package. The Odyssey FAX module supports the export of received FAX documents to PCX or TIFF format, which are the formats most commonly supported by PC OCR packages. We may even bundle our own OCR package one day.

- *A Laser Printer doesn't improve image quality*. Our first reaction at Skyro Software to the notion of PC based FAX was "We'll be able to output FAX pages to a 300 dpi Laser, instead of that awful FAX thermal thingy".... if something similar occurred to you, then you will be in for a bit of disappointment. First, the resolution of the source image is far more important than the resolution of the printer - in the case of standard FAX image format, the resolution is only (approximately) 200 dpi in the horizontal direction, and 100 dpi in the vertical - so you just don't have the detail in the original image to make full use of the printers 300 dpi capability.

  Secondly, you may be surprised to discover (it certainly surprised us), that the quality of a FAX page has little to do with either the print quality, or of errors introduced during transmission, but is mostly to do with the poor quality of scanner used by the typical cheap office FAX machine - if you were able to directly capture the output from such a scanner and save it as a PC file, the result wouldn't look that much better than it does after it has been transmitted, then received and printed on the remote FAX machine.

  So, a FAX image printed on your 300 dpi Laser or Inkjet is still going to look unmistakably like a FAX image. You *do* have the more modest advantages however that your FAX can be printed on good quality single sheet paper instead of that "horrible thermal roll which won't lie flat", and you can also take comfort in the knowledge that the FAX page won't turn black if you accidently leave it in direct sunlight, or near a heater.

Odyssey FAX Server

Sending a FAX

The **Send FAX** facility allows you to send a FAX document to a remote FAX station. The file to send can be in FAX, PCX, TIFF or ASCII text format. The FAX server uses the filename you supply to guess the format - if the filename extension is not .FAX, .PCX or .TIF then it is assumed to be ASCII text.

You send a FAX by first selecting **Command|Send FAX...**, or by clicking the "Send FAX" toolbar button. Either of these actions causes the Send FAX dialog to be displayed, whose fields are described below. Note that most of the fields on this dialog provide the information necessary to flesh out the cover sheet macros (from, to etc). This information is not strictly required (even if you enable the "Send cover sheet" checkbox), but the generated cover sheet will look rather odd without it. The fields on the dialog are as follows:-

File to send: This field tells the FAX server which file contains the data which is to be transmitted as a FAX - note that a full path to this file must be supplied, if the file is not in the Odyssey directory. If the file to send is ASCII text, and is only a few lines long then you might like to omit this item and make use of the "*Add-Info file*" field instead. The "**search...**" button to the right of the "File to send" field brings up a standard file selection dialog, which is helpful if you can't remember the full path details.

If you don't supply a file name in this field then you *must* supply a file name in the "Add-Info file" field instead.

FAX number: This is the telephone number of the FAX machine you want to call. If you don't supply a number then the FAX server expects you to dial manually, and will prompt you appropriately when it is ready for you to dial.

Reformat text to fit margins: This field is ignored unless the file being sent is ASCII text. Most word processing packages have an option to export a document to ASCII text format, but many packages when doing so will create a file in which each text paragraph is a single long line which then needs to be reformatted to fit within the margins set in the importing application (the Odyssey FAX server in this case). If you would like the text to be formatted to fit inside the FAX page margins, then enable this checkbox. If checked, then any line which extends beyond the right margin is broken at a word boundary so that it fits. If not checked, then lines are simply truncated at the right margin.

When preparing a text file, it is not necessary to indent the text from the left, since indents are automatically applied to the left and top of the page by the FAX conversion routines. Each text line on the FAX page is a maximum 92 characters, assuming you use the default fonts supplied.

Preview FAX before sending: If this checkbox is enabled, then Odyssey will give you the chance to view and approve the FAX image resulting from an ASCII to FAX conversion before it is transmitted to the remote FAX station.

Send cover sheet: If this checkbox is enabled, then Odyssey will generate a cover sheet, and will transmit it prior to the main fax document.

Quality: This field consists of a pair of radio buttons which control the resolution of the FAX which is generated and transmitted. If quality is set to *Normal* then the FAX resolution will be 200 dpi horizontally by 100 dpi vertically. If quality is set to *Detail* then the resolution will be 200 dpi both horizontally and vertically. Note that while FAX image quality is significantly enhanced by the detail option, it also means that the FAX takes twice as long to transmit.

None of the fields described below are relevant unless the "*Send cover sheet*" option is enabled.

From: This is used to fill in the "from" macro of the cover sheet as defined in the COVER.TEM file. "From" is normally the name of the person sending the fax document.

To (Company): This is used to fill in the "to" macro of the cover sheet as defined in the COVER.TEM file. "To" is normally the name of the company to which the fax is directed.

Attention: This is used to fill in the "attn" macro of the cover sheet as defined in the COVER.TEM file. "Attn" is normally the name of the individual for whom the fax is intended.

Our Ref: This is used to fill in the "oref" macro of the cover sheet as defined in the COVER.TEM file. "Our Ref" is the senders file reference, account reference etc associated with the subject of the fax.

Your Ref: This is used to fill in the "yref" macro of the cover sheet as defined in the COVER.TEM file. "Your Ref" is the recipients file reference, or account reference etc associated with the subject of the fax.

Subject: This is used to fill in the "subject" macro of the cover sheet as defined in the COVER.TEM file. "Subject" is a one line description of the subject matter of the fax document.

Additional Info File: The standard cover sheet template supplied with the FAX server ends with the words "ADDITIONAL INFO". If you want to supply a few lines of text to follow this then you can do so by creating a short text file and then filling in this field with the name of that file. Note that the filename entered here must include a full path; if you can't remember the full path then simply click the "**search...**" button to the right of this field and you will be presented with a standard file selection dialog. If you supply an additional info file then it is permissable to leave the "*File to send*" field blank, in which case the entire FAX transmission will consist of just the cover sheet.

Odyssey FAX Server

Manual Dialing

As briefly mentioned in the documentation for the "Send FAX..." dialog, if you do not supply a telephone number in the appropriate field, then the FAX server assumes that you want to dial manually (a small message will appear to that effect at the point when Odyssey would normally have sent a dial command to the modem). One reason you might have for dialing manually could be because the destination number uses one of the increasingly common "FAX splitter" switches, allowing a FAX and answering machine to share a single line. The voice message from the answering machine is in that case likely to confuse your poor little modem if you allow it to auto-dial, because the modem is going to expect to hear FAX carrier tones as soon as the call is established.

Manual dialing works best if you have a telephone handset plugged into the "phone" socket on your modem, though it can also work (less reliably) if you have the handset plugged into an extension socket on the wall, with your modem on another extension on the same line.

To "manual dial" you simply enter the "Send FAX..." dialog details as described, leaving the telephone number field blank. Hit enter, or click **OK**, and then wait until the software prompts you to begin manual dialing. Now dial the number on the handset, wait until you hear the fax answering tone (ignoring answering machine messages etc), and then press the <Enter> key. At that point the FAX server will take over again, and attempt to force a connection. If the attempt fails then the FAX server will give you the chance to dial again, up to the normal redialing limit specified in the Odyssey setup menu.

☞ Sometimes the dialer will appear to give up after a single attempt: this is because a connection was actually made, and it was an early stage of the FAX protocol negotiation which failed.

If you do not wish to dial manually then press <Esc> at the "manual dial now" dialog which was mentioned above.

Odyssey FAX Server

Receiving a FAX

You don't need to do anything complicated to receive a FAX. Just make sure that the **Enable Receive** box is checked in the Setup|Fax... dialog, and that the Receive directory is properly configured (the directory exists, and is named in the above dialog), and then open the FAX server.

Bear in mind that if you share your modem line with a phone, then it isn't really a good idea to have "Enable Receive" checked all the time, since voice callers are going to be somewhat surprised by the FAX tones which greet them.

Odyssey FAX Server

Viewing a FAX

When the Odyssey FAX server is active, a window entitled "Received FAXes" is visible, containing a list of all the fax files in the fax directory. To view any of these files, simply double click on the appropriate entry in that list.

Doing so causes Odyssey to open a Bitmap Viewer window, in which the FAX is displayed, and from where it can also be printed or deleted.

The **bitmap viewer** displays a FAX page at a resolution of 100 or 200 dpi (controlled by the **Viewer resolution** field of the Setup|Fax... dialog). The 100 dpi setting is half the standard FAX resolution, but allows you to see more of the page on the screen at one time. This may result in complex lettering or graphics appearing to be of lower quality than is really the case in the file; when you print the page however the full FAX resolution is used (in fact it is printed at 300 dpi - a higher resolution).

Odyssey FAX Server

Printing a FAX

You print a FAX document by selecting the **Command|Print Fax...** menu option, or by clicking the "Print FAX" toolbar button.

Either of the above actions causes the Print Fax... dialog to be displayed, allowing you to select which FAX file and which range of pages to print.

Odyssey FAX Server

Exporting a FAX

The .FAX file format used by the Odyssey FAX server module is a specialised multipage image format ideally suited for FAX applications, but which is unique to Odyssey. If you need to convert individual pages into a more portable image format then you can do so by selecting **Command|Export Fax...** from the FAX server menu, or by clicking the "Export FAX" toolbar button, either of which causes the Export Fax... dialog to appear. Odyssey supports exporting FAX documents to **PCX** and **TIFF** formats.

Odyssey FAX Server

ASCII to FAX conversion

As mentioned elsewhere, a FAX transmission sends bitmapped image data from a source to a destination FAX station. Therefore, in order to send a text file, that text must first be converted into a bitmapped form. The Odyssey FAX server does this as the initial step before dialing a number (you will see "Converting..." on the progress display).

Conversion involves taking the ASCII source, extracting the character codes, and painting those characters on the page using a selected font. The FAX server uses one of two different fonts depending on the resolution of the FAX. For normal resolution (200 by 100 dpi), the FAX server uses the **PCFAX100.FNT** file. For "detail" resolution (200 by 200 dpi), the file **PCFAX200.FNT** is used. The .FNT files used by the FAX server are compatible with Windows 3 bitmapped fonts, and so may be replaced by other Windows fonts if you prefer. However, the replacement font you use should have the correct horizontal and vertical resolutions - the FAX server does not attempt to scale the font - and should be fixed width (in fact, variable width fonts are accepted, but tend not to produce good results, because no attempt is made to correct placement of tabular data, columns etc).

A text file may include "macro" fields which are replaced by the conversion process with cover sheet details, or with embedded PCX or TIFF images. A example of how this is done can be found in the file COVER.TEM, which although it is used for a special purpose (to control the layout of the cover sheet), is not special in terms of any macros it uses - these can in fact be used by any text document.

Odyssey FAX Server

Cover Sheet Generation

When the Odyssey FAX server is asked to generate a cover sheet, it simply runs its normal ASCII to FAX conversion procedure on the file **COVER.TEM** (a text file stored in the Odyssey directory), and then transmits the resulting FAX image ahead of the main FAX. COVER.TEM contains several *macro* fields which are expanded into their transmit-time values using the details entered by you into the Send FAX... dialog.

The best way to understand the purpose of COVER.TEM is to look at it with an ASCII viewer/editor (such as the integrated Odyssey text editor), or copy it to your printer. You will see that COVER.TEM is simply a template which defines the format which the FAX Server will use for its automatically generated cover sheets.

Note the fields such as '**%%date___**' which you will see in COVER.TEM. These are the macro fields which are automatically filled in by the routine which converts text files to .FAX files. Where a macro field is padded out with underscores, as in the date example shown, then the conversion process will pad the field out to the same length, but using space characters instead.

COVER.TEM uses all of the macro types provided, and may be used as a guide as to which macros are available, and how they are used.

☞ Notice that although this section has concentrated on COVER.TEM, the same substitutions can actually be performed on any text file which the Odyssey FAX server converts, provided that the text file uses the appropriate macro field. One particularly good use for this feature in a normal text file is to embed graphics, such as your company logo or signature.

Odyssey FAX Server

Embedding Graphics in a FAX

Take particular note of the "**%%logo=...**" macro in COVER.TEM. As its name implies, this macro is primarily designed to allow you to embed your company logo in the outgoing cover sheet, but can in fact be used anywhere in an ASCII document to embed any graphics image - for example you may wish to embed your signature at the end of the FAX.   Some tips on creating a graphics file containing your own company logo or signature are given elsewhere.

The image imported into a FAX can be in either PCX or TIFF format. You should completely specify the image file name in the importing document, **including path**, and in particular you should remember to include the file extension .PCX or .TIF on the %%logo= line so that the FAX server knows which image format to expect.

Accepted Image Formats

As mentioned above, the FAX server allows you to embed PCX or TIFF images in an outgoing FAX. However, you should be aware that in common with other applications which support graphics interchange, this software does not guarantee to read absolutely *any* image which might possibly be encoded using one of these picture formats, however the resolutions which *are* accepted are likely to include all you will ever meet in real life.

In the case of PCX files (PCX5), the FAX server accepts 1, 4, 8 and 24 bit color images. Monochrome and 256 color images are encoded using a single color plane. 4 and 24 bit color images are accepted in either packed or planar forms. "Color" includes grayscale.

In the case of TIFF files, the FAX server accepts the same resolutions as for PCX, and recognises the standard set of tags referred to in the TIFF 6.0 baseline specification. The TIFF image data must be either uncompressed, or packed using "packbits" (RLE) compression. Compression using Huffman (CCITT), LZW, or JPEG is not supported.

Where a monochrome image is imported, it will be copied to the outgoing FAX file as is. In the case of grayscale or color 4/8/24 bit images the destination image will be automatically converted to gray halftones (FAX images are monochrome, so it simply isn't possible to retain the original colors).

The PCX or TIFF reader does not attempt to downscale an image to fit on the page. The image will be clipped if it is either too wide or too deep.

Odyssey FAX Server

How to embed company logos or signatures

To embed any sort of graphic image in a fax, you simply make use of the "%%logo=<filename>" macro in the ASCII text file from which the FAX is generated. This help topic concentrates on how you go about generating bitmapped versions of your company logo and/or personal signature, saved in PCX format at the right resolution, which you can them embed in your FAX.

The best (and easiest) way to create PCX files containing your logo and signature is as follows:-

- Get a sheet of headed paper with your company logo, and sign it about half way down the page.
- Run Odyssey, make sure FAX receive is enabled in the Setup|Fax... dialog, then activate the Odyssey FAX server by selecting the **Window|FAX...** menu item.
- Borrow a friends dedicated FAX machine, and FAX the above page to yourself (ie. to the PC running Odyssey FAX server). **Be sure to use the high resolution option (sometimes called "detail" or "fine") provided by the FAX machine**.
- Use the **Command|Export FAX...** menu option to export the FAX page as a .PCX file. You could also export in TIFF format, but a) TIFF files are significantly larger, and b) many simple graphics programs seem to have more trouble reading perfectly valid TIFF files than they do when reading the simpler PCX format.
- Use one of the many easily available commercial or shareware graphics paint programs (eg, Windows Paintbrush) to clean up the detail in the image. Be aware that some simple paint programs have problems with images as large as this (a full page 200 DPI fax is quite large when decompressed and loaded into memory - nearly half a megabyte). Cut out the logo and signature using the cut or trim tools provided by the graphics package, saving each to a new PCX file in the Odyssey FAX directory. Don't add any margin on the left of the clipped image unless you want the image to be indented by more than the amount Odyssey normally adds. You may wish to further edit each of the clipped images to improve their quality.

You now have separate PCX versions of your company logo and signature, which can be embedded as needed in any ASCII document, simply by typing in a line such as :-

%%logo = C:\WINODY\FAXRECV\MYSIG.PCX

into the document.

You could also, of course, generate these images using a handheld or flatbed scanner if you happen to own one, but your problem there will be scaling the image to the right size at FAX resolution - the method described above produces a poorer quality image to begin with, but guarantees correct scaling. If you do decide to use a scanner then the image should be scanned at 200 dpi.

Odyssey FAX Server

Script Control Interface

The Odyssey FAX Server can be controlled by a script, using the generic DLL interface functions added in Odyssey 2.0. See the Script Language Reference for a description of the LoadDLL(), UnloadDLL() and SendMessage() script commands.

The following sections describe the command messages recognised by the FAX server script interface. A substantial script demonstrating these commands is included on the distribution disk - see **TESTFAX.SCR**, which you should find in your Odyssey directory.

In the examples shown, "fax" is assumed to be the numeric handle returned by a previous LoadDLL("FAXSERV.DLL") call.

☞ For brevity in these examples, the string argument is often shown as a string-literal, when in actual fact it must always be a string-variable (the TESTFAX.SCR declares its own version of SendMessage which accepts literal strings - you must be careful however not to use that routine with a message that returns a result in the string variable!) :-

---

INIT

String argument =>> unused
Numeric argument =>> unused

Forces the modem into FAX mode, and performs internal initialisation of the FAX server. This message is required, and must be the first command sent to the DLL after it is loaded.

Example:
```
SendMessage(fax, "INIT", "", 0);
```

PRINT

String argument =>> "filename pagelist"<R>
Numeric argument =>> unused

Prints selected FAX pages to the printer. The filename and pagelist must be separated by a single space, and there must be no other spaces in the string. The pagelist is formatted as described in the description of the Print FAX... dialog.

Example:
```
SendMessage(fax, "PRINT",  "myfax.fax 1-99", 0);
```

EXPORT

String argument =>> "faxfilename outfilename pagelist"
Numeric argument =>> unused

Exports selected pages from a fax file in either PCX or TIFF format, decided by the extension to the outfilename field. The three fields in the string argument correspond to the three fields on the Export FAX... dialog.

Example:
```
SendMessage(fax, "EXPORT", "my.fax page.TIF 1-99", 0);
```

IMPORT

String argument =>> filename_variable
Numeric argument =>> unused

Converts PCX,TIFF or ASCII text files to FAX format and writes the result to the FAX directory ready for transmission. Note that the string argument must be a variable, since it will be changed by the FAX server to the name of the FAX file just created. You can then use the destination file name as a parameter for a SET-FILETOSEND command.

Example:
```
fnvar := "myfile.txt";
SendMessage(fax, "IMPORT", fnvar, 0);
Write("Dest file was: ",fnvar);
```

ERASE

String argument =>> "filename"
Numeric argument =>> unused

Erases a FAX file from the FAX directory, normally used after the FAX is successfully transmitted. Using this command is more reliable than using Fdelete() because you don't have to know (in the script) where the FAX directory is, plus the FAX server will force a .FAX extension regardless of the extension used in the name you pass - so you could pass it the name of the source file and the correct FAX file will still be erased.

Example:
```
SendMessage(fax, "ERASE", "myfile.txt", 0);
```

SET-LOCALID

String argument =>> "Local ID"
Numeric argument =>> unused

Sets the ID string of the local FAX station. See the description of the equivalent menu option for limitations on the characters which can be used in the ID string. The string must be no longer than 20 chars.

Example:
```
SendMessage(fax, "SET-LOCALID", "Skyro Software Ltd.", 0);
```

SET-COVERSHEET

String argument =>> unused
Numeric argument =>> 0 or 1

Enables or disables the "cover sheet" option. 1 enables the option, 0 disables it.

Example:
```
SendMessage(fax, "SET-COVERSHEET", "", 1);
```

SET-DETAIL

String argument =>> unused
Numeric argument =>> 0 or 1

Enables or disables the "Detail" option. 1 enables the option, 0 disables it.

Example:
```
SendMessage(fax, "SET-DETAIL", "", 1);
```

SET-FAXDIR

String argument =>> "faxdirectory"
Numeric argument =>> unused

Sets the directory to be used by the FAX server for received fax documents, and files which have been converted to FAX format prior to transmission.

Example: `SendMessage(fax, "SET-FAXDIR","c:\winody\fax", 0);`

SET-FILETOSEND

String argument =>> "filename"
Numeric argument =>> unused

Sets the file which will be transmitted by the next "SEND" command. This should be the name of a .FAX file, not a text file; in other words, you should send an IMPORT command first if you want to send an ASCII file.

Example: `SendMessage(fax, "SET-FILETOSEND","my.FAX", 0);`

SET-NUMBER

String argument =>> "phone number"
Numeric argument =>> unused

Sets the number which will be dialed at the next "SEND" command.

Example: `SendMessage(fax, "SET-NUMBER", "0123-45678", 0);`

SET-REFORMAT

String argument =>> unused
Numeric argument =>> 0 or 1

This command sets the "reformat" option normally found in the send-fax dialog. If set to 1, an IMPORT command will reformat text lines which exceed the right margin. If set to 0, long lines will be truncated at the right margin.

Example: `SendMessage(fax, "SET-REFORMAT", "", 1);`

SET-FROM

String argument =>> "your-name"
Numeric argument =>> unused

Sets the value of the "From" string variable used by the cover sheet generator. Not required if cover sheet generation is disabled.

Example: `SendMessage(fax, "SET-FROM","Joe Bloggs", 0);`

SET-TO

String argument =>> "dest-company-name"
Numeric argument =>> unused

Sets the value of the "To" string variable used by the cover sheet generator. Not required if cover sheet generation is disabled.

Example: `SendMessage(fax, "SET-TO","ACME Products Inc.", 0);`

SET-ATTN

String argument =>> "dest-person-name"
Numeric argument =>> unused

Sets the value of the "Attn" string variable used by the cover sheet generator. Not required if cover sheet generation is disabled.

Example: `SendMessage(fax, "SET-ATTN","John Smith", 0);`

SET-OREF

String argument =>> "our-reference"
Numeric argument =>> unused

Sets the value of the "oref" string variable used by the cover sheet generator. Not required if cover sheet generation is disabled.

Example: `SendMessage(fax, "SET-OREF","JB-001", 0);`

SET-YREF

String argument =>> "your-reference"
Numeric argument =>> unused

Sets the value of the "yref" string variable used by the cover sheet generator. Not required if cover sheet generation is disabled.

Example: `SendMessage(fax, "SET-YREF","your ref", 0);`

SET-SUBJECT

String argument =>> "what-this-fax-is-about"
Numeric argument =>> unused

Sets the value of the "subject" string variable used by the cover sheet generator. Not required if cover sheet generation is disabled.

Example: `SendMessage(fax, "SET-SUBJECT","test of fax script", 0);`

SET-ADDINFO

String argument =>> "addinfo.txt file name"
Numeric argument =>> unused

This sets the name of a second file which will be imported and whose text will be appended after the ADDITIONAL INFO line of the cover sheet. This has no effect if cover sheet generation is not enabled. If cover sheet generation is enabled, and an additional info file is set, then you can pass an empty string to the SET-FILETOSEND command if you only want to send the cover sheet.

Example: `SendMessage(fax, "SET-ADDINFO","addinfo.txt", 0);`

SEND

String argument =>> unused
Numeric argument =>> unused

Dials the number given by the SET-NUMBER command, and sends the FAX named by the SET-FILETOSEND command. If the number field was an empty string then the FAX server will give the user the opportunity to dial manually (note that the script will lose control until a user provides a keyboard response - if you don't wish that to happen then always supply a phone number). If cover sheet generation is enabled then the FAX server will create and prepend a cover sheet before sending the FAX. If FILETOSEND is an empty string, and ADDINFO is not, and cover sheet generation is enabled, then *only* the cover sheet will be sent.

Example: `SendMessage(fax, "SEND","", 0);`

RECEIVE

String argument =>> string-variable
Numeric argument =>> unused

This command should only be sent when a ringing signal has been detected (ie. by receiving the RING response from the modem). The FAX server instructs the modem to answer the call, and then it receives the FAX, storing it in the FAX directory. You do NOT pass a name for the received FAX in the string variable; in fact the FAX server always creates a temporary name itself, which is guaranteed to be unique, and passes back that name to you in the string variable.

Example: `SendMessage(fax, "RECEIVE", rxfilename, 0);`

SAVE-SETTINGS

String argument =>> unused
Numeric argument =>> unused

Normally, any changes you make to FAX server variables (using SET-xxxx commands) will be lost when the FAX server module is discarded. Send this command if you want to make the changes permanent.

Example: `SendMessage(fax, "SAVE-SETTINGS", "", 0);`

END-SESSION

String argument =>> unused
Numeric argument =>> unused

This command takes the modem out of FAX mode and restores it to data mode. This command is required, and should be the last command sent to the FAX module immediately before calling UnloadDLL().

Example: `SendMessage(fax, "END-SESSION", "", 0);`

Using Odyssey

The Archive Viewer

The Odyssey **Archive Viewer** window class is designed to provide supplementary services for an Odyssey Directory Viewer window. Particularly, it allows you to view ARC, ZIP and LZH compressed archives within Odyssey, quickly view and extract single files from those archive formats, or extract the entire archive contents to a subdirectory on your hard disk.

Please select one of the following topics for further information. You can also step through these topics in order by selecting the first, and then using the browse buttons to move between topics.

Introducing the Odyssey Archive Viewer.
Viewing a ZIP, ARC or LZH Archive.
Viewing a file stored in an Archive.
Unpacking an Archive.

See also:
ARC
LZH
ZIP

The Odyssey Archive Viewer

Introducing the Odyssey Archive Viewer.

The Odyssey **Archive Viewer** feature is intended to supplement the capabilities of an Odyssey Directory Viewer, so we shall start this introduction by recapping the purpose of the latter Odyssey feature.

You may have already gathered that the **Directory Viewer** is primarily intended to allow you to perform common "housekeeping" tasks on the files which you have downloaded from a BBS. To that end, it displays a list of files which you can then double-click on in order to view them, and thus decide if you wish to keep them around. "Viewing" is mostly done using other Odyssey window classes, such as a text editor window, or a bitmap viewer window.

If you think about the sort of files you are likely to have downloaded from a BBS, some will be text files, some will be bitmaps, but most (probably the greatest majority) will be compressed archive files of some sort; and of those, most will be in the ZIP format. Therefore, if Odyssey is to allow you to view most of the files you have downloaded, allowing you to view compressed archive files would appear to be an important requirement - hence the existance of this **Archive Viewer** window class within Odyssey.

Note that there is no menu option anywhere within Odyssey to open an archive file; just as there is no such option for opening a bitmap file. An **Archive Viewer** window is not opened directly, instead you open a Directory Viewer window, and then double-click on any file you wish to view, and if the selected file is a supported archive format then an Odyssey **Archive Viewer** window is opened.

There are a great number of archive formats in existance, most of which are seldom used, so it simply isn't practical for Odyssey to support every single one of them. Instead, Odyssey currently supports the three most popular DOS archiving formats, which we believe are the ARC, ZIP and LZH formats. We shall probably add support for some popular Unix archiving formats (eg. .zoo, .gz) in the future, for the benefit of Internet surfers.

Odyssey's ability to read the **ARC**, **ZIP** and **LZH** formats is completely internal and integrated, ie. you do NOT need to have, for example, copies of PKARC, PKZIP or LHA etc somewhere on your path in order to view these files.

Please read the following notes about the three archive formats which Odyssey supports :-

ARC: was originally the format used by the utility of the same name published by *System Enhancement Associates* (SEA). This was the de-facto standard BBS archive format for several years, and so you can still find a great many BBS files in this format, even though the ARC format has largely been supplanted by ZIP etc for newer uploads. The SEA ARC utility supported a number of compression methods, starting with none (ie. file stored with no compression), then run-length encoding, and then several variations on the Lempel-Ziv-Welsh (LZW) scheme, each later version of the scheme slightly increasing in sophistication (eg. moving from fixed length codes to variable length codes). Phil Katz of PKWare also produced an ARC compatible utility called PKARC, and also added a new "Squashed" compression method - another LZW variant. The Odyssey ARC format reader handles all ARC and PKARC compression methods.

ZIP: this is the current de-facto standard BBS archive format, at least for DOS files. ZIP improves on ARC/PKARC by providing faster decompression, better compression ratios, and the ability to store complete directory trees in the archive (instead of the simple list of unqualified file names used by ARC). Although ZIP supports a number of different compression methods, the most important algorithms are derived from the "Sliding Dictionary" Lempel-Ziv schemes (ie. where a string which has previously occurred in the file is replaced by a code telling the reader the position and length of the previous occurrence). ZIP also uses a secondary static Huffman encoding to further compress the character, length and position codes. Later ZIP algorithms find efficient ways to handle large (32k) dictionary sizes. The popularity of ZIP was also boosted by the fact that the author (Phil Katz of PKWare again), published details of the algorithms used, which allowed programmers on other platforms (eg. Unix) to handle this format. The Odyssey ZIP reader handles all compression methods supported by PKZIP v2.04g and earlier (ie. up to and including the "Deflate" compression method).

LZH: the format used by the LHARC (later called LHA) utility, LZH has attained a respectable following among BBS users. Initially this was because, while LHARC was significantly slower at compression than PKZIP: a) decompression wasn't *much* slower, b) compression ratios were higher, c) LHARC was free, d) the source was also available for free, which made it easy to port the format to non-DOS platforms. LHARC also had the ability to create self extracting archives with a surprisingly small file size overhead. Subsequent PKZIP versions pretty much matched LHA on compression ratio, though the other attractions of LHA remained. The compression algorithms used by LZH are variants on the "sliding window LZ with secondary Huffman encoding" theme, much like PKZIP (the LZH word itself is an acronym for "Lempel-Ziv-Huffman"). Unfortunately, the only available documentation on the LZH algorithms resides in the aforementioned freely available sources, much of which consists (at least for later methods) of sparsely commented assembly language modules. For this reason Odyssey does *NOT* support all compression methods supported by LHA. Instead, we support those methods which we have been able to work out, specifically the -lh0- (ie. the "stored" or "uncompressed") method, the "-lh1-" method which was used by LHARC 1.13, and the "-lh5-" format used by LHA 2.xx. One reason that we have not implemented other methods is because we have been unable to find .LZH archives containing files compressed with these other algorithms, and thus would have been unable to test any code we developed. The latter point hopefully means that you won't find any such files either, ie. the lack of support for these methods will not inconvenience you in the least. LZH, like ZIP, has the ability to store entire directory trees in the archive.

ZIP and **LZH** formats support encryption of files with a text password, however Odyssey can only decrypt files which were encrypted and use the ZIP format.

The Odyssey Archive Viewer

Viewing a ZIP, ARC or LZH Archive.

To open an Odyssey **Archive Viewer** window, simply double-click on a ZIP, ARC or LZH file shown in the file list of a Directory Viewer file list panel. The Odyssey **Archive Viewer** window consists of a standard Windows multiple selection listbox, each entry providing details of one of the files in the selected archive. This list looks pretty much like what you would get if you typed "pkzip -v archive" from DOS (or the equivalent, for other archive formats). A scrollbar is provided to allow you to scroll up and down the list of compressed files.

Odyssey provides several options in the Archive Viewer **Command** menu which allows you to manipulate the list of files shown, for example you can specify the type of files to be listed (by means of a wildcard file specification), as well as the sort order of the file list - the sorting options are very much like those provided by the Directory Viewer feature, with the addition of a sort order of *None*, which means that files are listed in the order they are stored.

The Odyssey Archive Viewer

Viewing a file stored in an Archive.

This feature allows you to view files in an archive without having to decompress the entire archive first.

Once the Odyssey **Archive Viewer** is opened, it is a simple matter to view any single file from that archive: simply double-click on the entry for that file in the list displayed. Just as in the case of the Odyssey Directory Viewer window, double-clicking on a listed file causes Odyssey to open up a secondary viewing window which is specialised towards the type of file being viewed. For example, if you click on a BMP, GIF, JPG or FAX file then a bitmap viewer is opened, if you click on a windows **.HLP** file then the Windows Help system is used to animate that help file. You can even double-click on a ZIP, ARC or LZH file stored in the first archive, and a second **Archive Viewer** window opens, and you can then proceed to view the files in that archive also!

In each of the cases mentioned, Odyssey implements the feature by decompressing the selected file into a temporary file, which it then creates a viewing window for. This temporary file is normally deleted when the associated viewing window closes. ***Note however*** that if the selected file was a Windows .HLP file or an .EXE file then the appropriate "viewer" is actually a separate program, which may not necessarily terminate before Odyssey does; and Odyssey obviously cannot delete that temporary file if Odyssey is no longer running. In short, to avoid cluttering your disk you should always make it your practice to close any viewing windows before you shut down Odyssey itself.

The Odyssey Archive Viewer

Unpacking an Archive.

If you decide that you want to unpack the contents of a ZIP, ARC or LZH file, you simply select the **Command|Extract files...** option from the **Archive Viewer** menu, or click on the **Extract files** toolbar button.

You will be presented with the Extract files... dialog, in which you give details such as the target directory into which the files should be unpacked (if the archive contains a directory tree, this could become the root directory of that directory tree). You also select whether to unpack all files from the archive (see note), or whether to unpack only those files which are selected in the multiple selection listbox used by the **Archive Viewer** window. Finally, you also specify whether or not Odyssey should overwrite existing files, and also (optionally) set a password for decryption, if the files are encrypted.

☞ Unpacking "All files" means unpacking all files shown in the listbox, which may not necessarily be the same as all files stored in the archive. For example, if you tell the **Archive Viewer** to list all files matching the "***.C**" pattern, and then select "Unpack all files" in the Extract files dialog, then the archive viewer will unpack all files in the archive which match the "***.C**" pattern. If you really intend to unpack every file stored in the archive you should make sure that the file type shown in square brackets in the Archive Viewer window caption is either "**[*]**" or "**[*.*]**" (select **Command|Show files of type...** or click the **Show files of type...** toolbar button to change the pattern).

Using Odyssey

Odyssey Dialogs

Press **F1** while any dialog is displayed, for context sensitive help specific to that dialog.

Odyssey Dialogs

Dialing Directory Dialog

You reach the dialing directory dialog by selecting **Command|Dial...** from the main menu, or by clicking the "phone" toolbar button, or by pressing **ALT+N**.The dialing directory is split into two parts - a list of services in a large panel on the left, and an array of action buttons on the right.

The service list is the large panel on the left hand side which occupies most of the dialing directory dialog. Each directory entry occupies one screen line, a format which makes it convenient for you to quickly scan the directory, and for *point and shoot* dialing. The information displayed is however only a brief summary of the much more detailed information kept about each entry. To see and edit the extra detail you should highlight the directory item that interests you and press the **Edit** action button. The action panel on the right contains a number of other buttons which will be described in the topics below.

Adding New Directory Entries
Editing Directory Entries
Dialing a number
Resetting Call Statistics
What Odyssey does when it Dials

See also:
The Edit Dialing Directory Entry dialog.

Dialing Directory Dialog

Adding New Directory Entries

Adding a new directory entry consists of selecting a slot for the new entry, and then editing it to give the correct information about the new service.

You can create a new entry by using the **Insert Line** button to insert a blank entry at the current position, which you would then edit. Alternatively, if the new entry will be similar to an existing one, you can cut the other entry with the **Cut** button or the DEL key, then paste it back twice using the **Paste** button or the INS key. That gives you two identical entries, one of which you can modify. A final alternative is simply to move to one of the free slots near the end of the dialing directory and edit that.

Dialing Directory Dialog

Editing Directory Entries

Edit a directory entry by first highlighting it using the arrow keys, then click the **Edit** button. This causes the Dialing Directory Edit dialog to appear which displays the current values of all the details Odyssey keeps for that entry, and allows you to change them.

Dialing Directory Dialog

Dialing a Number

There are two ways of dialing a number from the dialing directory.

Point and shoot dialing is the simplest. To do that you use the mouse to select the entry for the service you want to call, then click the **Dial** action button. Alternatively, you can simply double click on the service entry to dial. In either case, Odyssey will then dial the number, and if necessary redial, until a connection is established or it has made the maximum number of dial attempts defined in the Setup/Modem... menu.

Tagged Dialing is more flexible/powerful, but takes a little more setting up. You first tag any entry by selecting it and then clicking the **Tag Single** button. Alternatively, you can simply click the entry with the right mouse button to both select it and toggle the tag attribute. Once the entry is tagged you should then click the **Dial Tagged** action button (or press **ALT+T**), which causes Odyssey to dial the number you just tagged. The main point that hasn't been mentioned yet is that you may tag as many numbers as you like, and when you initiate tagged dialing, Odyssey will dial each number in turn until it makes a connection.

Odyssey offers a further action button - **Tag by Key** - which can be used to set tags on all entries with a particular key. Pressing **ALT-K** on the keyboard has the same effect. You can clear all the tags from tagged items by clicking the **Clear Tags** action button.

Dialing Directory Dialog

Resetting Call Statistics

Every time you complete a call, Odyssey updates the appropriate dialing directory entry for the service: incrementing the count of the number of calls, and also recording the time, date and duration of the call.

If you make a copy of a directory entry using the cut and paste buttons then these statistics are copied as well. If you want to reset the statistics for the new entry you can do so by selecting the entry and clicking the **Reset Stats** action button.

Dialing Directory Dialog

What Odyssey does when it Dials

The following is the procedure which Odyssey follows when dialing any number, or any queue of numbers. Understanding this procedure may allow you to make more efficient use of the Odyssey dialing feature, especially in conjunction with a script. During the dialing process Odyssey makes use of information defined in the dialing directory entry (or entries), and also from the modem configuration available in the Setup/Modem... dialogs.

1. The first thing that the dialer does is make up a list (queue) of the different numbers it needs to dial. It displays a window showing as much of this queue as possible. If you used *point and shoot dialing* then the queue will contain only one number. If you asked Odyssey to dial all "tagged" numbers then the queue may contain one or more numbers. If you used the "Continue dialing" option of the Command menu then the dial queue is carried over from a previous dialing procedure. Odyssey now marks the first number in the queue as the "next number to dial" and moves to step 2, which is the start of the dialing loop.

2. Odyssey selects the number from the dial queue which is marked as the "next number to dial", highlights this queue entry in the dial progress window, and then examines what modem configuration changes need to be made for this number.

3. If modem reconfiguration is necessary (this is the first dial attempt, or the necessary config is different from the last dial attempt, or "Always init" is enabled in the Setup|Modem configuration section), then Odyssey selects and transmits an init string to the modem. The selected init string is either the "Dialer init string", or one of the error correction strings, depending on the settings in the dialing directory and **Setup/Modem** menu. Odyssey looks for a reply from the modem to the init string. If it gets one we proceed to the next step, otherwise an error message is generated saying "Modem does not respond to init", and we abort here.

4. The Odyssey dialer now changes the comm port baud rate and parity settings to that specified in the current directory entry, transmits the dial command to the modem, and waits for a response. See the description of the Setup|Modem dialogs for details of how the dialing command is constructed.

5. The dialer starts counting down from the timeout figure allowed for a connection (this value is configured in the setup menu). If the modem fails to give a recognised response in time, or if it responds with a connect failure message then we go to step 6, otherwise we go to step 7.

6. This step is reached if the last dial attempt failed. If there is more than one number in the dial queue then we mark the next number as the "next number to dial". If we have already made the maximum number of dial attempts on this number then we issue an error message and abort, otherwise we pause for the number of seconds specified in the "Delay before redial" setup option, then return to step 2.

7. We get here if a connection has successfully been established to a dialed number. The dialer now performs some post-connection configuration:

    - The successful number is removed from the dial queue. This leaves the remaining numbers (if any) in the queue, to be processed by the Command|Continue Dialing menu command.

    - If the "Baud Rate Detection" option is enabled, then the dialer examines the full connect message, and changes the comm port baud rate to the speed which the modem reported. If Software MNP is enabled, and error correction is requested, then the software MNP engine is alerted, and starts up. The appropriate terminal emulation is loaded, along with the associated keyboard file (if any). If text logging is enabled then the appropriate logging mode is initiated.

- If a script is attached to the directory entry (ie. if a script exists named <KEY>.SCR), then that script is started up, and control is passed to it. Otherwise, control is passed to the terminal keyboard.

The dialing procedure is now complete. However, Odyssey takes certain additional steps when the call is completed.

1. If text logging was started by the dialer, then the log file is closed when the call is completed. Odyssey will *not* automatically close a log file which was not opened by the dialer, ie. one which was opened manually or by a script. In the latter case it is up to the user or script which opened the log file to also close it.

2. Likewise, if a terminal emulation was loaded by the dialer, then Odyssey will return to the default emulation when the call is completed, if the default is different from the selected terminal.

3. If "Event logging" is enabled in the setup menu, then the dialer will record the service name and duration of the call just completed.

Odyssey Dialogs

Editing a Dialing Directory Entry

You edit a dialing directory entry by first selecting it with the mouse, and then clicking the **Edit** button. This causes the **Dialing Directory Edit** dialog to appear which displays the current values of all the details Odyssey keeps for that entry, and allows you to change them. The fields in this dialog are described below.

Key
Service
Phone number
CR xx translation
Strip parity bits
Baud rate
Terminal emulation
Parity
Text Logging
Error Correction

See also: The Dialing Directory dialog.

Key: The key field is used to identify a dialing directory entry. It is up to you whether the keys should be unique, or whether several related entries should all have the same key. The key is used by Odyssey when it wants to retrieve files related to a directory entry, such as a script file, whose name is derived from the key. You use the key yourself if you want to tag several directory entries by key. We recommend that you do give every directory entry a key, which must be no longer than eight characters. This is a string field; to edit you just select the field and start typing.

Service: The Service Name is exactly what it sounds - a readable and concise name for the service. Odyssey does not use this name internally except to tell you who it is calling when it is dialing the number. You may therefore have anything in this field that you like. This is a string field; to edit you just select the field and start typing.

Phone number: This field should contain the telephone number for the service. The text in this field may include digits, hyphens and spaces. It doesn't matter if your particular choice of modem takes a dislike to the latter two characters in a dial command, because Odyssey in any case strips them out when constructing the command.

You may also embed **@x** and **%x** prefix codes in the telephone number, where x is a letter from A to J selecting one of the number prefixes defined in the setup menu. @ prefixes are expanded where they occur in the number. % prefixes are prepended to the start of the eventual dial command. These prefixes are configured in the setup menu - see the description of the Setup|Modem dialog. This is a string field; to edit you just select the field and start typing.

CR in -> CRLF: Odyssey can, if you wish, translate all incoming carriage return characters into CRLF pairs. If lines overwrite each other as they appear on your terminal then you should enable this option. The field is disabled by default, which leaves incoming carriage returns untouched.

CR out - CRLF: Odyssey can, if this field is selected, translate all outgoing carriage return characters into CRLF pairs. This field is disabled by default, which leaves outgoing carriage returns untouched.

Strip parity bits: If you normally connect to a long distance host using a low cost X.25 network (eg. Tymnet or DialPlus), you may sometimes find that the network expects parity checking while the host itself does not. If you enable even parity to satisfy the network then file transfers with the host system fail to work properly. On the other hand, if you leave parity checking disabled then you get strange semi-graphic characters appearing on your terminal until you manage to log onto the host. This may even break your login script. The solution is to leave parity set to *None* and instead enable parity bit stripping. When enabled this setting tells Odyssey to mask off the high (parity) bit of every character it displays on the terminal, thus you see no strange characters appearing. This will not however prevent file transfers from working correctly, since parity bit stripping is only applied to characters displayed in the terminal window. This checkbox is enabled by default.

Baud rate (bps): This field tells Odyssey what baud rate the service expects. The default baud rate is 2400 bps. This is a listbox field, so to change the baud rate field you don't type the number - instead you pull down the list of baud rates by clicking on the "down arrow" icon, then click on the appropriate entry from the list of supported baud rates which is presented to you.

You will notice that the baud rate menu includes a **Default** setting. Choosing this option means that the dialer will use whatever baud rate was last set in the Setup|Comms... submenu, or the baud rate at startup, if no change has been made in the current session. This is useful for users who own two or more modems, capable of different speeds, since it means that you don't need to edit every directory entry each time you switch between modems - you can just change the baud rate in one place, and the change will affect all relevant dialing directory entries.

Terminal Emulation: The terminal emulation field tells Odyssey what terminal type the service expects. To change the terminal emulation field you simply pull down the list of supported emulations and select the one you need. If you select ANSI emulation then Odyssey will automatically disable parity bit stripping, since that emulation is usually expected to be capable of displaying IBM PC (Windows OEM character set) characters with codes greater than 127. The default terminal type is **TTY**.

Parity: The parity field tells Odyssey what type of parity checking the service expects. This section of the dialog is a group of radio buttons. You change the parity type by using the mouse to select the radio button which represents the parity type you need. Notice that Odyssey does not provide *data bits* or *stop bits* fields in this dialog. This is because Odyssey assumes eight data bits any time you select "*parity: none*", and it assumes seven data bits otherwise. Similar assumptions are made about stop bits. There is no need for you to concern yourself with whether or not Odysseys assumptions are correct - in the real world these assumptions are always correct, hence there is no need for you to waste time entering data into unnecessary fields. The default for this section is eight data bits, no parity.

Odyssey Dialogs

Edit Dialing Directory Entry

Text Logging: Text logging (or text capture) allows you to record the characters received by Odyssey during an online session. Odyssey provides various text logging modes which you control by clicking the appropriate radio button or check box in the text logging group. Available modes are:-

Off	- No text logging is performed (the default).
Create new file	- Text logging is enabled, any existing log file for this service is erased, and data for this session is written to a new file.
Append existing file	- Text logging is enabled, any existing file is opened, and data for the latest call is appended to the old data.

The *Raw Logging* checkbox controls what Odyssey does with terminal emulation control sequences (sometimes called escape sequences), if text logging is enabled and a non-TTY emulation is used. If Raw Logging is selected then Odyssey records the escape sequences directly in the log file. If Raw Logging is not selected then Odyssey attempts to strip these escape sequences out, in order to produce a clean ASCII text file. Note that production of a usable flat ASCII file cannot be guaranteed if the host used complex cursor movements or erase operations to update the display. The ASCII conversion feature works best when the host paints the display in a TTY-like top to bottom scrolling fashion, and simply uses terminal control sequences to change colors, fonts etc.

If Text Logging is enabled you may also provide a name for the log file using the **file:** field in the text logging section of the dialog. If you don't, then Odyssey will use a default name of ODYSSEY.LOG. Only the main part of the log file name is required, eg. "LOGFILE". Odyssey will add the .LOG extension itself, and the file will be stored in the download directory (configured in the Setup|General... dialog), or the current directory if that setup field is blank. This is a string field; to edit you just select the field and start typing.

Odyssey Dialogs

Edit Dialing Directory Entry

Error correction: This field tells Odyssey whether or not the service described by this directory entry supports error correction. This field should be filled in even if the error correction is to be handled by your modem; this section does *not* apply solely to users of Odysseys software MNP feature.

This dialog section has four radio buttons, one of which should be selected to enable the appropriate error correction mode. One of the four settings is applicable only to software MNP and is grayed out if software MNP is disabled in setup. The settings provided are:-

Disable - This service does not support error correction. In this case Odyssey will transmit the "disable error correction" string to the modem prior to dialing the number, if the appropriate modem control string is not blank.

On, no compression - The service supports error correction, but you do not want to use data compression, for example because you want to transfer .ZIP files with an MNP5 modem. In this case Odyssey will transmit the "Enable Error Correction ONLY" string to the modem prior to dialing the number, or will use software MNP4.

On, with compression - The service supports error correction. Odyssey will transmit the "Enable Error Correction and Data Compression" string to the modem prior to dialing the number, or will use software MNP5.

Forced - The service supports MNP, and you require Odyssey to abandon the attempt to establish a connection if MNP negotiation fails. This setting is available only when using software MNP error correction.

The default is *Disable*.

Odyssey Dialogs

Find String Dialog

The **Find String** dialog is displayed when you type **Ctrl+Q F** in one of the text editors, or when you select the Find... option from the editor Command menu, or when you click the "magnifying glass" editor toolbar button. The dialog contains the following fields:-

Find: is the string you want to find in the current document. The first time this dialog is displayed in any session, this field will be blank. Thereafter, it will default to the string entered last time.

The **Scope** panel controls which parts of the current file is searched for the string. The panel contains three radio buttons, one of which should be selected. The radio button options are :-

> *Local* - Searching takes place from the cursor position onwards, but only strings found inside the currently marked block are reported. If no block is currently marked then no strings will be found.
>
> *Global* - Searching begins at top of file, and any occurrence of the string is reported (marked blocks have no relevance to the search). In other words, this function always finds the first matching string in the file, regardless of where the cursor was when the search started.
>
> *From cursor* - Searching takes place in the region from the current cursor position to the end of the file. The first matching string in that region is found, if it exists.

The **Direction** panel controls the direction of the search. *Forwards* searches forward in the file, *Backwards* causes the search to progress backwards.

The **Options** panel provides miscellaneous search options:-

> *Case sensitive* - If checked, then case (eg. ThisWord vs THISWORD) is significant in deciding whether a given target string matches the search parameter. If not checked then case differences are ignored.
>
> *Whole words only* - If checked, then target strings will only be considered as matching the search string if the target string is not part of a larger word. Eg. a search for "man" will not stop on "mandrake".

Click on **OK** to begin the search, click on **Cancel** to abort the search. The **Help** button (or F1) displays this help topic.

Odyssey Dialogs

Find and Replace Dialog

The **Find and Replace** dialog is displayed when you type **Ctrl+Q A** in one of the text editors, or when you select the Replace... option from the editor Command menu, or when you click the "magnifying glass**...**" editor toolbar button. The dialog contains the following fields:-

Find: is the string you want to find in the current document. The first time this dialog is displayed in any session, this field will be blank. Thereafter, it will default to the string entered last time.

Replace with: is the string you want to replace the search string with. The first time this dialog is displayed in any session, this field will be blank. Thereafter, it will default to the string entered in this field last time.

The **Scope** panel controls which parts of the current file is searched for the string. The panel contains three radio buttons, one of which should be selected. The radio button options are :-

Local - Searching takes place from the cursor position onwards, but only strings found inside the currently marked block are reported. If no block is currently marked then no strings will be found.

Global - Searching begins at top of file, and any occurrence of the string is reported (marked blocks have no relevance to the search). In other words, this function always finds the first matching string in the file, regardless of where the cursor was when the search started.

From cursor - Searching takes place in the region from the current cursor position to the end of the file. The first matching string in that region is found, if it exists.

The **Direction** panel controls the direction of the search. *Forwards* searches forward in the file, *Backwards* causes the search to progress backwards.

The **Options** panel provides miscellaneous search options:-

Case sensitive - If checked, then case (eg. ThisWord vs THISWORD) is significant in deciding whether a given target string matches the search parameter. If not checked then case differences are ignored.

Whole words only - If checked, then target strings will only be considered as matching the search string if the target string is not part of a larger word. Eg. a search for "man" will not stop on "mandrake".

Ask before replacing - If checked, the editor will display a Yes/No dialog requesting confirmation of the operation before any occurrence of the target string is replaced.

Replace all - If checked, all occurrences of the target string in the region defined by the Scope panel are replaced. If not checked, only the first occurrence is replaced.

Click on **OK** to begin the search and replace operation, or click on **Cancel** to abort. The **Help** button (or F1) displays this help topic.

Odyssey Dialogs

Print Fax Dialog

The **Print FAX...** dialog is displayed when you select **Command|Print Fax...** from the FAX server menu, or when you click the FAX server "Print FAX" toolbar button.

This dialog allows you to print selected or all pages from a FAX file.

Fax file name: is the name of the FAX file to print. If you can't remember the name of the file then a "**search...**" button is provided, which brings up a standard file selector dialog.

Page(s) to print: allows you to enter a list or range of page numbers (for example "1,2,5-99") - only those pages are printed. It is not an error for you to enter a page range greater than that which actually exists in the file (eg. printing with a page range of "1-99" guarantees to print all pages).

Click on **OK** to accept the values you have entered, click on **Cancel** to abandon the printing procedure. Clicking on **Help** displays this help topic.

Odyssey Dialogs

Export Fax Dialog

The **Export FAX...** dialog is displayed when you select the **Command|Export Fax...** item from the FAX server menu, or when you click the FAX server "Export FAX" toolbar button.

The .FAX format used by the FAX server module is a specialised multipage image format ideally suited for FAX applications, but which is unique to the Odyssey FAX server. If you need to convert individual pages into a more portable image format then you can do so using this dialog. PCX and TIFF formats are supported as export formats.

The dialog contains the following fields:

FAX file name: is the name of the FAX file which contains the pages you wish to convert to PCX or TIFF format. If you cannot remember the FAX file name then you can click the "**search...**" button, which displays a standard file selector dialog.

PCX/TIF name: is the name which will be given to the destination file(s). Note that the name you enter here will be used to derive more than one file name. For example "PAGE.PCX" would generate PAGE00.PCX, PAGE01.PCX etc, depending on the page range you choose to export.

Pages to export: allows you to select which list or range of pages in the FAX file are to be exported, eg. "1,2,5-99". It is not an error for you to enter a page range greater than that which actually exists in the file (ie. exporting with a page range of "1-99" guarantees to export all pages).

Click on **OK** to accept the field values you have entered, or click on **Cancel** to abandon the export procedure. Clicking on **Help** displays this help topic.

Odyssey Dialogs

File Selector Dialog

The **File Selector Dialog** is used when opening files, saving files with a new name, and in other situations when a file name is required. The dialog allows you to enter the file name and path for the operation about to be performed.

Notice the various components of the dialog: the "**file name**" field at the top, the "**file type**" listbox, the "**files**" listbox to the bottom left, and the "**directories**" listbox to the bottom right.

The **File Name:** field is where you enter the "tail" part of the file name. If you enter an ambiguous name here (ie, containing wildcards), then the "files" and "directories" listboxes will be updated to match the new specifications. If the name is not ambiguous then you have made your selection.

The **File Types** listbox allows you to choose the type of file you are interested in - this is a quick way of changing the default file extensions for the file list.

You can click on a file in the "files" listbox to select a file, then you click on **OK** to complete the selection. Alternatively, double clicking on the file listing both selects and completes the selection in a single operation. You can also double click on an entry in the "directories" listbox to change directories.

Click on **OK** to accept the current selection, **Cancel** to close the dialog without making a selection, or **Help** to view this help topic.

Odyssey Dialogs

Show files of type...

The **Show files of type...** dialog is part of the Odyssey Directory Viewer subsystem, and appeared because you clicked the "New file type" toolbar button, or because you selected the **File|Show files of type...** menu item.

This dialog is used to change the wildcard search pattern which controls which files will be listed in the file list panel of a directory viewer window. The dialog consists of one main field, which is where you type in the new file type, for example MYFILE*.*. This field is actually a listbox - if you pull the listbox down you will be presented with a list of common search masks, if the one you want is on this list that selecting that item will be faster than typing it yourself.

Click on **OK** to accept the new search pattern, or click on **Cancel** to retain the existing search pattern. Clicking the **Help** button displays this help topic.

Odyssey Dialogs

Select...

The **Select...** dialog is part of the Odyssey Directory Viewer subsystem, and appeared because you selected the **File|Select...** menu item.

This dialog allows you to select or deselect files by name, rather than clicking on the files with the mouse.

In the **File type** field you enter a file specification (eg. *.TXT) which matches the names of files you would like to select or deselect. This field is actually a listbox, if you open the listbox you will be presented with a list of common file specifications. If the specification you want is on the list then selecting that listbox item will be faster than typing in the specification yourself.

The next two fields on the dialog determine whether you are selecting or deselecting files. Click the **Select** radio button if you want to select files, or click the **Deselect** radio button if you want to deselect files.

Click on **OK** to begin selecting or deselecting files, or click on **Cancel** to leave the selection states as they are. Clicking on **Help** displays this help topic.

Odyssey Dialogs

Extract files...

The **Extract files...** dialog is displayed when you select **Command|Extract files...** from the Archive Viewer menu, or when you click on the "*Extract files...*" toolbar button. Both the menu and the relevant toolbar are only visible when an **Archive Viewer** window is active. The dialog is used to supply Odyssey with the details it needs in order to unpack the archived files. The fields on this dialog are as follows:-

Destination path: is the directory into which the compressed files will be copied as they are unpacked. Contrary to most other places within Odyssey, you *may* name a directory which does not yet exist - Odyssey will create the directory for you in that case. If the archive itself contains a directory tree, and you have selected the "*Restore directories*" option also on this dialog, then the destination directory you enter in this field will become the root directory for the directory tree stored in the archive (ie. Odyssey will create all subdirectories in addition to the named target directory).

The **Files to extract:** panel contains two radio buttons and one checkbox. The two radio buttons are (as always) mutually exclusive and are described as follows :-

- ***Extract all files*** - if this radio button is selected, then all files listed in the Archive Viewer listbox will be unpacked, regardless of their selection state.
- ***Extract selected files only*** - if this radio button is selected, then only those files which are selected (ie. highlighted) in the Archive Viewer listbox will be unpacked. This radio button will be disabled if there are no files currently selected.

Also on the "Files to extract" panel, the "**Restore directories**" checkbox controls whether or not Odyssey recreates the directory structure (if any) stored in the archive. If this option is enabled then the archived directory tree will be restored, otherwise the path part of the archived file names will be ignore and the uncompressed files will all be copied into a single subdirectory, ie. the directory named in the *Destination path* field.

The radio buttons in the "Overwrite existing files" panel control what happens when Odyssey discovers that a file it would like to unpack from an archive has a name and path which matches that of an existing file. There are three radio buttons, which are, as usual, mutually exclusive. The meaning of the radio button options are :-

- ***Always*** - Odyssey will overwrite any existing files without asking - USE WITH CARE!
- ***If confirmed by user*** - Odyssey will wimp out and ask you what to do when it finds that a file already exists. If a file does already exist then Odyssey displays a dialog which gives you the choice of overwriting it, or leaving the existing file (skipping the archived version), or cancelling the unarchiving task altogether.
- ***If extracted file is newer*** - in the event of a name clash, Odyssey compares the date stamps of the two files. If the archived file is newer, Odyssey overwrites the existing file (without asking), otherwise it leaves the existing file intact, skipping to the next file in the archive.

Password for decryption: - This option applies to encrypted ZIPped files only. If a file is encrypted then unpacking is only possible if you know the password which was used during the encryption. If you do know that password then you should enter it here. This field is ignored for non-ZIP archive formats, and for any archive which does not contain encrypted files.

Click the **OK** button to accept what you have entered, and to begin the archive unpacking process. Click the **Cancel** button to abandon the unpacking process. Clicking the **Help** button causes this help topic to appear.

Odyssey Help

Using Other Odyssey Features

Please select one of the topics below for information on a specific Odyssey feature.

Odyssey Features

File Transfer

File Transfer is the general name given to any mechanism which provides a solution to the problem of transferring data between computers. We are mainly interested here in serial file transfer, ie. transferring files by means of the serial port. A serial file transfer system avoids the question of the compatibility of disk drives and computers, and instead depends on both computers having compatible serial ports, which is much more likely, since RS232 is almost universal in the computing world.

Serial file transfers can also take place over a telephone connection, which means that the other computer could be on the other side of the world, or just in the next room, and yet you would use the same procedure for copying the file. One problem with using the telephone network is that files could potentially be damaged by line noise, and so any file transfer procedure we use should hopefully be able to prevent this.

Ignoring all but serial file transfer, we can summarise by saying that file transfer is the task of transferring a document held on one computer system to another computer system, optionally verifying that an accurate copy was made, and optionally converting the transferred document into a form suitable for use on the target computer system.

The topics below contain further information on file transfer using Odyssey :-

Odyssey File Transfer

Uploads and Downloads

A common area of confusion with file transfer is the meaning of the terms **Upload** and **Download**. These are standard jargon terms for, respectively, sending and receiving data. However, it is not always obvious whose point of view is implied. For example, if upload means send, does that mean that you are sending to the host, or that the host is sending to you?

In fact, the terms are always used from your point of view. An **Upload** will always mean sending from your machine to a remote modem, and **Download** always means from the remote machine to your PC.

When you want to receive a file from a remote host, and the host offers upload and download menu options or commands, you should remember that the host is using these terms from *your* point of view - so you would select download to receive a file from the host, or upload to send a file to the host.

Odyssey File Transfer

What is a File Transfer Protocol?

According to the dictionary definition, a **protocol** is "*the formal etiquette and procedure for state and diplomatic ceremonies*".

The use of that word in computing is an analogy drawn on the above definition. When two computer systems wish to exchange information, they cannot simply blast data at each other, since the receiving computer will have no idea what the data is, or what to do with it. Each must instead use a formal method of communication which has previously been agreed by both sides - this agreed system of communication is what we call a **Protocol**. Since the purpose of the communication we are interested in is the transfer of a file, the complete jargon term is a **File Transfer Protocol**.

A file transfer protocol always has a few basic ground rules, such as the structure of a basic unit of information - in jargon terms this unit is often called a **Packet**, although other names are sometimes used, such as block or frame. Once a packet format has been agreed, a file transfer can be accomplished by reading the file data, breaking it into packet sized chunks, and transmitting those packets to the remote computer, which copies the data inside the packets to a newly created file. Most information packets sent during a file transfer will normally contain file data, but more sophisticated protocols may also use packets to pass other information about the file, such as what it is called, how large it is, the date and time it was created, and so on.

Packets will normally include other information which helps the protocol to verify that information has been received in the correct order, and has not been damaged by line noise. The protocol can protect itself against line noise by adding a checksum to the data transmitted with the packet. A checksum is simply an arithmetic sum of (at least) all the data bytes in the packet. When the receiving protocol has reconstructed the packet it sums the data bytes again, and the answer it gets should match the one received with the data; if the two results do not match then a message must be passed to the sending protocol asking it to send the packet again.

A checksum is not the only method which can be used to verify data. More sophisticated protocols use a similar mechanism called a CRC check. This check type is rather too complicated to describe fully here, however we can summarise by saying that it involves forming all the data in a packet into one huge number, dividing that by a constant, and transferring the remainder resulting from the division (the remainder is the CRC).

This is all rather complex, but the important point to bear in mind about CRC checking is that it is usually more reliable than a simple checksum, since it is quite within the bounds of possibility that an incorrect packet could actually produce the correct data sum by accident. For example, suppose the data consists of eight zero bytes - the sum is naturally zero. Now suppose that line noise caused one byte to be lost, the sum is still zero, so a checksum would miss this error, whereas a properly implemented CRC check would not.

Odyssey File Transfer

Why are there so many file transfer protocols?

Odyssey supports eight different file transfer protocols (including ASCII), a small fraction of the number it could have had, so the question of why so many are needed is a natural one.

There are several answers to this question. On the one hand, there are many programmers out there who fancy that they can design a better protocol than the next guy, some are even justified in that belief, and so their creation becomes established on a popular bulletin board and spreads from there.

More importantly, new protocols arise as the comms market adapts to new requirements. The old mainframe world had little use for file transfer protocols, since all the data was held on one central computer. The people initially interested in file transfer were the early micro users, and so one of their number invented the Xmodem protocol. The principle was quick and dirty, but it did its job. As the micro world threatened to swamp them, the mainframe people in large academic institutions started to take notice (better stamp some order on these infidels), and the need arose for protocols which could exchange data between a variety of different PCs and the mainframe. Xmodem could not be used, because Xmodem makes assumptions about the type of connection it has, and so the Kermit protocol was born. Around the same time users were getting tired of the time Xmodem was taking to transfer large files, especially over international satellite links - and so came the Ymodem protocols, followed by Zmodem.

The Compuserve B+ protocol is something of a special case, since its development was not user led like other protocols mentioned above, but was in fact developed by and for the Compuserve online service. The B protocol has however developed along similar lines as other protocols.

It is not quite true to say that each protocol was better than the last. In many cases the choice of best protocol depends on the application. For mainframe transfers Kermit is king. For high speed transfers on international links Zmodem is the leader, and for simplicity and reliability Xmodem and Ymodem are hard to beat.

You will find that not all services support all possible protocols, which is another reason that Odyssey provides several, allowing you to choose which you prefer of those the service supports. You would be wise when you join a new service to ask established users which is the best protocol to be used in your situation.

Odyssey File Transfer

Which is the Best Protocol to use?

There is no protocol which is best in all circumstances. As mentioned in another topic, Xmodem is attractive because so simple and hence reliable - a useful fallback if nothing else works. For transfers to mainframes where control characters in the file are significant, Kermit is a good choice. Zmodem is usually the best choice in most situations, assuming it is available.

Given the caveats mentioned above, here is a list of the file transfer protocols which Odyssey supports, listed in decreasing order of desirability:-

Zmodem
Kermit *
Compuserve B+
Ymodem Batch
Ymodem
Xmodem
ASCII

Zmodem is the most desirable, because it is the fastest, and the most convenient to use. Kermit is marked with an asterisk because its position in the league table depends very much on whether the host Kermit supports a certain optional Kermit feature known as "Sliding Windows". This feature greatly improves Kermit performance, which is otherwise abysmal. If the host Kermit does not support Sliding Windows, then Kermit drops from second in the table to second last (in the relegation zone).

The different positions in the table mostly reflect the relative performance of each protocol, except that is for ASCII. This one comes bottom not because it is slow (it is potentially the fastest of all), but because it is not actually a protocol in any meaningful sense of the word. ASCII file transfer is simply the dumping of text to the serial port, which no attempt at verification, and no way to handle binary data. ASCII is a protocol of last resort only.

Odyssey File Transfer

PC to PC File Transfer

This section assumes that you wish to perform a PC to PC file transfer using a copy of Odyssey on both computers. If one of the packages is not Odyssey then you will need to read the appropriate section of the manual for the other package, and find a protocol supported by both Odyssey and the other package. Most comms packages can be expected to support Xmodem at the very least.

Step 1 - Getting the hardware right.

To connect two computers together you will need a cable, and you must also identify the correct serial port connector on each machine. If you are transferring between two PC compatibles then you will need a correctly wired serial cable - see cabling requirements.

PC compatibles sometimes have a serial connector built in as standard, and sometimes a separate serial adapter card must be plugged in an available slot. If your machine does not have a serial adapter installed then you will need to purchase one before you can go any further. Do not mistake the PC printer port for a serial port. The printer port is female 25-pin, whereas the serial port is a 25-pin male on an XT, and usually is a 9-pin male on an AT - this means that in both cases the required connector on the cable must be female. Note that a male connector has pins, while a female connector has holes into which the pins of a male connector can fit.

Make sure that both ends of the cable are firmly inserted into their respective PC connectors. When removing a cable from a PC you should always grip the connector shell - never pull on the cord, as this could break the wires.

Step 2 - Setting the Communications Parameters.

You now need to decide what speed you will transfer data at. PC compatibles running Windows can generally handle 9600 baud with no problems, and ATs can often manage 19,200 or even 38,400. If you have not tried a direct file transfer before then you should set a slow speed to begin with (perhaps 2400 baud), once you have the basic file transfer procedure sorted out you can test reliability at higher speeds later.

Having chosen a speed you should run Odyssey on both systems, and enter the Odyssey Setup| Comms... dialog and set the baud rate to that speed on each. You should also set both computers for eight data bits and no parity, and you should disable flow control for this first attempt (again, this is something you can play with once the basic procedure has been established).

Remember also that if your PC has more than one serial port then you must make sure that Odyssey has been configured to use the correct one (ie., the one your cable is attached to).

Click on OK in the setup dialogs to confirm any changes made, and activate the terminal window on both machines.

Step 3 - Testing the Link.

You can test the link by typing a few characters on the keyboard of one machine. These characters should appear on the terminal window of the second computer - they should not appear on the terminal display of the one whose keyboard you are using. If this is so, then you should now do the same test on the other machine, checking again that characters appear on the screen of the remote computer only.

If you do not see your characters being echoed, then there are several possible causes:-

- *You did not select the correct serial port in Odyssey*.
  This can only happen if you have more than one, since Odyssey will not allow you to select a serial port which does not exist in your hardware.

- *Your baud rate is incorrect*.
  This can be easily checked in the Setup|Comms dialog, and should be corrected if different baud rates are set on the two machines.

- *Your cable is incorrect*.
  This is the most likely problem, and unhappily is the most difficult to fix. If you own a breakout box and know how to use it, then you can sometimes diagnose the problem, otherwise you may need to seek expert help. This is not a problem which we can diagnose over the telephone.

If something does appear on the other computer when you type, but it is not the characters you typed, then you likely have a baud rate or parity problem, which is easily fixed - go back to step two.

Step 4 - Transfer the File (using Zmodem).

Since communications are established and tested, you may now transfer the file. Go to the computer which has the file you want to send, and pull down the **Upload** menu.

Select the Zmodem file transfer option.

A dialog will appear, asking you to enter a file name, which you should now do. Zmodem allows multiple files to be sent if the filename you enter contains wildcards (Kermit and Ymodem Batch allow this also, but the other protocols do not).

Click **OK** in the filename dialog, and the file transfer window will appear on both systems, showing the progress of the file transfer as it proceeds. An alarm will sound when the file transfer is completed.

When Odyssey receives a file, that file is stored in the Odyssey directory unless an alternative has been named using the **Directory for Downloads** field of the Setup|General... dialog. Similarly with uploads, if you do not specify a file path when you enter a filename, Odyssey will normally look for the file in the Odyssey directory, unless an alternative has been named in the Setup|General... dialog field, **Directory for Uploads**. Odyssey will not look in this directory if the entered filename includes a drive or path.

Using other Protocols.

When transferring between two Odyssey packages, Zmodem is by far the simplest protocol to use. However, if one side is not Odyssey, and does not support Zmodem then you may need to select another protocol.

You should not attempt to use the Compuserve B+ protocol to transfer files between two directly connected PCs; this protocol is specifically designed to be controlled by a Compuserve host - Odyssey is not a Compuserve host, and so the transfer will not work.

All protocols which Odyssey supports other than Zmodem require that you tell the receiving computer to receive a file - unlike Zmodem, which will automatically enter receive mode when it detects an incoming file header.

The following description assumes that Odyssey is used at both ends. However, it is likely that in actual fact you will be using another package at one end (otherwise you would use Zmodem as above). You should therefore translate these instructions for one end into those appropriate for the package you are actually using.

On the sending computer, display the **Upload** menu, and select a protocol other than Zmodem. You will be prompted for a filename, which you should answer, pressing <Enter> to complete the entry. Remember that Ymodem Batch, Kermit and Zmodem allow wildcard filenames, the others do not.

On the receiving computer you should display the download menu, then select the same protocol as you chose on the sending machine. If you choose ASCII, Xmodem or Ymodem you will be asked to enter a filename for the received file. You are not asked this question if you select Ymodem Batch or Kermit, since those protocols have the ability to exchange filenames during the transfer.

You will find that if you take too long to select the receiving protocol, and enter a filename for the received file, then the sending computer may have timed out and aborted. A useful technique is to delay pressing return on the filename prompts at each end until you are ready to go, ie :-

- Type the filename at the sending end, but do *not* press <Enter> to start the transfer.
- At the receiving computer, if the protocol required is Xmodem or Ymodem, type 'X' (or 'Y') and enter the filename, but do not press return. For other protocols except Zmodem, display the download menu, but do not select the protocol.
- On the sending computer, press the <Enter> key.
- On the receiving computer, press the <Enter> key (or, if not X or Ymodem, then press the key to select the required protocol).

In other words, prepare each machine up to the point of the final <Enter> (or button click) which starts the transfer, then press <Enter> on both machines. Performing the transfer in this way will ensure that the sender never times out before you have time to start the receiver.

Odyssey File Transfer

File Transfer via Modem

Since you wish to perform a file transfer using your modem, it is assumed that you are connected to some remote host. Unfortunately it is not possible to give exact details of what you should do next, since that will depend on the host software used. The procedure can be described in general terms however.

We will assume that you have already established a session with the host concerned. These are some of the things you will need to know about the host system :-

- Where to find files which are available for download.
- Which file transfer protocols are supported.
- How to select your preferred file transfer protocol.
- How to select the file you wish to download.
- How to tell the host that you wish to upload/download.

If you have the answers to these questions then continue as follows.

1. Enter the host "files" subsystem. Not all hosts divide themselves into subsystems, some may have files available for download at any time.

2. Display the list of available files, and note the name of one you would like to download.

3. Tell the host that you wish to download a file.

4. Tell the host which protocol you require. This must be a protocol which both Odyssey and the host support. Ideally this should be Zmodem, but if the host does not support Zmodem then you must choose another protocol.

Some hosts do not prompt you for a choice of protocol before each transfer. This could be either because the host only supports one protocol, or else the the preferred protocol has been saved in a permanent configuration "profile" for each user such as yourself. If this is the case you will need to find out how to edit your profile if you would like to use a protocol other than the default. In any case, you will need to find out what the default protocol is.

5. The host will now ask you which file you wish to download. You should answer this question by typing in the file name you noted earlier.

6. The host will now tell you that it is about to start a file transfer, and will warn you to get your software ready to receive. If you are using Zmodem or Compuserve B+ then you need take no further action, since Odyssey will automatically detect the incoming file (provided this option is enabled in Setup|File transfer...).

   If you are not using Zmodem or Compuserve B+, then you should (as quickly as possible) :-
   - Press **ALT+D** to display the Odyssey download menu.
   - Select the required protocol (this must be the same protocol you chose on the remote host).
   - If the protocol used is ASCII, Xmodem or Ymodem, then you will be asked for a filename for the received file. You can enter any valid DOS name, but you would normally keep the name the same as it was on the host. Other Odyssey protocols do not require you to enter a file name.

7. The Odyssey File Transfer Status window will then appear, and the file transfer will start. The status window will allow you to monitor the progress of the transfer as it proceeds.

These instructions are naturally a guide rather than a rule. Most host systems will work in this way, although some may ask the questions in a different order. The procedure for uploading to the host is much the same, except that you should of course choose the upload options on both the host and in Odyssey. Unlike file downloads, Zmodem cannot automatically detect when you wish to upload, although Compuserve B+ can! For Zmodem you must select the **Upload** menu and enter a filename as you would for other protocols. Except for Compuserve B+, a filename is always required for uploads.

You may find uploads to be less reliable than downloads. The reason for this is that while the Odyssey software can usually receive as fast as most current modems can throw data at it, this is not always when sending to the modem or to the host. If you have a problem of this type then you will need to think about enabling a flow control option.

Whether you choose XON/XOFF or RTS/CTS flow control depends very much on your combination of file transfer protocol, modem, and host software. Of the protocols which Odyssey supports, only Zmodem, Kermit and Compuserve B+ will work with XON/XOFF flow control, because the other protocols cannot distinguish between an XOFF which is part of the file data, and an XOFF meant for flow control.

Hardware flow control (ie. **RTS/CTS**) will work with all protocols, however you should bear in mind that this form of flow control applies *only* to the link between your modem and your PC. An instruction to your modem to stop transmitting to you does *not* cause the remote computer to stop sending to your modem, and data would almost certainly be data lost. RTS/CTS flow control is useful only when your modem is communicating with the remote modem using a slower link than the one it is using to communicate with your PC, for example when you use a speed buffered V.23 modem, a multi-speed modem, or a modem with internal error correction and data compression.

If you are using Odyssey software MNP, then flow control problems should not arise, since MNP automatically manages flow control.

Remember that when Odyssey receives (downloads) a file, it places that file in the Odyssey directory, unless an alternative directory has been named using the **Directory for Downloads** option in the Setup| General... dialog. Also, if the protocol being used requires that a receive filename is entered (ie. when you use ASCII, Xmodem or Ymodem), and you enter a path drive or path as part of the filename, then Odyssey will put the file where you have specified, and will not use the default directory.

Similarly, when you name a file for upload, and do not include a directory path, then Odyssey expects to find that file in the current directory, unless an alternative directory has been named using the Setup| General... **Directory for Uploads** field. If the file name you enter includes a drive or path then Odyssey will not look in the default upload directory.

Note that Odyssey does not automatically create a download directory if it does not already exist. Therefore, if naming a destination directory for the received file using either method described above, you must ensure that the directory exists, otherwise an immediate "File Creation Error" will occur as soon as the download starts.

Odyssey File Transfer

Summary of Protocols

This section gives some technical details and a historical summary of the various file transfer protocols which Odyssey supports.

Odyssey Supported Protocols

ASCII

ASCII is the simplest of the protocols, and the most error prone. Although Odyssey offers ASCII in both the **Upload** and **Download** menus, in download mode ASCII file transfer is just an alternative method of turning on text logging (text capture).

There are options in the Setup|File transfer... dialog which control some aspects of ASCII file uploading. These settings are the character and line delays, and the blank line expansion feature.

The character and line delays are used to reduce the effective transmission speed of an ASCII file. This is needed because quite often the host has no specific file transfer capability, and you are in fact simply pasting data into a host text editor. These have naturally been designed with human typing speeds in mind, very much less than the 240 cps or so which can be expected of a typical modem. Sending characters at these speeds would overload the host system, causing it to lose some of the data it has been sent. The character delay introduces a pause between each character as it is being transmitted, and the line delay inserts a separate pause at the end of each text line. Taken together, these delays can reduce the effective speed of transmission to a level which the host can handle. You may however need to experiment to find out what the proper delays should be. Remember when deciding this point that the appropriate delay can depend very much on the time of day, ie. how busy the host happens to be at that time.

The **Blank Line Expansion** feature solves a problem which arises with some BBS systems, and possibly some mainframe hosts too. The problem is that these systems treat reception of a blank line as an instruction to leave the editor (the one you are currently uploading lines to). This is bad news, since if the blank line was actually not the last line in the file, then remaining lines are uploaded to the host after it has left the editor, and may potentially wreak all sorts of havoc if they happen to contain valid host commands. Enabling the blank line expansion feature tells Odyssey to replace each blank line in the file being uploaded with a line containing a single space character. This is normally enough to satisfy the host that you have not finished entering text.

Odyssey Supported Protocols

Xmodem

Xmodem is one of the oldest and simplest of file transfer protocols, designed by Ward Christensen in 1977 and placed in the public domain. It has gained in popularity over the years until now it would be a hard task to find any BBS or comms package which does not support this protocol. The main reason for the popularity of Xmodem is almost certainly its simplicity, a factor which makes it easy for programmers to implement, and once implemented, it is generally more robust than other protocols.

Xmodem works by breaking the file to be transferring into packets each containing 128 bytes of data, and transmitting these packets to the remote modem one at a time. After each packet is transmitted the sender waits for it to be either accepted (ACKed) or rejected (NAKed) by the receiver, which determines whether the sender next retransmits the previous packet, or sends the next one. If there is no more data to transmit, then the sender transmits a single end-of-transmission (EOT) character, which tells the receiver that the file transfer is complete.

Xmodem does have design drawbacks, mainly that it requires a completely transparent eight bit link (ie. control characters, or bytes with the eighth bit set must not be intercepted by the medium), and that its performance on timesharing systems, packet switching networks and satellite links can be extremely poor; because of all that time Xmodem spends waiting for the ACK or NAK reply to a packet (a significant wait on systems with long turnaround delays). However, under the right conditions, Xmodem generally does its job very well.

Over the years, several important weaknesses of Xmodem have been highlighted. The simple checksum method it uses to protect data integrity is not particularly effective, and the single character messages it uses to acknowledge and reject packets, or mark an end of file makes it prone to being confused by line noise, where these characters can occur accidentally.

The solution to the checksum problem was to design a variant of Xmodem which used a CRC check instead. The problem of single character messages has not been dealt with, although certain tricks can be used by the Xmodem implementation to try and make it less prone to spurious end of file characters; the trick is to always NAK the first EOF seen, and if it was genuine the sender should respond by transmitting it again.

Odyssey supports both Xmodem and the Xmodem-CRC variant, and also performs the EOF check for improved robustness.

Odyssey Supported Protocols

Ymodem and Variants

Ymodem is a protocol designed by a programmer named Chuck Forsberg, although Ward Christensen may have been responsible for coining the actual "Ymodem" term.

Ymodem is an attempt to improve upon the performance of standard Xmodem over slow networks. The solution adopted by Ymodem is to use larger packets, 1024 bytes instead of the 128 byte packets of Xmodem. This results in a Ymodem transfer requiring fewer packets for the same file, and hence fewer waits for acknowledgment. However, the performance benefit derived from using this technique is not exactly startling. One penalty associated with using a larger packet size is that if errors occur, more data has to be retransmitted. On a noisy line, Ymodem performance can actually be worse than simple **Xmodem**.

Although all Ymodem implementations use the 1024 byte packet size, the exact definition of this protocol has always been a matter of confusion, despite the persistent efforts of Forsberg to get programmers to implement the complete protocol before slapping the "Ymodem" label on it.

There are two main variants which are consistently called Ymodem. The first of these is simply Xmodem with the few minor changes required to support the larger packet size, in fact Forsberg prefers to call this version **Xmodem-1k**.

Note: The Odyssey **Upload** and **Download** menus continue to call the above protocol **Ymodem**, since that is what it is likely to be called on most BBS systems. Odyssey refers to the other Ymodem described below as "**Batch Ymodem**".

The other common Ymodem variant version is "True" or "Batch" Ymodem. This has a similar packet structure to the simple variant, but adds several extras to make it a slightly more sophisticated protocol. The first of these extras is that a file transfer may use any combination of Xmodem or Ymodem sized packets, as appropriate. It sometimes makes sense to reduce packet size, for example when heavy line noise is encountered (thus avoiding some of the performance penalties mentioned above), or when there is less than 1024 bytes of data left to transmit. The other main feature of **Batch Ymodem** is implied by its name, and that is the ability to send a "batch" of several files in a single session. It manages this by sending an initial header packet before the file data, and this packet will contain details about the file such as its name, size in bytes, and various other items of information which the receiver can use to construct an exact duplicate of the received file on the local file system. After the first file has been sent, the receiver waits for a new header packet, and if this packet contains another file name, then another file transfer begins. If the header packet is blank, then the batch session is completed.

There is a final variant of Ymodem, called **Ymodem-g**. This protocol was designed for use on error corrected links, such as can be provided by Odyssey using its internal MNP engine. This variant is almost identical to Batch Ymodem, except that it never waits for acknowledgements. Instead it assumes that the error controlled link will ensure that all packets are received correctly by the remote. Ymodem-g is an example of a what is known as a *streaming protocol*, referring to the fact that it sends data in a continuous stream without pauses. This can make Ymodem-g performance look quite impressive, but the lack of any kind of error recovery or flow control capability makes this a risky protocol to use. The "g" option of the Batch Ymodem protocol can only be negotiated by the receiver, however for reasons of symmetry Odyssey offers this protocol in both the **Upload** and **Download** menus; however selecting Ymodem-g in the upload menu is just another way of selecting Batch Ymodem - Ymodem-g will only be used if the receiver requests it. To use Y-g with a remote host which doesn't explicitly offer a Y-g option you would select Batch Ymodem on the host, Ymodem-g download in Odyssey, and Odyssey will determine for itself whether the host actually supports this protocol or not. If it does not, then Odyssey will revert to a standard Batch Ymodem file transfer.

Odyssey Supported Protocols

Kermit

Kermit is a protocol developed by Frank Da Cruz at the University of Columbia in New York, in 1981. It has been developed almost continuously since then. In case you are wondering, yes, this protocol was named in honour of the famous frog.

Unlike the other protocols supported in Odyssey, Kermit was never placed in the public domain. Instead Da Cruz has retained all rights to the protocol with the intention of controlling future development, and also ensuring that no one has to pay more than a nominal amount to use it. Commercial implementors are allowed to include Kermit in a product such as a comms package, provided that the price charged is not raised substantially because of it.

Kermit arose in an academic environment, in response to a need to exchange data between the central computing facility, and a wide variety of mini and micro computers operated by students and staff. The protocol therefore had to make minimal assumptions about the capabilities of the machine on which it might run, or on the communications link which might be used; in fact Kermit is the only supported protocol which is capable of operating both over networks which are not transparent to all control characters, and on links which are parity checked (ie. in which it is only possible to send seven bits of data per character).

Unlike other protocols, the main design consideration in Kermit was portability, rather than performance. Even so however, Kermit performance can be *very* poor. For example, Kermit was originally designed to assume minimal buffering of data on the part of the receiving computer, and so packets contain only 64 bytes of user data. The original Kermit was also a stop and wait protocol like Xmodem, and so where Xmodem performance can be bad on slow networks, standard Kermit, having twice as many packets to send, is far worse.

Over the years many improvements have been made to Kermit which greatly enhance performance, however you should be aware that these are *optional* features which many Kermits do not implement, and so you may still suffer worst case performance during a Kermit file transfer. There have been two main improvements to Kermit which improve performance, namely **larger packet sizes**, and **sliding windows**.

Larger packets do for Kermit what Ymodem did for **Xmodem**, that is, reduce the number of packets which need to be sent for any given file, hence reducing the time spent waiting for acknowledgments.

Sliding Windows Kermit (sometimes called **SuperKermit**), adds a feature called *windowing* to the Kermit protocol. A windowed protocol is one which sends several packets before it must wait for an acknowledgment, the "window" being the number of packets it can send. This is a big improvement on protocols like Xmodem which must wait for an ACK for every packet. In fact a windowed protocol only ever stops sending when it has N (its window size) outstanding packets, ie., not yet acknowledged.

All of this would simply be the equivalent of increasing the packet size were it not for one other feature, which is that an acknowledgment (ACK) for an old packet can be received at any time, even while it is currently sending a later one. If the sender is kept well "fed" with acknowledgements, then it may never have N unacknowledged packets, in which case it never stops sending until the file transfer is complete. This means that even on a slow satellite link, the data will arrive at the remote modem in a continuous stream, with almost no pauses.

In a normal windowed protocol, if the receiver NAKs a packet, then the sender must retransmit the packet in error, *and* all the packets which followed it which it had already sent. However, Kermit rather cleverly implements a "*sliding window*" which means that it is only necessary to transmit the individual bad packet. This feature makes all the difference for Kermit when used on slow networks, and makes it one of the best performers around, second only to Zmodem, and probably better than Zmodem if you have a noisy line.

Odyssey Kermit supports all of the Kermit options, including large packet sizes and sliding windows.

Odyssey Supported Protocols

Zmodem

Like Ymodem, **Zmodem** was designed by Chuck Forsberg and placed in the public domain. This was actually a condition of Forsberg's contract with Telenet, for whom he developed the protocol. Zmodem is probably the fastest of the protocols supported in Odyssey, as well as being the most sophisticated.

Zmodem, at least in the form implemented in Odyssey, is a streaming protocol, as was Ymodem-g. Like a windowed protocol it is capable of sending a continuous sequence of packets without waiting for acknowledgment, however unlike a windowed protocol it has no upper limit on the number of packets which may be sent. Unlike Ymodem-g, Zmodem does have the means for error recovery and flow control; the receiver can interrupt the Zmodem sender at any time to tell it to return to particular point in the transfer, and flow control is managed using ordinary **XON/XOFF** software controls. These features make Zmodem performance similar to that of Ymodem-g, but without the riskiness.

Odyssey supports two other features which make Zmodem especially desirable if you find that your host BBS supports it, the two features being **automatic downloading**, and **resumption of an interrupted file transfer**.

Auto-download means that Odyssey automatically recognises an incoming Zmodem file header at any time, and jumps directly into its Zmodem receive routine when it does so. This means that on the host you simply need to select Zmodem download, name the file(s) you wish to receive, and take your hands away. Unlike other protocols you do not also need to tell Odyssey that a download is starting - the host will tell Odyssey all it needs to know.

Resuming an Interrupted Transfer means exactly what it says. In other protocols if a file transfer is interrupted (by a lost carrier perhaps), then you must redial the host BBS, get back into the file section, then download the file again from the start. With Zmodem the difference is that the file transfer continues from the point at which it was interrupted, potentially saving a great deal of time and expense.

Odyssey provides options in the Setup|File transfer... dialog which can be used to control some aspects of Odyssey Zmodem file transfer. This setup section allows you to enable or disable **Full Streaming** - if disabled, Zmodem will wait for an acknowledgment every now and then to avoid outrunning a slow receiver, you can choose to **Escape all control characters** if the communications link is not entirely transparent, and you can also disable the auto download feature if you want to retain manual control, perhaps because you want to control file transfer from a script.

Odyssey Supported Protocols

Compuserve B+

The following is an extract from the **Compuserve B+** protocol specification document by Russ Ranshaw.

The CompuServe B protocol was developed in 1981 to provide support for a special purpose Vidtex terminal manufactured by the Tandy Corporation. It was the outgrowth of a proposed Bi-Sync oriented protocol, but with a different packet structure and provision for more than even and odd packets. The file transfer capability was added in 1982 to replace the CompuServe A protocol with a more robust protocol which was in keeping with the over-all B Protocol design.

Some of the underlying assumptions made in designing the B Protocol were due to the capabilities of personal computers which were available at the time. Such machines were generally limited in the amount of available memory, 64 kilobytes being a large capacity. Other factors, such as the lack of a true UART for data communications, resulted in the send/wait nature of the protocol where only a single protocol packet at a time was sent.

The explosive growth of the Personal Computer industry has given us a plethora of machines, most of which have far exceeded the early limitations of memory and communication ability. This growth has been accompanied by a multitude of file transfer protocols, such as XMODEM, KERMIT, and ZMODEM. CompuServe, realising the need for enhancement, has developed the B Plus Protocol to meet the increasing demands being made upon its communication network and host computers, and to provide added utility for its large family of users.

As the name implies, B Plus is an extension of the B Protocol. In particular, the B Plus enhancements add:

- The ability to send multiple packets without waiting for individual acknowledgements (ie., windowing).
- Larger data packets (up to 1k at present).
- Optional use of modified XMODEM CRC-16 check method.
- Extensions to the standard control character quoting.
- Provision of a mechanism to exchange transport and application parameters.

Odyssey fully supports the B Plus protocol. Odyssey also provides some options in the Setup|File transfer... dialog which can be used to control some aspects of B+ file transfer. This setup section allows you to enable or disable windowing or **packet send-ahead** - if disabled, CIS B+ will wait for an acknowledgment for every packet. You can choose to **Escape all control characters** if the communications link is not entirely transparent, and you can also disable the auto transfer feature if you want to retain manual control.

☞ There is one nasty problem with the CIS B+ protocol which you should be aware of. B+ allows for the host to automatically initiate a file transfer, just like Zmodem does. However, unlike Zmodem, the Compuserve host triggers the terminal package using a single character (Zmodem uses a sequence of ten characters or so for this). That is bad enough, as the character could be produced accidentally by line noise; but to make matters worse, the chosen character is ENQ (ASCII 5), which is commonly transmitted by many hosts or intermediate networks when they wish to receive an answerback message from the terminal.

The upshot of all this is that, while it is completely safe to leave the Zmodem Auto-Download feature enabled all the time, doing the same with CIS B+ would be very dangerous. Odyssey provides a Setup menu option to enable and disable this feature, and we strongly recommend that you leave it disabled until after you have made a connection to the Compuserve host (ie., until after you have logged on). This is best handled in your autologon script for Compuserve. Look for a sample script in your Odyssey directory called CISV32.SCR, which demonstrates a good way to do this.

Odyssey File Transfer

ASCII transfers with Zmodem and CIS B+

Both Zmodem and CIS B+ protocols support an *ASCII transfer* mode, in which the local machine may perform any format changes required for the local system - for example, converting a LF-terminated Unix file to CRLF for DOS. For both protocols this feature is ***completely ignored*** in Odyssey, and the file is stored on disk, containing data exactly as the host sent it. This is done for good reason: both of the above protocols have nice "resume-interrupted-transfer" features, and in both cases the feature is nearly impossible to implement if the portion of the file on the local machine has been modified by a conversion process and therefore has a different length or CRC than the same portion of the same file on the remote. We consider the auto-resume feature more important than the convenience of automatic conversion, so we leave that to external filter programs once you are sure that the file has been correctly received.

Odyssey Features

Terminal Emulation

Odyssey has the ability to emulate a variety of different terminal types. Choose one of the topics below for further information.

See also: The Viewdata (Prestel) terminal emulation.

Odyssey Terminal Emulation

What is a Terminal Emulation

The dictionary definition of **terminal** defines the word as meaning "*of, being, or situated at an end, terminus, or boundary*". In the computing sense the terminal is the boundary between the human and the machine. It is the point at which a human operator plugs into a computer system.

Traditionally, a terminal is a peripheral device attached to the computer by means of a communications link, at which the operator can enter commands, or review data produced by programs.

In the early days of computing the terminal was a highly mechanical device which was a combination printer and keyboard, called a teletype, and often abbreviated to TTY. Teletype is actually a trade name, but as in many other situations, the trade name became a generic label for all such devices. The teletype made a distinctive clattering sound when printing which was synonymous with computers (especially in films and TV news programs) long after the device itself was obsolete. Most of the non-printing characters in the ASCII set were originally designed with control of this device in mind, for example ASCII code thirteen is the carriage return character, which moved the print head to the left margin, code ten is a linefeed, which caused the paper roll to feed forward by one line, and so on. The teletype was hardly the ideal output device - it was extremely limited in printing capabilities, it was slow, it was noisy, it kept running out of paper, the ribbon would dry out or become tangled - and those were its good points!

As usual, technology came to the rescue. A revolutionary new terminal device was designed which replaced the paper roll with a glass TV tube. This was totally silent (except for the sound of keys being pressed), and had no paper roll or ink ribbon to cause problems. Also, since it was an electronic device, it could accept and display computer output far faster than was possible with its mechanical forbears, and instead of a print head, a symbol called a **cursor** was used to indicate where the "carriage" was at any given moment. However, since this device was designed as a direct replacement for a teletype, it had no extra capabilities, and was still controlled by ordinary ASCII codes. This led to the device being referred to as a "glass teletype", and in later years as a "dumb terminal", when smarter terminals started to appear.

Eventually it was realised that there was no reason why this electronic terminal should be limited to the same capabilities as a device which produced physical output on a paper roll. For example, there was no reason why the carriage (cursor) could not be repositioned instantly to any point on the display, or why text could not be overwritten or erased. The only problem was that the standard ASCII set did not include any characters for advanced terminal control, so some extensions to ASCII would need to be developed.

It would have been nice if industry leaders had got their heads together and come up with an extended terminal control standard, however this was still in the days where large computing corporations such as IBM, Honeywell, Sperry, and DEC were almost the only players in the market, and they generally did their best possible to tie customers into their proprietary hardware. The last thing these manufacturers wanted was a standard which allowed a customer to buy equipment at competitive prices from any supplier they chose. And so each manufacturer designed their own terminal control standards, many of which still exist today in refined versions - the most common early terminal control standard still in use today is probably the DEC VT52 code set. A recognised standard was in fact eventually developed by ANSI (an American standards body), and DEC to their credit were among the first to adopt at least part of this standard in their DEC VT100 family of terminals. These days most manufacturers are slowly doing the same, although many include the ANSI control standard as an optional feature in their otherwise proprietary devices. Most small BBS also use a subset of ANSI to provide attactive color user interfaces, that subset is therefore called ANSI-BBS

In the eighties the micro revolution really arrived, and with it came an inconvenience. In many of the large companies running a big central computer, a computer user would often find himself with both a microcomputer to perform personal work, and a separate terminal to access the company mainframe. This took up a great deal of desk space. It would be simpler if the user could get rid of the terminal and use the micro for everything, and this was made possible through a technique called ***Terminal Emulation***.

A terminal emulator is a program which runs on your personal computer, and which can tie into the communications link to a mainframe host, and which is capable of recognising and reacting to the commands which the host computer sends in order to control what it thinks is an ordinary terminal. To simulate a full terminal the emulation must translate keys typed on the PC keyboard into equivalent key sequences understood by the host, as well as performing the correct display operation when a command is received (such as a command to erase a portion of the display, or reposition the cursor). Sometimes there are fundamental hardware differences between a PC and the terminal which make emulation difficult, such as a different keyboard layout, or a different number of lines on the display, or specialised text effects not available on the PC. However, given a little compromise, it is usually possible to emulate almost any proprietary terminal on the PC, given the availability of appropriate emulation software.

Odyssey is a comms package which includes some terminal emulation capabilities, although by default it emulates a simple dumb terminal (the good old TTY device). Odyssey supports emulations of several different manufacturers' terminals, including VT320, VT100, VT52 and DG200. Also included is an ANSI emulation, which implements that aforementioned subset of the ANSI terminal standard often used by bulletin board systems. Odyssey also supports a special terminal emulation for use with the British Telecom PRESTEL service, and similar **Videotex** hosts. Finally, there is a debug terminal (DBGTERM) which is used when you want to know what characters that host is *really* sending to you.

Odyssey Terminal Emulation

Which emulation should you use?

The emulation you should use depends on what you want to use it for. If you are not using Odyssey for anything except file transfer between two machines running Odyssey, then you do not need any particular terminal emulation at all, and the default TTY setting is perfectly satisfactory. Likewise, if the service you use produces output in a purely sequential, scrolling manner then no emulation is required and you should leave Odyssey set to **TTY** emulation. Some users like to configure their comms package for ANSI emulation as soon as they get it, simply because that was what they used to do on other packages. Although this is normally quite harmless, it is rather a waste of memory and computer processing time, since all those terminal features being emulated are not actually being used. It can also lead to strange effects, for example when line noise accidentally coincides with a terminal control function, perhaps to clear the display, or even change color!

If you are connecting the PC to a host computer, either by a direct serial cable, or by a telephone connection, then you must first find out which emulation the host expects you to use. Sometimes the host allows you to use any of several terminals, in which case you should just pick whichever makes life easiest. Ask the host help desk if you are not sure.

Many of the smaller BBS hosts support ANSI terminals to give nice color menus etc. This would normally be optional, and the BBS login sequence will ask whether you want to use ANSI or not. Remember that before you say yes, you must have Odyssey configured to expect ANSI control sequences by selecting that terminal emulation in the Setup|Terminal dialog (or by putting **ANSI** in the emulation field of the dialing directory entry).

Odyssey Terminal Emulation

How to select a terminal type

There are several ways of telling Odyssey which terminal emulation the host requires. You can do it either from the keyboard, from a script, or from the dialing directory.

If you want to make a particular terminal emulation a permanent default, then you open the Setup|Terminal emulation dialog, and select the terminal type from the Terminal type listbox - remember to save your changes if you don't want this selection to be lost when this Odyssey session ends. However, this procedure is not always the best option, since in many cases it is not necessary for Odyssey to be emulating a terminal except when it is actually online to a host.

The second (and easiest) method is to provide the name of a terminal emulation in the **Terminal Emulation** field of the dialing directory entry for the relevant service. If you have read the description of the dialer's calling procedure you will know that Odyssey automatically loads the emulation named there when a successful connection is made with that service. This feature is especially useful with PRESTEL emulation, since that emulation is rather inconvenient to use when communicating with the modem offline.

A third alternative is to control the loading of terminal emulations from a script using the **Emulate("name")** command. To do this you will first need to become familiar with the Odyssey script language.

Note that whatever method is used to load a terminal emulation, in all cases Odyssey must be able to find the appropriate .TRM file for the emulation you select. These files contain the code for the emulation itself, so that the main Odyssey program is not cluttered up with terminal emulation routines that are never used. The INSTALL program places these files in your Odyssey directory automatically.

Odyssey Terminal Emulation

Customising a Terminal Emulation

In some situations you may wish to add a few extra features to an Odyssey terminal emulation. The most common reasons are that you have an extended keyboard, and would like to have a more convenient keyboard layout (Odyssey terminal emulations are normally designed so that they will work on the original IBM PC keyboard), and another common reason is that you are communicating with a host mainframe through a device called a protocol converter (a device which translates from one terminal emulation to another), which often expect you to type strange keyboard sequences to obtain equivalents of special function keys not found on the terminal being used.

In both cases the solution is to use the Odyssey Keyboard Mapping feature to design a new keyboard definition. This utility is described in full elsewhere.

Once you have a new keyboard definition in the form of a .KEY file for Odyssey, you then need to load that definition on top of the current terminal emulation. You can do that using the ALT+K command (or by selecting that option from the terminal window Command menu), however an easier way is to have it loaded automatically whenever Odyssey loads a particular terminal emulation.

Whenever Odyssey loads a terminal emulation, it looks to see if there is an attached keyboard template with the same name. For example, if you load VT100 emulation, Odyssey looks for a file called VT100.KEY, and will load this keyboard definition after the emulation, if it exists.

If you do not want to use the keyboard definition every time that emulation is used, then you can take care of that by using File Manager to make a copy of the terminal emulation file, using a different name. For example, you could make a copy of the VT100 emulation and call it VT-SPEC.TRM, then create a keyboard template called VT-SPEC.KEY. Now, whenever you load the VT-SPEC emulation, the VT-SPEC keyboard template will be loaded as well, but will *not* be loaded when you select ordinary VT100 emulation.

Odyssey has a field on the terminal window status line which it uses to tell you the type of terminal currently being emulated. However, if you have loaded a keyboard definition on top of an emulation then Odyssey will display the name of the keyboard template instead, in the case of the above example, Odyssey would display "VT-SPEC" in that field. This allows you to tell, by glancing at the status line, whether you are using the standard Odyssey terminal emulation or your customised version.

Odyssey Terminal Emulation

Summary of Odyssey Terminal Emulations

This section provides specific information on the various terminal emulations supported in Odyssey, as of the time this manual was prepared. Note that the authors reserve the right to make changes to the terminal emulations, or to add new emulations or withdraw old ones. We would only expect to do the latter if we had replaced it with a new emulation which fully supported the original as a subset. In any case, you will find details of any such changes in various .DOC files on the Odyssey distribution disk, which you can view using the Odyssey message editor.

You may feel that some of the keystrokes required by one or more of these terminal emulations is rather awkward. If so, please bear in mind that Odyssey cannot be designed only for users with the most capable hardware. Most default key combinations for example are designed so that they can be used on an original IBM PC keyboard. Remember that you can always redefine your keyboard as you prefer, using the keyboard remapping facility.

The topics below provide information on specific terminal types supported by Odyssey :-

Teletype (TTY)
Digital Equipment Corporation (DEC) VT52
Digital Equipment Corporation (DEC) VT100
Digital Equipment Corporation (DEC) VT320
Data General Dasher 200 (DG200)
ANSI-BBS (ANSI)
Odyssey Debug Terminal (DBGTERM)
British Telecom PRESTEL (Viewdata)

See also: The Viewdata emulation in detail

Odyssey Terminal Types

Teletype (TTY)

This is the default terminal emulation in Odyssey, and the only emulation actually integrated into the main Odyssey program. Use this "emulation" when no special emulation is actually required.

Note also that the Odyssey Command Line Recall feature only works when TTY emulation is being used.

Odyssey Terminal Types

DEC VT52

The Odyssey VT52 terminal type fully emulates the true DEC product. The keyboard produces standard VT52 sequences for cursor and special function keys. VT52 emulation is also available in the Odyssey VT100 emulation, since VT52 emulation is also a feature of a VT100 terminal.

Odyssey Terminal Types

DEC VT100

The Odyssey VT100 Emulation supports the features of the original VT100 terminal, as well as those of the later VT102. The emulation however identifies itself to a host as an ordinary VT100, since some hosts do not recognise the VT102 identity.

The IBM PC family has several models of keyboard, but none exactly match that of a VT100 (although AT enhanced keyboards look quite similar). Certain compromises therefore have to be made in implementing a VT100 emulation on IBM clones.

The numeric keypad on an IBM PC does double duty as a cursor pad. When **NumLock** is set the keypad normally generates numbers, and if not set, the keypad generates cursor movement commands etc. This feature is a hangover from the early IBM XT-style PCs, which did not have a separate cursor key cluster.

A VT100 terminal has a very similar looking keypad, except that on the VT100, the top four keys on the keypad are function keys, named PF1, PF2, PF3 and PF4. Some DEC host software refers to PF1 as the 'gold key'.

In order to emulate the VT100, Windows Odyssey 'pretends' that the top four keys on your keypad are to be interpreted as PF1, PF2 etc (though only if a VTxxx emulation is in use, and then only if the terminal window is the active window). This naturally means that these keys cannot serve their normal purpose, ie. you should note in particular that the **NumLock** function is not available when you are typing into the VT100 terminal, and nor are the other keys in that row. In you press NumLock, it will cause the PF1 keyboard sequence to be transmitted to the host.

This layout does however create one minor problem, which is that a real VT100 terminal also has a '-' key on the numeric keypad, a key which is sometimes used as a function key; yet it looks like we can't generate that key code because we have just mapped it to PF4. To solve this problem Odyssey maps **ALT+<keypad '-'>** as the VT100 <keypad '-'> key, which will generate either '-' or the appropriate function key sequence, depending on the VT100 keypad mode set by the host.

Note that we could not have used the obvious looking mapping of PF1-->F1 etc, since a) this would block access to the online help normally on the F1 key, and b) it would not be consistant with the more sophisticated VTxxx emulations, which have the PFx keys *and* programmable Fx keys. Note also the keyboard mapping described here applies to all the Odyssey VTxxx emulations, ie. VT52, **VT100** and VT320.

The Odyssey VT100 emulation *does* support 132 column mode, however Odyssey does not have fonts thin enough to display this on one screen. Odyssey therefore implements "panning" which treats the terminal window as a "sliding window" on which you can view a selected portion of the display. The following keys control the portion of the 132 column region shown :-

Ctrl+Home	Show Left Side
Ctrl+End	Show Right Side
Ctrl+Left Arrow	Pan 10 columns to the left.
Ctrl+Right Arrow	Pan 10 columns to the right.

Odyssey Terminal Types

DEC VT320

This emulation can be thought of as a superset of the VT100 emulation described previously. All of the VT100 notes regarding keyboard mapping and 132 column support also apply to the VT320 emulation.

This emulation does not support down-line loading of soft fonts, since the VT200/VT320 font size is not usable on an IBM type display - we *may* add this in future for users who want to download soft fonts to Odyssey with a VGA cell size; however, this would make the host application Odyssey specific, so it probably isn't a good idea.

A VT320 terminal has the option of running in either "7 bit" or "8 bit" mode. When this emulation runs in "7 bit" mode it reports the terminal type on the Odyssey status line as "VT320-7b", and in "8 bit" mode it reports "VT320-8b". The terminal always load itself in seven bit mode, and requires a host command to switch to eight bit mode. Even then, Odyssey will switch to eight bit mode only if parity is set to "none" and "Strip Parity Bit" (in Setup|Terminal) is disabled, at the time that the command is received.

As briefly mentioned above, the notes regarding keys in the VT100 emulation topic also apply to the VT320 emulation. However, VT320 has a number of extra keys, in particular it provides up to 20 programmable function keys. For this reason we recommend that you use an enhanced keyboard (102 key) with the VT320 emulation, a keyboard which is as close as PC keyboards get to the DEC equivalent.

For convenience, applicable sections of the notes from the VT100 emulation are repeated here:-

The IBM PC family has several models of keyboard, but none exactly match that of a VT100 (although AT enhanced keyboards look quite similar). Certain compromises therefore have to be made in implementing a VT100 emulation on IBM clones.

The numeric keypad on an IBM PC does double duty as a cursor pad. When **NumLock** is set the keypad normally generates numbers, and if not set, the keypad generates cursor movement commands etc. This feature is a hangover from the early IBM XT-style PCs, which did not have a separate cursor key cluster.

A VT100 terminal has a very similar looking keypad, except that on the VT100, the top four keys on the keypad are function keys, named PF1, PF2, PF3 and PF4. Some DEC host software refers to PF1 as the 'gold key'.

In order to emulate the VT100, Windows Odyssey 'pretends' that the top four keys on your keypad are to be interpreted as PF1, PF2 etc (though only if a VTxxx emulation is in use, and then only if the terminal window is the active window). This naturally means that these keys cannot serve their normal purpose, ie. you should note in particular that the **NumLock** function is not available when you are typing into the VT100 terminal, and nor are the other keys in that row. In you press NumLock, it will cause the PF1 keyboard sequence to be transmitted to the host.

This layout does however create one minor problem, which is that a real VT100 terminal also has a '-' key on the numeric keypad, a key which is sometimes used as a function key; yet it looks like we can't generate that key code because we have just mapped it to PF4. To solve this problem Odyssey maps **ALT+<keypad '-'>** as the VT100 <keypad '-'> key, which will generate either '-' or the appropriate function key sequence, depending on the VT100 keypad mode set by the host.

On the main keyboard, the backspace key on the PC generates BS (ASCII 8) and not DEL (ASCII 127) which a VT100 user may expect. You can reverse this using the "BackSpace key sends..." option in Setup|Terminal.

The Odyssey VT320 emulation *does* support 132 column mode, but the PC hardware is not capable of displaying this on one screen. Odyssey therefore implements "panning" which treats the physical display as a "window" on which you can view a selected portion of the 132 column display. The following keys control the portion of the 132 column region shown:-

Ctrl-Home	Show left side
Ctrl-End	Show right side
Ctrl-Left Arrow	Pan 10 columns to the left
Ctrl-Right Arrow	Pan 10 columns to the right

A VT200/VT320 keyboard adds a separate arrow key cluster, a cluster of editing keys (Find/Insert etc), and 20 function keys, the first five of which generate no keystrokes, but serve local terminal functions.

Arrow key cluster. This is almost identical to the PC enhanced keyboard equivalent.

Editing keys. An enhanced keyboard is far more convenient for this than the old PC keyboard. This is the mapping of PC key to VT320 equivalent :-

PC key	**VT320 equiv**
Insert	Insert
Delete	Remove
Home	Find
End	Select
Page Up	Prev Screen
Page Down	Next Screen

Note that although the appearance of the PC cluster is similar to the DEC equivalent, the keytops are different. The emulation follows the PC keytops (where possible), rather than the VT320 keytops, otherwise things could get rather confusing, however you should beware of help messages from the host system (such as when using VMS *EDT*), which indicate the correct keys using a keyboard diagram.

Function keys. As noted above, the first five function keys serve local terminal functions both in Odyssey and on a real VT320. This is convenient, since it means that Odyssey F1 (help) can still work. Other keys :-

PC key	**VT320 equivalent**.
F6 to F10	F6 to F10
Shift-F1 to Shift-F10	F11 to F20

Also, for convenience, VT320 key **F15** (Help) is duplicated on PC key F11, and **F16** (Do) is duplicated on PC key F12.

Odyssey Terminal Types

Data General DG200

As with all PC terminal emulations, some compromise has to be made to account for the fact that what you are using is a PC, and not in fact the terminal it is pretending to be. The DG200 emulation is no exception.

Most compromises have to be made in the area of hardware requirements and keyboard differences. Happily the DG200 hardware does nothing which the PC cannot cope with, but there are small problems with the keyboard.

The major differences on a DG200 keyboard are that the latter has 15 function keys, whereas the standard PC keyboard has ten. Odyssey also normally requires use of function key F1 (for help), however this is set aside when DG200 emulation is in use, and another key is used to bring up help in terminal mode *ONLY* (see below). The compromise used by Odyssey is to use the SCROLL LOCK key as a toggle which decides whether the first five function keys are treated as PF1-PF5, or whether they should be PF11-PF15. If the SCROLL LOCK key is NOT active then F1 to F5 correspond to PF1-PF5, and if SCROLL LOCK is active then F1-F5 correspond to PF11-PF15. There is a similar mapping of these function keys (controlled in the same way) when used in SHIFT and CTRL combinations. As a convenience the DG200 keys PF11-PF15 can also be generated by typing ALT+F1 to ALT+F5, however no control or shift combination can be used with this method.

Another difference that you should be aware of is that the DG200 has an ENTER key (generating a linefeed - ASCII code 10) where most other keyboards have a key which generates a carriage return (ASCII code 13). Odyssey follows the DG200 standard and will generate LF instead of CR for the PC return key, however this introduces a problem in that you then cannot easily enter direct commands to a Hayes type modem (the most common variety attached to PCs), because these commands need to be terminated by a carriage return. You can get around this problem by typing Ctrl+Return when you want a carriage return to be generated, and Return on its own to produce the DG standard linefeed character.

DG200 Key	**PC Keyboard**
PF1 to PF10	F1 to F10 (with SCROLL LOCK off.
PF11 to PF15	F1 to F5 (with SCROLL LOCK on), or ALT+F1 to ALT+F5.

The above keys, apart from ALT+Fx, may also be used in combination with the Shift and Ctrl. For example Ctrl+Shift+F1 with SCROLL LOCK off will generate its DG200 equivalent, Ctrl+Shift+PF1. With SCROLL LOCK on the DG200 equivalent is Ctrl+Shift+PF11.)

Enter	Enter
Return	Ctrl-Enter
Cursor Keys	Cursor Keys
Home,End PgUp,PgDn	Same on PC
Delete	Backspace (key above enter key).

Remember also that you cannot use F1 to enter the Odyssey help system from terminal mode while using the DG200 emulation. Use Alt+= instead. This *only* applies to terminal mode. Terminal emulations do not affect the use of the F1 key when you are in an editor, a menu, or the dialing directory etc.

Odyssey Terminal Types

ANSI-BBS

This is the **ANSI** emulation expected by some BBS hosts. When active the keyboard will produce standard ANSI sequences.

Note that this emulation is designed to allow a host BBS to control the display of standard IBM PC colors and expects the complete 256 code IBM character set to be available. To ensure that it is able to do this, the ANSI emulation expects the **Strip Parity** option found in the Setup|Terminal dialog menu to be disabled. The Dialing Directory editor automatically disables parity bit stripping if you select ANSI emulation for a directory entry.

Odyssey Terminal Types

Debug Terminal

Some terminals offer a "debug monitor" or "display controls" mode, in which terminal control sequences and ASCII control codes are displayed rather than being acted upon. Odyssey provides a separate terminal emulation which does this. When loaded, this emulation will display standard ASCII abbreviations for all control codes it receives. Since a carriage return is not acted upon in this emulation the cursor always wraps when it reaches the right hand margin. Use this emulation if you need to know exactly what characters are being sent by a host, including control characters which are not normally printable.

Odyssey Terminal Types

PRESTEL (Viewdata)

This is the terminal emulation required in order to use the British Telecom **Prestel** online database system, and other systems which require Prestel compatible terminals. A Prestel terminal implements a subset of the CEPT Viewdata terminal standard. The Odyssey PRESTEL emulation is a much revised and totally integrated version of the Viewdata program by DORTEC DANMARK ApS. Since this emulation forms a major subsystem in its own right, the much longer description of is it made available elsewhere - see the Odyssey Prestel Emulation.

Prestel terminals have one feature which you may find confusing, which is that they do not communicate using the standard ASCII character set used by everyone else. The character set used is largely similar, but some characters have been moved around; for example, the '#' character occupies a position in the Prestel character set occupied by '_' (underline) in the ASCII table. This is unfortunate, since '#' is one of the most used keys in Prestel.

In most cases the character set issue is not a problem, because the Odyssey Prestel emulation performs the necessary translation. Where it is a problem however is in keyboard macros, and Transmit() commands from a script. The characters transmitted by these Odyssey features are not filtered by the terminal emulation, and so are not translated. This means that if on Prestel you normally type a command such as:-

***90#**

in order to put this into a macro or transmit command you first have to replace the hash with an underline, so the command as it appears in the macro is:-

***90_**

if you do this then macros and transmit statements will be accepted as valid input by Prestel.

Odyssey Features

Text Logging

Text Logging (sometimes called Text Capture), means creating a permananent record of everything received from the host computer which was displayed on the terminal. Odyssey provides several text logging modes, these are:-

- **Raw logging**, in which Odyssey makes an exact record of every character received.
- **ASCII logging**, in which Odyssey attempts to produce a pure ASCII form of the output, removing any control characters or terminal control sequences it finds.
- **Printer logging**, means directing the output of the logging function to a printer device attached to your PC. This does not prevent logging to disk at the same time if you wish. Printer logging may also be raw or ASCII mode.

Raw logging mode (whether for printer or disk) is controlled by an option available in the Setup|General dialog. The same option controls both disk and printer logging. In each case, logging is automatically suspended when a file transfer begins, and resumes again when it is completed. The following topics provide additional detail on the use of text logging in Odyssey.

What is Text Logging?
Raw Logging
ASCII Logging
Printer Logging

Odyssey Text Logging

What is Text Logging?

Text Logging (sometimes called Text Capture), means creating a permananent record of everything received from the host computer which was displayed on the terminal. In the old days of the teletype (see the last chapter), text logging was an everyday feature, since the paper roll maintained a permanent "audit trail" of all commands typed, and the output received, during every session on the computer.

When the days of the "glass teletype" arrived, text logging was no longer possible, since terminals no longer provided a permanent storage medium.

However, with the arrival of terminal emulation on Personal Computers, this possibility has returned, since these computers have access to both disk and printer storage, making it possible once again to create a permanent recording of an online session for later review. Storage on disk of these logs is especially useful, since it is possible for the text output to be searched for key words using an editor, or even to massage it into a form which can be loaded into a word processor or spreadsheet.

Odyssey Text Logging

Raw Logging

Odyssey defaults to Raw Logging disabled, to enable it you must toggle the option in the Setup|General dialog, or you can control the setting for individual services using the dialing directory.

When enabled, raw logging means that Odyssey will make no attempt at all to alter the data received from the host computer. This allows you to record, for example, the exact sequence of characters received by an Odyssey terminal emulation, a feature which allows you to play this data back later, using Odyssey playback mode. If Odyssey had cleaned up some of the control characters, then the log would contain nothing but printable ASCII characters, hence colors and so forth would not be visible when played back.

Remember that if Raw Logging is enabled, then printer logging will also be "raw". This means that if the data stream contains terminal control sequences, then unless your printer is rather specialised, these control sequences are going to be very confusing to it (note that the Odyssey terminal emulations support the printer redirection features of the terminal they emulate, so possibly providing a more useful form of printer output).

There is a trick you can use if you want to get a raw log *and* a cleaned up printer log, and that is to use online Raw logging to get a raw log file on disk. Once offline, you disable raw logging and use the Odyssey playback feature to review your raw log offline. In playback mode Odyssey does everything it normally would if it was receiving the data live, including text or printer logging! This means that you can play back the raw log with raw logging disabled, and printer output enabled, which produces cleaned up output on your printer.

Logging can be enabled in several ways. You can type **ALT+L** at the terminal window, and you will be prompted for a file name for the log. Unless you supply an alternative, Odyssey will always give the log file a .LOG extension. To close a log file simply press **ALT+L** again (ie. the command is a toggle). You can also enable logging automatically for any service checking one of the buttons in the "Logging" panel in the dialing directory entry for that service, you can also use another directory field to enter the name of the log file you would like (do not put a .LOG or other extension in that name). Finally, you can enable logging by using the **LogFile()** command from a script.

Odyssey Text Logging

ASCII Logging

ASCII logging is the opposite of Raw logging, in that Odyssey will attempt to produce pure ASCII text from an incoming data stream, whatever terminal control sequences or other characters the input contains. Colors and other text effects which cannot be represented as plain ASCII are lost.

However, you should be warned that cleanup is not always guaranteed to produce sensible results. A highly animated terminal display, eg. one in which the cursor is moved around a lot, and portions of the display are erased or overwritten, cannot be represented in a simple, "flat" ASCII form. For example, suppose that the host produces a nice looking display - so far so good, we can capture an image of that display for the ASCII log. Now suppose that the host positions the cursor and alters one character in that display - how do we represent that change in the log file? Do we store another complete image of the display? Suppose the host does that sort of thing a lot - how many copies of this virtually identical display do we need to store, and how useful would the resulting (vast) log file be?

In summary, the less complex the host display manipulations are, the better the results of the cleanup will be.

Odyssey Text Logging

Printer Logging

Printer logging directs output to a printer instead of to disk. This mode can be enabled from the file menu, or by using the Printer() command in a script.

☞ Note: In the DOS version of Odyssey, "printer logging" sent output directly to the printer in real time. This habit does not translate well to the Windows environment, where such long term hogging of a shared resource is quite unacceptable. So, Odyssey instead implements printer logging by first logging to a file, and then automatically submitting that file to the print server when the "printer logging mode" is closed. This of course means that you cannot view the printer log while it is open, which cannot be helped. This makes the Windows version of Odyssey unsuitable for "alarm printer" type applications, where the data stream to the log must be printed immediately.

Odyssey Features

Using Scripts

If you use Odyssey to perform a repetitive task, for example if you log on to the same BBS every day, and download messages from the same section, entering the same commands, then what you really need is a script. A script can be used to automate almost any interaction with the host. Scripts can range in complexity from simple logon scripts, or complex "blink" scripts designed to search the BBS for messages and download them as quickly as possible, in order to minimize connect charges. The following topics provide more details on why scripts are used, how to write simple scripts, and how to use scripts. This help section is aimed at non-programmers - see elsewhere for a detailed script language reference.

Using Odyssey Scripts

What do we need Scripts for?

Even though you are a Windows™ user, you are very likely to have used DOS in the past, and so you are probably familiar with the concept of "batch" files. These are text files, all with a .BAT extension so that DOS will recognise them, and all containing a simple sort of program which the DOS command interpreter can understand. DOS batch files consist mainly of ordinary DOS commands, although batch files can also use programming features such as IF, GOTO and REM statements. The purpose of the batch file is to make it easier for you to perform any repetitive task. Instead of entering individual commands to change directories, load a mouse driver, run a word processor, and change back to the root directory when you exit, you replaced all of the above with a single command which executes a batch file which does it all for you.

A comms package needs a script language for similar reasons. Many of the things you do inside a comms package are repetitive tasks which you perform (at least) daily. For example, you may log on to your friendly local BBS to check for any electronic mail which has arrived. If any has, then you might want to open a log file, capture the new messages, log off, and read the messages offline. If you use the same service often enough, then it will almost certainly be worthwhile for you to prepare a script to automate this procedure.

Using a script has at least these advantages :-

- Your hands are free for other work while the comms package gets on with its daily chore.
- Once a script is prepared it can be used by anyone, including those who have no knowledge of how to perform the task manually.
- Time wasting typing mistakes are eliminated.
- The script responds to prompts and types messages much faster than a human possibly could, which minimizes connect time.

However, there is one fly in the ointment, which is that a script can be badly affected by line noise. For example, if a script is waiting for a particular prompt, and that prompt is corrupted by spurious characters, then an indifferently written script could end up waiting all day for a prompt that has already passed by.

This is where error correction comes into its own. Whether you use the error correction provided by your modem (if any), or whether you use Odyssey's software MNP, the important feature in common is that it is no longer possible for a prompt to be corrupted by line noise, and so a breakdown such as the one described above can never occur. Of course, this requires that the host BBS supports MNP, but these days that is more and more likely to be the case. MNP cannot prevent a carrier loss, another potential cause of script failure, but that eventuality is usually much easier to cope with than the problem of noise. Error correction has made script programming reliable, and thus has turned it into an essential feature of a modern communications package.

So, you are now sold on the merits of a script language, however the nature of the script language which should be provided in a comms package is a matter for debate. Should we, the implementor of the package, go for power or simplicity in the script language? On the one hand a limited script language is usually easier for a beginner to learn, on the other hand our beginner will not be that forever, and sooner or later will come to resent the limitations of a "simplified" script language.

In Odyssey, we decided to go for power, but in such a way that beginners can largely ignore the wide range of features until the day comes when they are needed. A learn mode is provided which is capable of creating scripts automatically, requiring negligible understanding of the underlying language. From there you can progress to simple, manually created scripts using a basic set of four or five script commands, and finally, when you are feeling confident, you have the option to learn the complete script language.

Readers who are also programmers might like to know that Odyssey script closely resembles a conventional programming language, especially structured languages like C, Modula-2 or Pascal. If you have a familiarity with any of these, or with any other common language then you should find that you have no problems at all in getting to grips with Odyssey script programming, and you will be able to quickly progress to the expert level.

Using Odyssey Scripts

Scripts execute in the background

This topic is a brief note about how Odyssey executes a script.

In some packages, running a script takes over the comms package completely, blocking out everything the user does at the keyboard, except for the "cancel script" key, which is usually an <Esc>. In Odyssey, the script executes in "parallel" with the user, and so you are free to type Odyssey commands while a script is running, and to "help a script along" if a mistake in the script or a burst of line noise causes it to get stuck at a prompt. However, this is not true "background" operation, since entering any major Odyssey subsystem (eg. pulling down a menu, or opening a dialog box), will cause the script to pause until you return to terminal mode.

Using Odyssey Scripts

Using Learn Mode to create Scripts

Learn mode provides the easiest way for a beginner to create an Odyssey script. Typically this facility will be used to create logon scripts for a variety of BBS host services. All you need to know when creating such a script using learn mode is how to log on to that service manually. You teach Odyssey how to do it once, and thereafter Odyssey will be able to do the job for you.

The first step in using Learn mode to create a logon script is to create a dialing directory entry for the new service - see elsewhere for a description of how to do that.

When you create the dialing directory entry, you must put something in the "key" field. This key can be anything you like, up to eight characters long, but it should be unique to that entry (ie. until you are more experienced, you should make sure that the new key is not the same as that of any other entry). The purpose of the key is twofold: first it allows a script created by learn mode to identify the correct directory entry to use, secondly it allows a script to be attached to that entry so that the script is automatically started when you select that entry for dialing. A script is "attached" to a dialing directory entry when the script has the same name as the directory entry key.

For the sake of the following example, let us suppose that you want to create a script for dialing a BBS host called "ANYBBS". First, you create the dialing directory entry, making sure that you have the right settings for baud rate, parity, error correction and so forth. In the key field of the new entry you should type the word "ANYBBS" (without the quote marks). Having completed the directory entry, including the key, you then click on **OK** to leave the Edit Directory Entry dialog, and then click the **Close** button in the dialing directory dialog, taking you back to the Odyssey terminal window (make sure that it is the terminal window which is active after all of the above, and not any other Odyssey window).

Select the **Command|Learn script...** menu item. Odyssey will ask you for a name to give to the learned script - for this example you should give the name "ANYBBS" (without the quotes). The name given must match the directory entry key you entered earlier, so if you used a different key name, then you should also use that alternative at the script name prompt. You should now click the **OK** button in the "script name" dialog to start learn mode.

Learn mode is now active. From now on, everything you do will be noted by Odyssey, and duplicated in the new script - so you should avoid doing anything which is unrelated to the task of logging on to the service.

Press **ALT+N** to bring up the **dialing directory** dialog again, and double-click on the entry you created earlier (this should still be highlighted if you exactly followed the instructions given above). This causes Odyssey to begin dialing the number, so you should now wait to see if you get a connection.

On most host systems, the next thing you will see after a successful connection is the BBS logo, followed by a prompt for your user name (the name the BBS knows you by). You should enter that now. The BBS will then prompt you for a password, and again you should respond. You are now logged on to the BBS.

Odyssey now knows everything it needs to know in order to log on to that service, so assuming that is as far as you want to take it, you should now leave Learn mode by pressing <**Esc**>. Odyssey will convert what it has learned into a script file and write it to disk. If you have been following this example then the script file created will be called "ANYBBS.SCR" - all Odyssey scripts have the .SCR extension, in the same way that DOS batch files always have a .BAT extension (note that an Odyssey script created by learn mode is a text file which can be viewed in any ASCII editor, such as the Odyssey text editor).

Learn mode is now terminated. Odyssey is no longer remembering what you do. You can if you wish stay on the BBS to continue this session, or you may log off in order to test the newly created script.

The next time you want to call that service simply open the dialing directory dialog, and double click on the directory entry. Odyssey will automatically run the learned script, which will not only dial the number for you, but also enter your user name and password details, just as you did the first time, leaving you logged on to the BBS and ready to read messages or transfer files.

There are other ways of running scripts in Odyssey, a subject which is discussed later on (see the section "How to Run a Script").

If, during the learn procedure, something goes wrong with the login process, for example if you get a busy signal, then you should abort learn mode by pressing the <**Esc**> key, then later on you should start again from scratch. Whatever you do, you should ***not*** attempt to redial the number without first stopping and restarting learn mode - if you ignore this warning you will have dialed the number twice, which is exactly what the learned script will do when you run it!

Limitations of Learn Mode

For most systems, Odyssey learn mode will do a perfectly good job of automatically creating a login script for you. However, some systems may cause problems if the timing of responses is very critical (strict timing problems).

Strict timing problems occur when there is a minimum delay which must occur between a prompt and a corresponding reply. Odyssey is not able to differentiate between the delay of a slow novice typist, and a delay deliberately introduced by the typist because he/she knows that the system will not react correctly if the response comes too quickly. The British Telecom PSS service is an example of a system in which you must wait a second or two after a connection is established before you can start the login process. If the delay inserted by Odyssey is too short then you may have to increase it, by manually editing the script. For that you can use the built in Odyssey editor, or else use any preferred external editor, provided that it is capable of writing a clean ASCII text file. Look for the "**Delay(xx)**" commands inserted into the script by learn mode, and increase the number shown between the brackets (**xx** is a delay in seconds). Remember to save your changes; if you use the Odyssey editor then the save key is **F2**.

Using Odyssey Scripts

Creating your first Script by Hand

For the slightly more adventurous, this section describes how to create scripts manually, using a minimal understanding of the script language, but providing a little more flexibility than is possible with learn mode. Reading this topic will also allow you to better understand scripts created by learn mode, should you need to modify one.

A script is created with a text editor; probably the most convenient editor would be the one built into Odyssey (accessed via **File|Open...**), however you are free to use any external text editor provided that it writes clean ASCII text files. Use of the Odyssey text editor is described elsewhere; if you are not familiar with it then you should read that section first, before delving into the script language. A script file must have an extension of **.SCR** before it will be recognised by Odyssey, and furthermore it must have the same name as a dialing directory entry key if the script is to be invoked automatically when that directory entry is selected. You are not required to attach scripts in this way, since scripts can also be invoked from the terminal mode **Command** menu, or named on the Odyssey command line. Remember to put the script file in the correct directory, so that Odyssey can find it. This directory is the place where the main program file WINODY.EXE was loaded from, or the alternative directory named in the Setup|General configuration dialog. Storing the script anywhere else will make it rather awkward to use.

The Odyssey script language currently provides around 200 built in commands, yet it is quite possible to write useful Odyssey scripts using only four or five of them. However, first you must learn rule number one, which is that all Odyssey scripts start from the following minimum outline:-

```
SCRIPT myscript;
BEGIN

END;
```

This "skeleton" is what Odyssey expects from any script, and its purpose is to allow Odyssey to easily confirm that this file it is trying to use is indeed a script, and helps it to more easily find where a sequence of commands begin and end. The "myscript" name need not be literally as shown, in fact it can be almost anything you like (its purpose is to give the script a meaningful name which will be understood by a person reading a printed listing). However, you should perhaps leave it as shown until you become familiar with the rules surrounding the naming of things in the Odyssey script language. Note the semicolons after the "myscript" and after the "END" are required. The words "SCRIPT", "BEGIN" and "END" are shown in upper case for clarity only. You can enter them in lower case if you prefer.

Once you have created the basic script skeleton as shown above, you may then enter commands between the BEGIN and END lines. The basic four commands you will use are **Dial**, **WaitFor**, **Transmit** and **Delay**.
These commands are described below.

Dial()

The **Dial** command is used to dial a number, taking the necessary details from a dialing directory entry. In order to use the dial command you must pass it the name of a directory entry key, and that key must match an existing directory entry, otherwise the dial command will not be able to proceed. Here is an example of the dial command:-

```
Dial("ANYBBS");
```

Note that the key name is surrounded both by double quotes and round brackets, and that the command is followed by a semicolon. This may seem like excessive punctuation, but once you get around to learning the complete language you will find out why they are required. For the moment you should simply follow the rules.

In the above example, the script would look up a dialing directory entry with a key of "ANYBBS", and would dial that number. The script will resume from the statement following the Dial command once a connection has been established.

WaitFor()

WaitFor is a basic command which you will use a lot, however expert you become. This command tells the script to wait until a particular sequence of characters (called a "string") arrives from the host. Here is an example of the waitfor command:-

```
WaitFor("User name?");
```

Note again the use of double quotes, round brackets and the terminating semicolon. In this example the script will wait until the string "User name?" arrives from the host.

You can also, optionally, add a timeout to the WaitFor command. This would mean that instead of the WaitFor command waiting forever for the string to arrive, it can terminate either when the string arrives, or when the timeout period has elapsed. This is an example of the WaitFor command making use of a timeout:-

```
WaitFor("User name?",10);
```

In the above example the WaitFor will terminate when the string "User name?" arrives from the modem, or when a ten second period has passed.

Transmit()

The Transmit() command tells the script to transmit a string via the serial port. Here is an example of the transmit command:-

```
Transmit("John Smith|");
```

In this example the script would transmit the characters inside the double quotes to the modem, and hence to the host BBS. Note the '|' character which forms part of this string. This is treated as a special character by the transmit command, in that it is not transmitted literally, but instead tells the transmit command to send the code for newline (usually the carriage return character). This is required because if you were typing this manually, in most cases the host would require you to press <Enter> to complete the answer to the prompt.

Delay()

This is the last of the four basic commands. The purpose of this command is to insert a delay into the script, usually to avoid sending messages to the host BBS before it is ready to receive them. This command is given a number which is the delay you require, in seconds. Here is an example of the delay command:-

```
Delay(2);
```

In the above example, the script would pause for two seconds, and would then proceed to the next command.

Putting It All Together

Now that you know what the basic commands are, we can proceed to put them together in the form of a script to log on to an imaginary BBS.

In the following example the script will dial a BBS which has an entry in your dialing directory with a key of "ANYBBS". The script will then wait for the username prompt from the BBS and enter a name of John Smith, then it will wait for the password prompt, and answer it with "smithy". In both cases it will follow the characters in the reply with a carriage return code.

```
SCRIPT myscript;
BEGIN
        Dial("ANYBBS");
        WaitFor("User name? ");
        Transmit("John Smith|");
        WaitFor("Password? ");
        Transmit("smithy|");
END;
```

Note the indentation of the lines containing the commands. This is not a requirement of the script language, but in many people's opinion it does make the script look nicer, as well as making it a little easier to follow. This is a good habit to get into early, since readability becomes very important once you begin to write larger scripts using the advanced script features.

These four basic commands should enable you to write a login script for any service. However, such a script does not deal with errors which might occur (such as an engaged tone). If that happens then you should press the <Esc> key to abort the script. The next section contains information on how to have the script recognise errors itself, however you needn't read that now unless you are feeling confident!

Dealing With Errors

Although the example script shown above will work perfectly in most cases, it does have a problem, in that it makes no attempt at all to cater for those occasions when things do *not* work perfectly. For example, suppose the number dialed is engaged - the script shown does not check whether the dial attempt succeeded, so succeed or fail, the script will go on to execute the next statement. The next statement is a WaitFor, and that is never going to succeed if the dial attempt failed, because Odyssey at that time is not even connected to the host. We therefore need to put some error protection into the script, which we can do using an **IF** statement.

An IF statement can be used to test whether a command succeeded. You cannot test the success of all commands, because not all of them can fail. However, among the commands you can test are DIAL and WAITFOR.

Here is the example script again, but this time we test the success of the DIAL command using an IF statement. If the dial command is not successful then the script stops:-

```
SCRIPT myscript;
BEGIN
      Write("Calling the ANYBBS BBS Host.|");
      IF Dial("ANYBBS") THEN
          WaitFor("User name? ");
          Transmit("John Smith|");
          WaitFor("Password? ");
          Transmit("smithy|");
```

```
        END;
    END;
```

Note the extra END which has appeared. The IF statement always has an END associated with it in the same way that a BEGIN always has an END, and it serves the same purpose, ie. to "bracket" a sequence of commands. In the above example the sequence of commands between the "IF" line and the first "END" will only be executed if the dial attempt succeeds. You could if you wished put other commands before the "IF" line, or following the first "END" line, and these would not be dependent on the "IF" test. In other words those commands would be executed whether or not the dial attempt succeeds. This is how you control the execution of commands in the Odyssey script language.

The **Write()** command is an example of such a command which will always be executed when you run this script (the Write command has not been previously discussed, it simply displays a message on the terminal window, but does not transmit that message to the modem).

Finally, we would like to cope with the situation that occurs when you get a connection with a modem, but the remote computer does not respond. To handle this, you can again use an IF statement, this time using it to test the success of a WaitFor command. Since we now want the WaitFor to give up after a certain period, we will specify a timeout this time - ten seconds in the following example:-

```
    SCRIPT myscript;
    BEGIN
        Write("Calling the ANYBBS BBS Host.|");
        IF Dial("ANYBBS") THEN
            IF WaitFor("User name? ",10) THEN
                Transmit("John Smith|");
                WaitFor("Password? ");
                Transmit("smithy|");
            END;
        END;
    END;
```

Semicolons

We should perhaps return to the question of the placement of semicolons. In the first script example there was a semicolon following the first use of WaitFor, yet there was no semicolon after the first WaitFor in the most recent example shown. Also, there is no semicolon on either of the two "IF" lines. If you study the above script you may be able to spot the common factor for yourself, which is that a semicolon tells the script processor where a complete statement ends. Let us take the last script from top to bottom - "SCRIPT myscript" is a complete statement, so it is followed by a semicolon. "BEGIN" is not a complete statement, since it must have a matching end, so it is NOT followed by a semicolon. The "Write" line is a complete statement, so there is a semicolon. The first "IF" line is NOT a complete statement, since like the "BEGIN", it must have a matching "END" - and so on down the list. In the case of the first "WaitFor", it is not a complete statement, since it forms part of an "IF condition THEN" sequence, whereas an unconditional "WaitFor" is a complete statement, and so is followed by a semicolon.

The semicolon in the Odyssey script language serves more or less the same purpose as the full stop which terminates a written sentence in the English language. The only real difference is that the script language grammar allows any "sentence" to be constructed from any number of smaller sub-sentences, each of which is also terminated with a semicolon.

Take all the time you need to understand this section of the manual. If all was not clear the first time you read it then you are encouraged to read it again. When you have read and understood the above discussion you will know everything you need to know in order to manually create your own login scripts for any BBS, and have them cope with almost any eventuality.

Using Odyssey Scripts

How to Run a Script

Other sections of this user guide have already given information on how to run a script. However, for reference, here are all the different methods gathered in one place. There are in fact many ways in which execution of a script can be accomplished.

From the Terminal Command menu.

You can run a script from the terminal window **Command** menu. Press **ALT+C** while the Odyssey terminal window is active, and the Command menu will be displayed. This menu lists the names of scripts which Odyssey has found in its home directory (the directory where WINODY.EXE was loaded from). If there are more scripts available than can be displayed in the menu (nine), then an extra menu option called "**More scripts**", which if selected, displays a dialog containing a full list of all the available scripts. To execute any script simply select it from the **Command** menu or the "more scripts" dialog.

From the Dialing Directory.

When you press <Enter> on a dialing directory entry in order to dial a BBS, Odyssey will check for an "attached" script, and will load and run one automatically, if it finds that an attached script exists. Odyssey considers a script to be attached to a dialing directory entry when the name of the script is identical to the dialing directory entry key. For example, if the key field for the chosen entry is "ANYBBS", and a script file exists called "ANYBBS.SCR", then Odyssey will load and run "ANYBBS.SCR" after dialing the number. What happens from that moment is entirely up to the script, but in most cases it will proceed to log on to the host service.

From another Script.

A script has the facility to chain to any other script. It does this by making use of the script Chain() command. The script which used the chain command is halted, and the chained script (if found) is executed. Odyssey has no facility to "call" another script, ie. with control returning to the original script, however one possible approach if you need to do that would be to explicitly chain back to the parent script from the child.

From the Command Line.

You can execute Odyssey scripts directly from the Odyssey program command line, by supplying the name of the script as a command line argument to Odyssey in the Program Manager "Properties" dialog. For example, the command line :-

```
    C:\WINODY\WINODY.EXE   ANYBBS
```

would tell Odyssey to execute the script "ANYBBS" immediately after loading. Odyssey does not display its normal welcome banner when a script is named on the command line.

Using Odyssey Scripts

How to Stop a Script

Pressing the <Esc> key while a script is running will cancel the script, unless the script has used the special CanEscape(FALSE) command to disable that feature. If you want to type and transmit an Esc character while a script is running, without aborting the script, then a keyboard command has been provided for that purpose - try **ALT+F2**.

Odyssey Features

Host mode

For active communications users, there sometimes arise certain occasions when you want to make a file or files on your PC available to a caller. In order for the caller to be able to dial into your PC you must operate as the host, just like the host systems you are used to calling, and it is in that situation that Odyssey **Host Mode** is required.

Odyssey Host Mode

What is Host Mode?

If you are a typical comms user, you normally just use Odyssey as a terminal package. In other words, you use a direct connection or a dialed link to talk to a remote host, from which you receive messages, and also upload and download files.

For active communications users, occasions sometimes arise when you want to make a file or files on your PC available to a caller. In order for the caller to be able to dial into your PC you must operate as the host, just like the host systems you are used to calling, and that is when Odyssey ***Host Mode*** is required.

Odyssey host mode provides a very simple BBS-"like" facility on a PC equipped with an auto-answer modem. Our host mode provides only the most basic of features, since it is not intended to be a full blown BBS. You would typically use this feature if you want a caller to be able to upload or download a file directly to your PC - the caller might often be yourself, calling the office PC from home to fetch a spreadsheet file, or vice versa. Odyssey host mode is not like the "remote control" packages which are now becoming popular - Odyssey host is designed so that the caller can use any package, not just another copy of Odyssey.

Odyssey host mode can enable error correction, including the use of Odyssey's software MNP. So, if the caller also has Odyssey, or an error correction equipped modem, then you can also have the benefit of error free connections when running as a host. You will need to enable the "**Error correction...**" option in the Setup|Host dialog if you want to use this feature.

By its very nature, a host mode is much more demanding of the modem connection than standard terminal mode. For one thing, your modem must provide an auto-answer capability. You can check this in your modem manual. Odyssey comes already configured to enable the auto-answer feature on Hayes compatible modems when you act as the host, which is not to say that all Hayes modems support it. The modem *must* also be configured so that it does not hold the carrier signal high. If it does, Odyssey will never be able to recognise when a caller connects. If Odyssey displays a message "Call in Progress" as soon as you enable host mode, then your modem has the carrier signal held high, and will need to be reconfigured.

Host mode in Odyssey is handled by a script, which has the unusual name of **ODYHOST.HSC**, but which is otherwise unremarkable. Source for this script is provided on the distribution disk (name as mentioned), so you could in fact modify the current script to your own requirements, or even replace it entirely with a more sophisticated script of your own design. The paragraphs in these help topics document the operation of the standard script.

When you enter host mode, Odyssey sends the "Enable Auto-Answer" command string to the modem, and then starts monitoring the **DCD** signal (DCD = Data Carrier Detected). This signal comes from your modem, and tells the PC that a connection has been established with a remote modem. When Odyssey sees this signal change, it knows that a call has started, and so it enters the interactive host mode.

The interactive host mode starts by asking the caller to enter a password. Odyssey knows two passwords (configured in Setup|Host), and the password selected by the caller determines what type of access that caller will have, ie. *privileged* or *non-privileged*.

Non-privileged callers are restricted to one directory only, and the only commands they can request allow them to view the contents of that directory, upload or download a file, or log off.

Privileged callers have an extra "change directory" command, which gives a privileged caller almost complete access to files on your machine. You should therefore be extremely careful about to whom you allow privileged access. Do not feel that *not* providing this level of access to a friend is either mistrustful or impolite - even the best of friends have accidents, and it is better that if he is going to accidentally delete some files, that those files be on his own machine rather than yours! For most people, the only privileged caller will be the owner of the PC in question.

Once the caller has entered a password, he is then free to upload and download files, plus use other commands for which he has sufficient access privileges. When the caller is finished, he selects the "Goodbye" command, and Odyssey drops the line. If necessary, Odyssey restores the modem to the default baud rate, and is then ready to receive another call.

You leave host mode by pressing the Esc key, at which point Odyssey sends the "disable auto-answer" command to the modem.

Odyssey Host Mode

Preparing to use Host Mode

Before you can use Odyssey host mode, you should be very careful about configuring your machine correctly.

First, you should enter the Setup|Host dialog and study the options there. In particular, you should make sure that you have changed the default passwords to something only you know, otherwise any Odyssey user will know what those passwords are, and will be able to gain full access to your machine.

You may also change the welcome messages and so on, but this is not critical, especially if you are experimenting with host mode for the first time.

The next thing to check is the current baud rate. This is shown on the Odyssey status line (but *not* when in host mode). You should make sure that your baud rate is set to the highest speed your modem is capable of, prior to entering host mode. The reason for this is that most modems will not allow a caller to connect at a speed higher than the current baud rate between itself and the PC. So, if you had just finished a call to a 1200 baud service, and then entered host mode without resetting the baud rate, then your modem will only answer calls at 1200 bps or less, even though it might be capable of much more! This is a feature of the modem, and not something that Odyssey can do much about.

Next, if you want to allow error correction for host mode callers, then you should remember to enable the error correction checkbox in the Setup|Host dialog.

Finally, if you would like Odyssey Host mode to adapt the speed of the PC to modem link to match the physical connection with the remote modem, then you should enable the "**Baud Rate Detect**" feature in Setup|General. Note that not all modems like the terminal to switch baud rates after a connection is established. Some modems do not allow it at all, while others may need to have a switch setting changed before it is allowed.

Odyssey Host Mode

Host Mode in Use

This section describes how Odyssey Host mode appears to a caller.

The caller is first asked for a password, as mentioned previously. Assuming that he enters the Privileged user password, he will then be presented with the following simple menu.

C)hange dir, F)iles, U)pload, D)ownload, G)oodbye?

Pressing the letter **C** invokes the change directory option. The user is asked for a directory name, and when one is entered, Odyssey makes that directory the current "host" directory. This command is available only to privileged callers.

The Files command, selected by pressing the **F** key, causes Odyssey host to prompt for a file mask. The mask is a standard DOS file specification, and allows wildcards. When the specification has been entered Odyssey host will list all the matching files.

The Upload and Download commands allow file transfer. Odyssey will prompt for the name of the file to be transferred, and the protocol to use. Remember that in this case, the host mode is using the **Upload** and **Download** terms from the point of view of the caller, not the Odyssey host.

Finally, the Goodbye command is selected by pressing the **G** key, and this causes Odyssey host to say goodbye to the caller, and then hang up the phone.

When host mode is in use, a user watching the host PC can see everything that happens. Also, the review editor can be used to view previous transactions which have scrolled up the top of the display, and text logging can be used to keep a permanent record. You will need to leave host mode before you can view the review buffer or log file however. Host mode is terminated by pressing the <Esc> key on the keyboard of the host computer.

Odyssey Features

Chat mode

Chat mode is enabled by an option on the terminal mode **Window** menu.

Chat mode is a real time mode which allows you to "talk" directly to another modem user via the keyboard, much as you would talk to that person on the telephone (except chat mode is slower, and more expensive). Chat mode is of genuine use in two situations, firstly, when the host software allows, it provides the means to chat directly with another caller to a multi-user BBS system. Secondly, it is sometimes useful when a caller calls your PC directly, and you want to exchange a few brief words prior to, or just after transferring a file.

When Chat mode is entered the terminal window is split into two panels, called the "Local" and "Remote" panels.

The Remote (top) panel is where any received characters from the remote modem are displayed. These will normally be characters typed by the other party, but may also include echoes of text which you have typed.

The Local (lower) panel is used to display anything that you type on the local keyboard.

Press the <**Esc**> key to leave chat mode.

Odyssey Features

Playback mode

A "**Play back log file**" option may be found on the terminal mode Command menu. This allows you to "play back" offline, text which has been previously logged online. This is most useful for playing back data intended to drive one of the Odyssey terminal emulations such as Prestel or ANSI, but if you do so it is important that these logs should have been created using Raw Logging Mode.

Odyssey knows nothing of which terminal emulation is required to play back a log correctly - you must select the correct emulation yourself before playing it back.

When playing back a log file, Odyssey emulates the baud rate currently selected in Setup|Comms, so you can speed up or slow down the playback by changing the baud rate accordingly.

To pause a playback simply press the space key (in fact, any key other than <Esc> will have this effect). To cancel the playback you should press the <Esc> key.

When in playback mode Odyssey treats all the characters it displays as if it had just received them from the serial port. This means for example that you can enable text logging or look in the review buffer. If **Raw Logging** is disabled when you play back a raw log file, then the effect is to clean up the log file producing a clean ASCII equivalent. For this to work you must not give the ASCII log file the same name as the raw log file, otherwise the result will be one badly mangled file, and probable deployment of the DOS CHKDSK utility.

Odyssey Features

Command line recall

Odyssey implements a command line recall (or "command history") feature, a la the DOSKEY utility in DOS 5, or the popular public domain DOSEDIT. Currently, the command line recall facility is a feature of Odyssey's internal TTY terminal emulation, so the feature will not be available if you aren't using TTY. Use the Up-Arrow key to recall a previous command, and then use the normal text editing keys to modify it, if necessary, before pressing <Enter> to retransmit the command.

Note that once you enter the recall facility you have entered a *mode*, during which characters arriving from the serial port will be ignored. It is obviously not healthy for this to last too long, so if there is no keyboard activity for 20 seconds, Odyssey will automatically exit from the command recall/edit mode and return to normal terminal operations.

There is no fixed limit on the number of commands in the command history. Instead, a 1k buffer is allocated to the task, the limit being reached when that buffer is full. After that, one or more of the oldest commands will be discarded each time a new one is added.

Odyssey Features

Event Logging

Some other packages call this feature *Call Logging*, but that is a misnomer, since the feature is not only used for keeping a record of calls. To enable event logging you should turn on the **Event logging** option of the Setup|General dialog. There is also a script command called **EventLogging()** which can enable and disable this feature.

In the current implementation, Odyssey logs the start and end of a call, plus the completion or failure of a file transfer (all of the above with the appropriate statistics, naturally). Also, a script can record its own events using a general purpose **LogEvent()** command.

All events are logged to a file called **ODYSSEY.REC** (RECord). Currently, no substitute file can be used.

Odyssey Features

Keyboard Mapping

Note: This feature was provided as an external utility - MAKEKEY - in the DOS version of Odyssey. In Windows Odyssey the keyboard template compiler is built directly into the main application, since Windows Odyssey doesn't have the memory constraints of DOS Odyssey.

Odyssey provides an internal facility which is used to create keyboard templates, ie. a table which maps keys (or key combinations) onto replacement keys or keyboard sequences. The replacement keys may be either function keys (so you can perhaps tell Odyssey to treat F8 the same as PageUp), or the replacement might consist solely of ASCII characters, which will then be transmitted to the remote host. In short, a keyboard template allows you to configure Odyssey such that any key combination can be substituted with a user defined string of characters.

Odyssey accepts as input a text file with the extension **.KDF**, which describes the keyboard mapping, and it outputs a file with the extension **.KEY**, which you can use elsewhere within Odyssey, perhaps by loading them explicity at the terminal window Command menu, or load them from a script by invoking the **LoadKeyDef()** script command.

Keyboard definitions are also loaded automatically if its name matches the name of a terminal emulation being loaded, that is, if you load a terminal emulation called XYZ.TRM, and a keyboard definition file exists called XYZ.KEY then the keyboard file will also be loaded. This allows you to produce your own variants of the standard terminal emulations for use with special hardware, eg. protocol converters. You would normally copy the standard emulation to a new name, eg. VT100.TRM to VT_TSO.TRM, and prepare a keyboard definition called VT_TSO.KEY. Doing it this way ensures that the keyboard definition is not loaded when you the VT100 emulation to behave like the standard terminal.

The format of the keyboard definition source file is very simple, it is just a text file with one definition per line. The source file may also contain blank lines, and lines starting with a semicolon ';' are ignored, so such lines may be used for comments.

Below are some sample definitions:-

```
<CtrlF1>      =     <Esc>[0~
<CtrlF2>      =     <Esc>[1~
<CtrlF3>      =     <Esc>[2~
<CtrlF4>      =     <Esc>[3~
<ShiftF1>     =     <AltS>E
<AltShiftE>   =     XYZ123
```

Each line contains the key combination to be defined, an equals sign, and then the definition itself, which is made up of one or more key symbols (a letter or a key combination). There should be at least once space either side of the '=' character.

A key combination starts with '<' and ends with '>'. In between should be a key name optionally prefixed by '**Ctrl**', '**Alt**' or '**Shift**', or any combination of those. There are some valid key definitions:-

```
<ShiftTab>
<AltShiftTab>
<ShiftAltTab>
<ShiftCtrlTab>
<<>
```

Key combinations can contain the "shift keys", ie. '**Alt**', '**Shift**' and '**Ctrl**' in any order, but the combination must end with a key name (the shift keys are not sufficient on their own).

Table 1 lists the function key names recognised by the Odyssey keyboard template compiler.

Note in particular that enhanced keyboard function keys (Home etc) are mapped separately from the old equivalents on the numeric keypad. This allows you to assign different replacement keystrokes to each. Once you have prepared your source definition file you then run the keyboard template compiler, eg:-

Make sure that the Odyssey terminal window is the active window, then pull down the Command| Compile... menu option. The file selector dialog that follows will by default assume that you want to compile a script, and will therefore show you any available .SRC files in the current directory - ignore that and instead pull down the "**Show files of type...**" listbox, and select the ".KDF (Keyboard definition files)" entry. Now you see a list of KDF files, if you created any. Select the one you would like to compile (eg. *mydef***.KDF**). Provided that there are no errors in the kdf definition then Odyssey will create a file called *mydef***.KEY**. If there were any errors then Odyssey will open up a text editing window, and will use the text cursor to point to the error location - watch the status line, as this will contain a description of the problem Odyssey is complaining about.

Another way to compile a keyboard definition file is to load that file into an Odyssey editor, then press **F9**.

Note that Odyssey's key substitution is not nested, ie. a definition such as:-

```
<Esc>        =      <Esc>
```

does nothing, it does *not* produce an infinite loop. In other words keystrokes described on the right hand side of the '=' are the actual keys generated by typing the key combination on the left - none of the keystrokes on the right are expanded. To further clarify this, here is another example:-

```
<F1>         =      ABC
<F2>         =      <F1>
```

In the above example, pressing <F1> generates the string "ABC", but pressing <F2> generates <F1>, and *not* "ABC", despite the prior definition.

Table 1: Recognised Function Key Names

These are the standard function keys:

F1	F2	F3	F4	F5
F6	F7	F8	F9	F10
Esc	Tab	BS	UpArr	DownArr
LeftArr	RightArr			
Ins	Home	PgUp	Del	End
PgDn	Enter			

The following keys are all generated on the keypad when NUMLOCK is in effect. Since the Odyssey keyboard template compiler distinguishes these keys it is possible to have '7' on the keypad generate a different sequence from '7' on the main keyboard.

K0	K1	K2	K3	K4
K5	K6	K7	K8	K9
KDot	KMinus	KPlus		

The following keys are only useful if you have an Enhanced Keyboard, they include three further keypad keys (KSlash, KStar and KEnter).

F11	F12	ELeftArr		
EHome	EEnd	EUpArr	EDownArr	
EPgUp	EPgDn	ERightArr		
KSlash	KStar	KEnter	EDel	EIns

You should also be aware that the keyboard template compiler makes no attempt to verify that a key combination you define can actually be generated by your keyboard or BIOS. For example, the keyboard on the machine used to prepare this manual does not generate a keystroke when CTRL+ALT+F1 is pressed, but Odyssey will not prevent me from using this combination in a definition, eg :-

<CtrlAltF1>   =   XYZ

Odyssey Features

Script Compiler

Odyssey always compiles a script before running it, ie. Ody checks that the syntax is correct, and then converts it into a encoded form which can be executed more efficiently. This fact isn't necessarily obvious to the user however, since Odyssey normally performs this task quickly and transparently, each and every time the user asks Odyssey to load an .SCR file.

The Odyssey script compiler is fast, but not infinitely so. In other words there will always be a certain pause for the compile step during script loading, which for most scripts will be so brief that users will never notice it. However, with *very* large scripts this pause might begin to be noticeable, and perhaps irritating. In that case, you might like to take advantage of the fact that Odyssey scripts may optionally be stored on disk in a precompiled form, which gets rid of the compilation overhead while retaining the execution speed advantages of using a compiled rather than interpreted script language.

As briefly mentioned above, compiling involves converting the script into an encoded form; for some users this fact might be the main reason for wanting to store scripts in precompiled form. For example, technical support personnel might want to distribute Odyssey scripts to users within their company in this encoded precompiled form in order to prevent those users from fiddling with the script, and possibly breaking it.

You can create precompiled scripts in a number of ways, which are detailed below.

Preparing the script.

The first step is of course, write your script! However, instead of saving the script source file with the usual .SCR extension, you should save it with a .SRC extension instead (SRC is short for SOURCE).

If the script is already written then you can simply rename it with an .SRC extension; there are no syntax differences between normal and precompiled scripts.

Compile the script

There are two ways to compile a script from within Odyssey. The first is to select the Command|Compile... option from the terminal window menu, and then enter the name of the .SRC file you want to compile. Odyssey will compile the selected script and store the resulting .SCR file in the same directory, unless the source contained syntax errors, in which case Odyssey will display an error message and give you the opportunity to correct the error.

Alternatively, if the script source is loaded into a text editor window then you can compile the script by pressing the **F9** key, or by selecting the Command|Compile... option from the text editor menu. Again, an .SCR file will be written if the script is successfully compiled, or a syntax error will be highlighted if not.

Once the encoded .SCR file is created it can be used just like a normal .SCR file, except of course that you should not attempt to edit it, since the precompiled form of a script is not a text file.

Compiling a Script from DOS

You may also compile an Odyssey script from the DOS command line. This feature is provided for those old fashioned programming types who prefer command line versions of compilers and other similar tools. Seriously however, an external DOS version of the script compiler is of course necessary if you, for example, want to automate compilation of a number of scripts using a DOS batch file.

The external, DOS version of the script compiler may be found in the Odyssey directory as the utility called **ODYCOMP.EXE**. To compile any script simply run OdyComp, giving the name of the .SRC file to be compiled on the command line. For example :-

```
C:\path> odycomp myscript.src
```

Will compile "myscript.src", generating a file "myscript.scr" in the same directory.

Odyssey Features

Alternative Configurations

The DOS version of Odyssey had a feature whereby it was possible to launch Odyssey from different subdirectories, each such subdirectory containing a different ODYSSEY.CFG file. This allowed users to have (eg.) different configurations for talking to different modems. A similar, though not identical feature is available in Windows Odyssey.

It is anticipated that most users of Windows 3.x will launch Odyssey by clicking on an icon, selected from a *Program Manager* group. Program Manager does not offer any explicit "current directory" with which to associate alternative configurations, but an equivalent is possible through the use of multiple icons, each of which would have different options in the command line parameters section.

To force Odyssey to use an alternate config file, you need to add an option "**-S<filename>**" to the Odyssey command line set in the "**File|Properties...**" dialog in Windows "Program Manager". For example :-

```
c:\winody\odyssey.exe -sMYCONFIG.CFG
```

forces Odyssey to load and save its configuration data to MYCONFIG.CFG instead of the standard ODYSSEY.CFG. Note that this alternate name applies for the entire Odyssey session; there is no command provided which allows you to change config files in mid session.

Note that there should be no space between the '-s' and the filename. The filename can specify a complete path, or it can consist of just the tail part, in which case the config file will be stored in the Odyssey directory - in which case (of course) the tail name should not conflict with the standard name of ODYSSEY.CFG.

If you also specify a script on the Odyssey command line this may come either before or after the config file option, eg :-

```
c:\winody\odyssey.exe -sMYCONFIG.CFG myscript

c:\winody\odyssey.exe myscript -sMYCONFIG.CFG
```

both of these examples do the same thing, ie. load setup information from MYCONFIG.CFG, and then run the script 'myscript.scr'.

Odyssey Help

Odyssey Menus

Each of the MDI document windows in Odyssey has its own private menu bar. Please select the menu whose help topics you would like to view.

Terminal Window Menus
Text Editor Menus
Directory Viewer Menus
Bitmap Viewer Menus
Fax Server Menus
Archive Viewer Menus

Odyssey Menus

Terminal Window

These are the menus available when the terminal window is active.

File menu
Edit menu
Setup menu
Upload/Download menus
Command menu
Window menu
Help menu

Odyssey Menus

Terminal Window File Menu

It is conventional for all Windows applications to have a **File** menu, and for that menu to be the first item on the main menu bar. The File menu is used when loading and saving documents, printing, and other operations which concern access to files or directories.

File|Open...
File|Close all
File|Save
File|Save as...
File|Print screen
File|Text logging...
File|Printer logging
File|View directory...
File|DOS Shell
File|eXit from Odyssey...

Odyssey Menus

File|Open...

The terminal window **File|Open...** menu item is used to load a text file into an Odyssey text editor. The user is presented with a standard File Selector dialog from which to make his choice of files, and a text editor window is then opened to display and edit that file. The filename entered into the dialog need not already exist - if the file does not exist then a new file of that name is created.

Odyssey Menus

File|Close all

The **File|Close all** menu option closes all currently open document windows (directory viewers and bitmap viewers, text edit windows etc). This is quicker than closing each window individually.

Odyssey Menus

File|Save

The **File|Save** menu item is not relevant to the terminal window, and is therefore disabled.

Odyssey Menus

File|Save as

The **File|Save as** menu item is not relevant to the terminal window, and is therefore disabled.

Odyssey Menus

File|Print screen

The **File|Print screen** menu option converts the current contents of the terminal window into a bitmap, and then submits that bitmap for printing by the Windows print server.

The Setup|Printer configuration dialog contains an option called "**Convert screen dumps to monochrome**". If this option is checked then Odyssey will attempt to reduce the colors in the bitmap to black and white only. If the option is not checked then a color bitmap is passed to the printer driver, which will normally print the bitmap using gray halftones (unless a color printer is used).

Odyssey Menus

File|Text logging...

The **File|Text logging...** menu option is used to start or stop Odysseys text logging (alias: text capture) feature. The option is a toggle, ie. if text logging was inactive, then selecting this menu option will turn on text logging. If text logging was in progress then selecting this menu option closes the log file and ends the text logging mode. You can use **ALT+L** as a shortcut for this command.

If text logging is about to begin (it was not already in progress), then Odyssey displays a standard file selector dialog, which prompts the user to enter a name for the new log file. If the user enters the name of a log file which alreadys exists then text will be appended to that log file, otherwise a new log file is created.

Odyssey Menus

File|Printer logging

The **File|Printer logging...** menu option is used to start or stop Odysseys print logging feature, whereby text displayed on the terminal is also copied to the printer. The option is a toggle, ie. if printer logging was inactive, then selecting this menu option will turn on printer logging. If logging was in progress then selecting this menu option ends the printer logging mode.

Odyssey displays a "Printer Logging" icon on its desktop whenever printer logging is active. Another way for the user to close a printer log is to click on this icon and then select "Close" from the system menu which the icon displays.

☞ Odyssey does not physically print the data in the "printer log" until printer logging mode is closed, since it is not acceptable for Windows applications to "own" such shared resources for extended periods. It is therefore not possible to use Odyssey for printer streaming applications where it is important for the data to be printed immediately.

Odyssey Menus

File|View directory...

The **File|View directory...** menu item causes Odyssey to open a directory viewer window, which allows the user to copy, rename and delete files etc. See elsewhere for a detailed description of the Odyssey Directory Viewer module.

Odyssey Menus

File|DOS Shell

When the **File|DOS Shell** menu item is used, Odyssey asks Windows to run a copy of the DOS command interpreter. This can be useful for users who want to quickly manipulate a few files before returning to Odyssey. The keystroke **ALT+O** can be used as a shortcut for this menu item.

☞ Implementation note: Several beta testers commented about the presence of this feature - is it needed? Why wouldn't the user just switch to File Manager or somesuch, if he wanted to manipulate files?

All we can say in reply is that your humble author has found it quite useful for quickly nipping into DOS to create or zap a directory. Compare "ALT+O,md mydir,exit" with the standard equivalent of ALT-Tabbing to switch to Program Manager, open the "main" program group, run File Manager, change the view to the correct partition, File|Create directory..., close file manager, ALT-Tab back to Odyssey... However, If you are really desparate to do it the long winded way then the presence of the above menu item certainly shouldn't hinder you!

Odyssey Menus

File|eXit from Odyssey...

Selecting this menu item tells Odyssey to shut itself down. The **ALT+X** keystroke can be used as a shortcut for this menu item, as can clicking on the "Exit Odyssey" toolbar button. However, Odyssey asks for confirmation if you use a shortcut method.

If there are any "modified" text editor windows opened then Odyssey will ask if you wish to save those files.

Odyssey Menus

Edit menu

It is conventional for all Windows applications to have an **Edit** menu, and for that menu to be the second item on the menu bar, immediately after the File menu. The Edit menu normally deals with clipboard operations, plus specialised application functions which deal with examining or editing the current document.

Edit|Review buffer
Edit|Cut
Edit|Copy
Edit|Paste

Odyssey Menus

Edit|Review buffer

Selecting the **Edit|Review buffer** menu item causes Odyssey to open its special "Review Buffer" text editor window. The Review Buffer is used to display the most recent 32000 characters which was displayed in the terminal window. You may use the **ALT+R** keystroke as a shortcut for this menu item.

Odyssey Menus

Edit|Cut

This menu item is expected to appear in the Edit menu of a Windows application, however a "Cut to clipboard" function is not application to Odysseys terminal or directory view windows, and this menu item is therefore disabled.

Odyssey Menus

Edit|Copy

This menu item is expected to appear in the Edit menu of a Windows application, however a "Copy to clipboard" function is not application to Odysseys terminal or directory view windows, and this menu item is therefore disabled.

If you wish to copy text from an online session to the Windows clipboard then you should do so by opening the Review editor (see Edit|Review buffer), and then copying the text from there.

Odyssey Menus

Edit|Paste

This menu item is expected to appear in the Edit menu of a Windows application, however a "Paste from clipboard" function is not application to Odysseys terminal or directory view windows, and this menu item is therefore disabled.

Odyssey Menus

Setup menu

The **Setup** menu is one of the most important features in Odyssey. It is via this menu that you reach all the important configuration dialogs, which is how you make Odyssey work the way you want it to.

Setup|Communications...
Setup|Modem...
Setup|General...
Setup|Editor...
Setup|File transfer...
Setup|Terminal emulation...
Setup|Fax...
Setup|Host mode...
Setup|Keyboard macros...
Setup|Printer...
Setup|Save settings

Odyssey Menus

Setup|Communications...

The **Setup** menu is one of the most important features in Odyssey. It is via this menu that you reach all the important configuration dialogs, which is how you make Odyssey work the way you want it to. The **Setup|Communications** menu item leads to the Setup|Communications dialog which controls the serial port settings which Odyssey uses - baud rate, parity etc.

Odyssey Menus

Setup|Modem...

The **Setup** menu is one of the most important features in Odyssey. It is via this menu that you reach all the important configuration dialogs, which is how you make Odyssey work the way you want it to. The **Setup|Modem...** menu item leads to the Setup|Modem dialog, which can be used to change the command strings which Odyssey sends to the modem for initialization and dialing.

Odyssey Menus

Setup|General...

The **Setup** menu is one of the most important features in Odyssey. It is via this menu that you reach all the important configuration dialogs, which is how you make Odyssey work the way you want it to. The **Setup|General...** menu item leads to the Setup|General dialog, which affects miscellaneous features in Odyssey, such as the storage directory for downloaded files, and so on.

Odyssey Menus

Setup|Editor...

The **Setup** menu is one of the most important features in Odyssey. It is via this menu that you reach all the important configuration dialogs, which is how you make Odyssey work the way you want it to. The **Setup|Editor...** menu item leads to the Setup|Editor dialog, which is used to control the default settings for an Odyssey text editor window (for example, the default settings control whether a text editor window will have the "word wrap" feature enabled when it opens).

See also: The Odyssey Text Editor

Odyssey Menus

Setup|File transfer...

The **Setup** menu is one of the most important features in Odyssey. It is via this menu that you reach all the important configuration dialogs, which is how you make Odyssey work the way you want it to. The **Setup|File transfer...** menu item leads to the Setup|File transfer dialog, which can be used to control optional features of the ASCII, Zmodem, and Compuserve B+ protocols.

See also: File Transfer in Odyssey

Odyssey Menus

Setup|Terminal Emulation...

The **Setup** menu is one of the most important features in Odyssey. It is via this menu that you reach all the important configuration dialogs, which is how you make Odyssey work the way you want it to. The **Setup|Terminal emulation...** menu item leads to the Setup|Terminal emulation dialog, which allows you to control terminal features such as the terminal type to emulate, local echo and so on.

See also: Terminal Emulation in Odyssey

Odyssey Menus

Setup|Fax...

The **Setup** menu is one of the most important features in Odyssey. It is via this menu that you reach all the important configuration dialogs, which is how you make Odyssey work the way you want it to. The **Setup|Fax...** menu item leads to the Setup|Fax dialog, which stores variables involved in FAX transfer, such as the directory for FAX files, your local FAX station ID etc.

See also: The Odyssey FAX Server

Odyssey Menus

Setup|Host mode...

The **Setup** menu is one of the most important features in Odyssey. It is via this menu that you reach all the important configuration dialogs, which is how you make Odyssey work the way you want it to. The **Setup|Host mode...** menu item leads to the Setup|Host mode dialog, which can be used to edit control variables used by host mode, such as the default directory for callers, the normal and privileged user passwords, etc.

See also: Odyssey Host Mode

Odyssey Menus

Setup|Keyboard macros...

The **Setup** menu is one of the most important features in Odyssey. It is via this menu that you reach all the important configuration dialogs, which is how you make Odyssey work the way you want it to. The **Setup|Keyboard macros** menu item leads to the Setup|Macros dialog, which allows you to assign commonly used host commands to an **ALT+<*digit*>** keystroke.

Odyssey Menus

Setup|Printer...

The **Setup** menu is one of the most important features in Odyssey. It is via this menu that you reach all the important configuration dialogs, which is how you make Odyssey work the way you want it to. The **Setup|Printer...** menu item leads to the **Setup|Printer** dialog, which allows you to control Odyssey use of the printer, by selecting the printer to use. This dialog also gives access to the common "Print setup" dialog provided by the Windows printer driver.

Odyssey Menus

Setup|Save setup

When any of the setup options have been changed then you can save your changes to disk using this menu item. The configuration data is saved to a private binary file called "ODYSSEY.CFG" which is read automatically, the next time you run Odyssey.

Odyssey saves other configuration information between sessions, such as the size and position of the terminal window.

Odyssey Menus

Upload/Download menus

The **Upload** and **Download** menus are used to initiate a file transfer in Odyssey. When you want to upload a file, you select a protocol from the Upload menu. Likewise when you want to download a file, you choose the protocol from the download menu.

Upload/Download|ASCII
Upload/Download|Xmodem...
Upload/Download|Ymodem (Xmodem 1K)...
Upload/Download|Batch Ymodem...
Upload/Download|Kermit...
Upload/Download|Zmodem...
Upload/Download|Ymodem-G...
Upload/Download|Compuserve B+...

☞ The Zmodem and Compuserve B+ protocols support an *auto-download* feature whereby the appropriate Odyssey receive protocol is triggered automatically by the host. If this feature is enabled then you should *not* try to start the download manually from the Download menu, as this will only cause confusion. Compuserve B+ can also trigger an upload automatically. These protocols appear in the above menus only because the features mentioned can be disabled in Setup|File transfer, and in that case you *would* need to start the transfer manually.

Odyssey Menus

Upload/Download|ASCII

When **Upload|ASCII** is selected, Odyssey displays a standard File Selector dialog, which prompts the user to supply the name of the ASCII file which is to be uploaded - a wildcard name is not accepted. Note that since ASCII upload is a not a protocol in the true sense of the word (and hence has no provision for error checking), Odyssey uses character and line delays to avoid sending characters too fast for the host to process. The length of these delays can be configured by you using the Setup|File transfer dialog.

The **Download|ASCII** menu item is provided solely to preserve symmetry between upload and download menus. Selecting this menu item is just another way of opening a text logging (text capture) mode.

See also: File Transfer in Odyssey

Odyssey Menus

Upload/Download|Xmodem...

When **Upload|Xmodem...** or **Download|Xmodem...** is selected, Odyssey displays a standard File Selector dialog, which prompts the user to supply the name of the text or binary file which is to be uploaded or downloaded - Xmodem is not a batch protocol, so a wildcard name is not accepted.

Note that you should respond to the file selector dialog as quickly as possible, since the remote end is waiting to start the file transfer, and may time out and abort if you take too long.

Xmodem will not work if the serial connection has parity checking enabled.

See also: File Transfer in Odyssey

Odyssey Menus

Upload/Download|Ymodem...

When **Upload|Ymodem...** or **Download|Ymodem...** is selected, Odyssey displays a standard File Selector dialog, which prompts the user to supply the name of the text or binary file which is to be uploaded or downloaded - Ymodem is not a batch protocol, so a wildcard name is not accepted.

Note that you should respond to the file selector dialog as quickly as possible, since the remote end is waiting to start the file transfer, and may time out and abort if you take too long.

☞ Ymodem will not work if the serial connection has parity checking enabled.

The Ymodem protocol is sometimes called **Xmodem-1K**

See also: File Transfer in Odyssey

Odyssey Menus

Upload/Download|Batch Ymodem...

When **Upload|Batch Ymodem...** is selected, Odyssey displays a standard File Selector dialog, which prompts the user to supply the name of the text or binary file which is to be uploaded - as the name implies, Ymodem Batch is a batch protocol, so a wildcard name will be accepted (click on "*OK all matching*" in the file selector dialog). Note that you should respond to the file selector dialog as quickly as possible, since the remote end is waiting to start the file transfer, and may time out and abort if you take too long.

When **Download|Batch Ymodem** is selected Odyssey does not prompt for a file name, since that information will be supplied by the remote end of the protocol.

Ymodem batch will not work if the serial connection has parity checking enabled.

The Ymodem Batch protocol is sometimes called **True Ymodem**

See also: File Transfer in Odyssey

Odyssey Menus

Upload/Download|Kermit...

When **Upload|Kermit...** is selected, Odyssey displays a standard File Selector dialog, which prompts the user to supply the name of the text or binary file which is to be uploaded. Kermit is a batch protocol, so a wildcard name will be accepted (click on "*OK all matching*" in the file selector dialog). Note that you should respond to the file selector dialog as quickly as possible, since the remote end is waiting to start the Kermit file transfer, and may time out and abort if you take too long.

When **Download|Kermit** is selected Odyssey does not prompt for a file name, since that information will be supplied by the remote end of the protocol.

☞ Kermit is the only file transfer protocol supported by Odyssey which will work normally when the serial connection has parity checking enabled. However, you should still avoid using parity if you possibly can, since parity checking will make the file transfer take at least 20% longer to complete.

See also: File Transfer in Odyssey

Odyssey Menus

Upload/Download|Zmodem...

When **Upload|Zmodem...** is selected, Odyssey displays a standard File Selector dialog, which prompts the user to supply the name of the text or binary file which is to be uploaded. Zmodem is a batch protocol, so a wildcard name will be accepted (click on "*OK all matching*" in the file selector dialog). Note that you should respond to the file selector dialog as quickly as possible, since the remote end is waiting to start the Zmodem transfer, and may time out and abort if you take too long.

When **Download|Zmodem** is selected Odyssey does not prompt for a file name, since that information will be supplied by the remote end of the protocol. If you have "*Auto-Download*" enabled in the Zmodem panel of the Setup|File Transfer dialog then you do not need to select this menu option at all - the host will automatically trigger Odyssey into Zmodem receive mode.

☞ Zmodem will not work if the serial connection has parity checking enabled.

See also: File Transfer in Odyssey

Odyssey Menus

Upload/Download|Ymodem-G...

When **Upload|Ymodem-G...** is selected, Odyssey displays a standard File Selector dialog, which prompts the user to supply the name of the text or binary file which is to be uploaded. Ymodem-g is a batch protocol, so a wildcard name will be accepted (click on "*OK all matching*" in the file selector dialog). Note that you should respond to the file selector dialog as quickly as possible, since the remote end is waiting to start the file transfer, and may time out and abort if you take too long. In fact, Ymodem-G is identical to Ymodem Batch when uploading, since only the receiving side can request that the "G" variant of Ymodem be used.

When **Download|Ymodem-G** is selected Odyssey does not prompt for a file name, since that information will be supplied by the remote end of the protocol.

Ymodem-G will not work if the serial connection has parity checking enabled.

See also: File Transfer in Odyssey

Odyssey Menus

Upload/Download|Compuserve B+...

When **Upload|Compuserve B+...** is selected, Odyssey displays a standard File Selector dialog, which prompts the user to supply the name of the text or binary file which is to be uploaded. Compuserve B+ is a batch protocol, so a wildcard name will be accepted (click on "*OK all matching*" in the file selector dialog). Note that you should respond to the file selector dialog as quickly as possible, since the remote end is waiting to start the Compuserve B+ transfer, and may time out and abort if you take too long.

When **Download|Compuserve B+** is selected Odyssey does not prompt for a file name, since that information will be supplied by the remote end of the protocol.

Compuserve B+ will not work if the serial connection has parity checking enabled. Also, note that Compuserve B+ supports "auto-start" for both downloads and uploads. You would normally enable the relevant options in the Setup|File transfer dialog after logging on to Compuserve, and so you should never actually have to use these menu commands. In fact, CIS B+ timing is rather awkward, and you may find it quite hard to get the transfer started manually.

See also: File Transfer in Odyssey

Odyssey Menus

Terminal Window Command menu

Each of the Odyssey MDI document windows offers a **Command** menu, which (as the name implies) activates command functions particular to the type of window currently active. In the case of the Terminal Window, these commands mostly relate to controlling an online session by issuing commands to the modem, starting and compiling scripts and so on.

Command|Send break
Command|Hang up
Command|Dial...
Command|Dial again...
Command|Clear terminal screen
Command|Play back log file...
Command|Load keyboard template...
Command|Re-initialize modem
Command|Learn script...
Command|Compile script...
Command|1/2/3...9
Command|More scripts...

Odyssey Menus

Command|Send break

This menu option causes Odyssey to assert a break signal on the modem interface for roughly half a second. Some remote hosts require this signal to tell them that a particular terminal requires attention, for example because you wish to log on. A short-cut for this command is provided, use **ALT+B** when the **terminal window** is active.

Odyssey Menus

Command|Hang up

This menu command causes Odyssey to instruct the modem to hang up the phone. It is more normal for the remote system to terminate the call itself, after you type the "bye" command or some equivalent. A short-cut for this command is provided; use **ALT+H** whenever the terminal window is active.

Odyssey Menus

Command|Dial...

This command invokes the Odyssey dialing directory feature, your main starting point when dialing any number. The dialing directory feature is described in detail elsewhere. A short-cut for this command is provided, press **ALT+N** when the terminal window is active.

Odyssey Menus

Command|Dial again...

The dialing directory can be used to create a *dial queue* with one or more entries. When Odyssey fails to make a connection using the numbers in this queue the queue itself is left intact. Likewise if Odyssey succeeds in making a connection a single entry is removed from the queue, which otherwise remains intact. To continue dialing the remaining numbers in that queue, ie. without generating a new queue, you choose this menu option.

As an example of when this command might be used: suppose you call the same five bulletin boards every day. Each of those BBS has a dialing directory entry, which are left tagged at all times. When you first run Odyssey you select "*Dial Tagged*", which causes an initial dial queue to be created, and Odyssey connects to the first number. When that call is completed you select this menu option and Odyssey connects to the second number, and so on until you have visited each BBS.

Odyssey Menus

Command|Clear terminal screen

This command causes the terminal screen to be cleared. Review buffers etc. are unaffected. A short-cut alternative is provided for this menu option - type **ALT+T** when the terminal window is active.

Odyssey Menus

Command|Play back log file...

This facility allows you to play back a log file of data received from a remote system. The log is played back at the speed currently set up as your baud rate in Setup|Comms. Playback mode can be used as an offline reader or as a means of testing terminal emulations.

It is important that log files played back are recorded using Odyssey *Raw Logging* mode, otherwise Odyssey may have filtered out some control characters, and the playback will be inaccurate.

Playback has two modes - "single step" and "normal". In single step mode the playback stops every time Odyssey sees a control character, whereas in normal mode the playback is continuous. Playback mode is initially single-step, but can be switched to continuous by pressing space at the first pause.

See also: Text Logging

Odyssey Menus

Command|Load keyboard template...

A keyboard definition file allows you to redefine the meaning of any of the keys on your keyboard. You must prepare the file using the Keyboard Remapping facility, and the resulting .KEY file can then be loaded using this menu option or from a script.

Additionally, a keyboard definition file is loaded when you change terminal emulations, provided that the keyboard file is given a name matching the terminal emulation, eg. VT100.KEY. When a new keyboard definition is loaded any previous definitions are discarded, ie., Odyssey does not accumulate definitions from multiple keyboard files.

Odyssey Menus

Command|Re-initialize modem

This command causes Odyssey to transmit the configured init string to the modem. This contents of this init string is set using the "Init String" option of the Setup|Modem dialog, and is normally transmitted when Odyssey starts up. There is no need to retransmit the init string in a session unless you specifically want to restore the modem to its startup settings.

A shortcut for this command is provided - press **ALT+J** when the terminal window is active.

Odyssey Menus

Command|Learn script...

This menu item allows the non-programmer to create the scripts which will automate the process of logging on to a remote host. With a modem correctly connected to the computer and phone line, select **Learn script...** from the Command menu, then enter a file name for the script as prompted. From this point manually call up the remote system, using the normal procedure for that system. While you do so Odyssey is keeping a complete record of what the remote system was sending, and how you responded to it. Once the remote system that was called is online and all passwords etc. have been entered, you press the <Esc> key to end Learn mode. The recording of the call will be converted into a script and saved under the name you gave. Next time you call the same system you should look under the Command menu for the script which was created. From then on the process of logging on to that particular host will be automatic.

A more detailed discussion of using Learn mode is given in the Using Scripts section of this user guide.

Odyssey Menus

Command|Compile...

You are likely to use this menu command mostly to compile script sources, but it is also used when compiling keyboard templates. This discussion will concentrate on how you compiling scripts - for a discussion of compiling keyboard templates, see the description of the Keyboard Remapping feature.

Compiling Scripts.

It is not always obvious, but Odyssey always compiles scripts internally into a private pseudo-machine code before running them. This has many advantages including the ability to spot syntax errors before you go online and also because the script executes faster, with less overhead.

☞ This is feature best appreciated by Odyssey users who are also programmers. If you do not understand what this utility does, then you can safely assume that you do not need it.

If you have been writing large scripts then you will probably have noticed the long pause after you load a script before it begins executing - this is when Odyssey is compiling the script. You can get rid of these delays by saving them in a pre-compiled format, as follows:

Rename the script source file *name*.SCR to *name*.SRC. Then select this menu command, and enter the name of the source script (name.SRC) in reply to the file selector dialog which is displayed.

Odyssey will compile the named script, writing a new file called *name*.SCR (**the point to note is that scripts to be executed by Odyssey must have the .SCR extension whether they are precompiled or not, otherwise they will not be found by the Command menu or dialing directory**). The precompiled script may thereafter be chosen from the Command menu like any other script, but will load faster because it does not need to be compiled again.

You can also compile scripts by loading the .SRC file into an Odyssey text editor, and then pressing **F9**.

See also: Using Odyssey Scripts

Odyssey Menus

Command|1/2/3...9

When you pull down the **Command** menu, Odyssey searches for any scripts, and displays the first nine it finds in the Command menu, and numbers them 1 to 9. You can run these scripts by pulling down the Command menu and pressing the relevant digit.

If Odyssey finds more than nine scripts then it displays only the first nine in the menu, and it also displays a "More scripts" menu item, leading to a dialog from which you may select any unlisted script.

Odyssey Menus

Command|More scripts...

When you pull down the **Command** menu, Odyssey searches for any scripts, and displays the first nine it finds in the Command menu, and numbers them 1 to 9. You can run these scripts by pulling down the Command menu and pressing the relevant digit.

If Odyssey finds *more* than nine scripts then it displays only the first nine in the menu, and it also adds the "More scripts" item to the menu. Selecting this menu item leads to a dialog from which you may select any of the unlisted scripts.

Odyssey Menus

Window menu

The **Window** menu is a standard fixture of Multiple Document Interface (**MDI**) Windows applications. It exists primarily to allow you to move between open document windows.

Window|Chat mode
Window|Answer (Host) mode
Window|FAX server...
Window|Tile
Window|Cascade
Window|Arrange icons
Window|1/2/3..9
Window|More windows...

Odyssey Menus

Window|Chat mode

This function is disabled unless the active window is the Terminal Window. The **Window|Chat mode** menu item toggles *Chat mode*, ie. if Chat mode is currently in force then this function will terminate that mode and return to a standard terminal emulation. If Chat mode is inactive then this function enables it.

Chat mode allows you to have a keyboard "chat" with another modem user in real time. The terminal window is split into two panels - the *remote* and *local* panels. The *Remote* panel shows characters received from the remote system, and the *Local* window shows characters typed on the local keyboard.

Press <Esc> to leave Chat mode.

See also: Chat Mode

Odyssey Menus

Window|Answer (Host) mode

This function is only available when the Terminal Window is the active document window. The **Window|Answer (Host) mode** menu item causes Odyssey to enter "Host Mode", a mode which allows remote users to call your system, provided that your modem is configured to auto-answer. Press the <Esc> key to end Host mode.

See also:

- Setup|Host mode dialog
- Odyssey Host Mode

Odyssey Menus

Window|FAX server...

This function is only available when the Terminal Window is the active document window. The **Window|Fax server...** menu item causes Odyssey to enter "FAX Server Mode". A shortcut for this command is provided - press **ALT+V** when the terminal window is active.

See also: Odyssey Fax Server

Odyssey Menus

Window|Tile

The **Window|Tile** menu item is a standard feature of Windows Multiple Document Interface (MDI) applications. This command instructs Odyssey to move and resize its child windows so that all are visible, and none overlap.

Odyssey Menus

Window|Cascade

The **Window|Cascade** menu item is a standard feature of Windows Multiple Document Interface (MDI) applications. This command instructs Odyssey to move and resize its child windows so that all are visible, and overlap in a consistant fashion such that the caption bar of each window is visible.

Odyssey Menus

Window|Arrange icons

The **Window|Arrange Icons** menu item is a standard feature of Windows Multiple Document Interface (MDI) applications. This command instructs Odyssey to rearrange minimized child windows so that the icons are evenly spaces near the bottom of the Odyssey desktop area. This menu item will be grayed if there are no currently iconic children to rearrange.

Odyssey Menus

Window|1/2/3..9

The **F6** key can be used to switch to the "next" Odyssey child window. However, if you want to move to a specific child window then the only way to do so is to pull down the **Window** menu and select the child from the list of menu items numbered 1,2,3...9. If more than nine document windows are opened then a "*More windows*" item will be added to the menu. Selecting the latter displays a dialog which allows you to select any child window, including those not listed on the menu.

Naturally, you can ensure that a document window is always listed on the menu by ensuring that you never have more than nine document windows open at any time.

Odyssey Menus

Window|More windows...

The **F6** key can be used to switch to the "next" Odyssey child window. However, if you want to move to a specific child window then the only way to do so is to pull down the **Window** menu and select the child from the list of menu items numbered 1,2,3...9. If more than nine document windows are opened then the **More windows...** item is added to the menu. Selecting this menu item displays a dialog which allows you to select any child window, including those not listed on the menu.

You may find this a bit long winded, in which case you can ensure that a document window is always listed on the menu by ensuring that you never have more than nine document windows open at any time.

Odyssey Menus

Help menu

The **Help** menu gives you access to the major informational topics in the Odyssey help system / user guide. You can also access help by clicking the **?** toolbar button, or by pressing **F1** anywhere in Odyssey for context sensitive information. By convention, the Help menu is always the last item on the menu bar, and also provides access to "About" box. The items on the Help menu are as follows:-

Help|Contents...
Help|Odyssey configuration...
Help|Using Odyssey...
Help|Odyssey Features...
Help|Odyssey Menus...
Help|Script Tutorial...
Help|Script commands...
Help|Help on help...
Help|About Odyssey...

Odyssey Menus

Help|About Odyssey...

The **Help** menu gives you access to the major informational topics in the Odyssey help system / user guide. You can also access help by clicking the **?** toolbar button, or by pressing **F1** anywhere in Odyssey for context sensitive information.

The **Help|About Odyssey...** menu item causes Odyssey to display its "About Box" dialog, which contains the Odyssey logo, a copyright notice, and contact details for Skyro Software Ltd.

Odyssey Menus

Text Editor Menus

These are the menus available when the text editor window is active.

File menu
Edit menu
Setup menu
Options menu
Block menu
Command menu
Window menu
Help menu

Odyssey Menus

Text Editor File menu

It is conventional for all Windows applications to have a **File** menu, and for that menu to be the first item on the main menu bar. The File menu is used when loading and saving documents, printing, and other operations which concern access to files or directories.

File|Open...
File|Close all
File|Save
File|Save as...
File|Print document
File|Text logging...
File|Printer logging
File|View directory...
File|DOS Shell
File|eXit from Odyssey...

Odyssey Menus (Text Editor)

File|Open...

The **File|Open...** menu item is used to load a text file into an Odyssey text editor. The user is presented with a standard File Selector dialog from which to make his choice of files, and a new text editor window is then opened to display and edit that file. The filename entered into the dialog need not already exist - if the file does not exist then a new file of that name is created.

The **F3** key may be used as a shortcut for this command.

Odyssey Menus (Text Editor)

File|Close all

The **File|Close all** menu option closes all currently open document windows (directory viewers and bitmap viewers, text edit windows etc). This is quicker than closing each window individually.

Odyssey Menus (Text Editor)

File|Save

This command instructs Odyssey to save the text document in the currently active text editor window. The file is saved using its existing name (the name that appears in the window caption). If the "*Keep backups*" option is enabled in Setup|Editor then any existing file will be renamed as *filename*.BAK before the new file is written.

If you wish to save the file to a different name then choose the File|Save as... option instead.

The **F2** key may be used as a shortcut for this command.

Odyssey Menus (Text Editor)

File|Save as...

This command instructs Odyssey to save the text document in the currently active text editor window. A standard file selector dialog appears, which prompts the user to supply new name for the document. If a file already exists with the new name then Odyssey will display a dialog asking for confirmation that you wish to overwrite the existing file.

If you wish to save the file to its current name (the name that appears in the window caption), then choose the File|Save option instead.

Odyssey Menus (Text Editor)

File|Print document

This command instructs Odyssey to print the text document in the currently active text editor window. In this case, the entire document is printed - if you wish to print just a portion of the document then you can do so by marking that section as a block, and then using the "print block" command (**Ctrl+K P**).

Odyssey Menus (Text Editor)

File|Text logging...

This command is not applicable to the text editor, and is this disabled.

Odyssey Menus (Text Editor)

File|Printer logging

This command is not applicable to the text editor, and is this disabled.

Odyssey Menus (Text Editor)

File|View directory...

The **File|View directory...** menu item causes Odyssey to open a directory viewer window, which allows the user to copy, rename and delete files etc. See elsewhere for a detailed description of the Odyssey Directory Viewer module.

Odyssey Menus (Text Editor)

File|DOS Shell

When the **File|DOS Shell** menu item is used, Odyssey asks Windows to run a copy of the DOS command interpreter. This can be useful for users who want to quickly manipulate a few files before returning to Odyssey.

Odyssey Menus

File|eXit from Odyssey...

Selecting this menu item tells Odyssey to shut itself down. The **ALT+X** keystroke can be used as a shortcut for this menu item, as can clicking on the "Exit Odyssey" toolbar button. However, Odyssey asks for confirmation if you use a shortcut method.

If there are any "modified" text editor windows opened then Odyssey will ask if you wish to save those files.

Odyssey Menus

Text Editor Edit menu

It is conventional for all Windows applications to have an **Edit** menu, and for that menu to be the second item on the menu bar, immediately after the File menu. The Edit menu deals with clipboard operations, plus specialised application functions which deal with examining or editing the current document.

Edit|Review buffer
Edit|Cut
Edit|Copy
Edit|Paste

Odyssey Menus (Text Editor)

Edit|Review buffer

Selecting the **Edit|Review buffer** menu item causes Odyssey to open its special "Review Buffer" text editor window. The Review Buffer is used to display the most recent 32000 characters which was displayed in the terminal window. You may use the **ALT+R** keystroke as a shortcut for this menu item.

Odyssey Menus (Text Editor)

Edit|Cut

This menu option causes the Odyssey text editor to "cut" the currently marked and visible block to the Windows clipboard. In other words, a copy of the current block is placed on the clipboard, and the block is then deleted from the current document.

Odyssey Menus (Text Editor)

Edit|Copy

This menu option causes the Odyssey text editor to place a copy of the currently marked and visible block on the clipboard, ready to be pasted into this or another text editor window, or into a text control in another application.

☞ The Odyssey text editor is a "large file" text editor, ie. there is no limit on the size of file which can be edited - nor is there a limit on the size of the block you can place on the clipboard. However, you should be warned that *other* applications may have difficulty pasting very large blocks (eg. a standard windows multiline edit control has a 32k size limit, which will of course have to include the text already in the control). If you want to cut and paste between the Odyssey text editor and another application then you would be advised to do it incrementally, in reasonable sized chunks.

Odyssey Menus (Text Editor)

Edit|Paste

If any text object is available on the Windows clipboard, then this command causes the Odyssey text editor to insert that text into the currently active editor window, at the current cursor position. The pasted text then becomes the currently marked and visible block, to which all the normal editor block commands may apply.

Odyssey Menus

Text Editor Options menu

The text editor **Options** menu allows the user to control various mode settings in the currently active editor. Of of these menu options have keyboard equivalents prefixed with the **Ctrl+O** sequence.

Options|Insert mode
Options|Auto-indent
Options|Word wrap
Options|Hard tabs
Options|Smart tabs
Options|Create backup files
Options|Right justify lines
Options|End lines with LF

Odyssey Menus (Text Editor)

Options|Insert mode

This command is used to toggle the editor between insert and overtype modes. The legend "Insert" will appear on the editor status line when the editor is currently in insert mode, or else this word will be "Overtype" when you are in overtype mode.

Characters typed while the editor is in insert mode are inserted into the document at the current cursor position, with any characters on the line at and to the right of this position being shifted further right to make room. The cursor then advances to the right, ending up on the same character it was on before the insertion.

In Overtype mode the character typed replaces the character previously at that cursor position; the character overtyped is lost. The cursor then advances to the next column.

The **Insert** key be also be used to toggle Insert/Overtype modes.

Odyssey Menus (Text Editor)

Options|Auto-indent

Auto-indent refers to the editor feature whereby, when you type <Enter> to begin a new line, that new line is automatically given the same indentation as the line just completed, and thus the cursor is placed immediately below the first character of the previous line. When Auto-Indent is disabled the new line is not given any default indent at all, and the cursor therefore moves to column one in the new line.

The equivalent keyboard shortcut is **Ctrl+O I**.

Odyssey Menus (Text Editor)

Options|Word wrap

This command toggles "Word Wrap" mode, ie. if the word wrap was initially disabled then this command will enable it, and vice versa. When word wrap mode is enabled a "Wordwrap" symbol will be visible on the editor status line.

Word wrap occurs when the cursor reaches the right margin as you type. If that happens then the word currently being typed is moved to the next line, with the cursor positioned after the last character in the word. If the "Justification" option is also enabled then the editor will format the line just completed so that it exactly fits the line between left and right margins.

You may use **Ctrl+O W** as a keyboard shortcut for this command.

Odyssey Menus (Text Editor)

Options|Hard tabs

The Odyssey text editor can use **Hard Tabs** or **Soft Tabs** (note that this option is independant of the Smart or Fixed tabs option described next). In Hard Tab mode tab intervals are filled with actual tab characters (ASCII 9), whereas in soft tab mode the tab interval is filled with spaces. Hard tabs make the text file smaller, but may create formatting problems if you import the text into an editor that assumes a different tab interval. The Hard/Soft tab option *only* affects what Odyssey does when you insert a tab in the Odyssey editor - selecting soft tabs does not prevent Odyssey interpreting tab characters correctly when it reads in a foreign ASCII file. Note: the Setup|Editor dialog refers to this feature as the "*Tab Fill Character*". If the fill character is Tab then you are using hard tabs, if it is space then you are using soft tabs.

You may use **Ctrl+O H** as the keyboard shortcut for this command.

Odyssey Menus (Text Editor)

Options|Smart tabs

The editor can use **Smart Tabs** or **Fixed Tabs**. Smart tabbing means that when you press the tab key, the editor examines the line above the current line, and aligns the cursor with the next word on that previous line. This is extremely useful when laying out tables. Fixed tabs are your usual fixed-interval tabs - the default interval being eight columns, though this can be changed in the Setup|Editor dialog.

You may use the sequence **Ctrl+O F** as a shortcut for this command.

Odyssey Menus (Text Editor)

Options|Create backup files

This function controls whether or not the Odyssey text editor writes a .BAK file containing the old file contents, every time you save the file. If you disable this function then no backups are kept.

You may use **Ctrl+O B** as the keyboard shortcut for this command.

Odyssey Menus (Text Editor)

Options|Right justify lines

This command toggles "Right Justify" mode. If enabled, the editor ensures that every line exactly meets the right margin whenever a line or paragraph is reformatted. If disabled line ends are allowed to remain ragged.

You may use the sequence **Ctrl+O J** as the keyboard shortcut for this command.

Odyssey Menus (Text Editor)

Options|End lines with LF

The Odyssey text editor can read files whose lines end in **CRLF** (the normal DOS convention), or which end in **LF** only (the convention for text files originating on Unix systems). Odyssey does not mind if the same file contains a mixture of different line end types, and you don't need to set any modes in order to be able to read these files.

However, when you insert new lines, the Odyssey text editor must know whether you want the new line to end in LF or CRLF, which is why this option exists. If this option is enabled, then new lines will end in LF, otherwise they will end with CRLF. The editor **Options** menu shows the current state of the EOL toggle.

You may use the sequence **Ctrl+O L** as the keyboard shortcut for this command.

Odyssey Menus

Text Editor Block menu

Block commands allow you to mark out a segment of text in order to apply an operation to that entire segment (block). For example after marking a block you can then choose to delete, copy, move or print that block. You can also cut or copy the block to the clipboard (or write the block to disk) in order to paste it into another editor, or another application, and you can read a file from disk and merge it with the current file, in which case the merged text becomes a new marked block in the current document.

Block|Mark beginning
Block|Mark end
Block|Mark word
Block|Mark line
Block|Hide/Display
Block|Read from file...
Block|Write to file...
Block|Append to file...
Block|Print
Block|Copy
Block|Move
Block|Delete
Block|Indent
Block|Unindent
Block|Paste to serial port

Odyssey Menus (Text Editor)

Block|Mark beginning

This command is used to tell Odyssey that you want a new block to begin at the current cursor position in the active text editor window. No physical marker appears in the text, however the entire block will be highlighted once you have marked both beginning and end (in either order). Once a block is marked and highlighted it may be the subject of other block commands such as move, copy or delete block.

You may use **F7** or **Ctrl+K B** as keyboard shortcuts for this command.

This is the WordStar™ method of marking a block. The Odyssey text editor naturally also supports the Windows (CUA) method of block marking using the cursor or **Shift+<*movement key*>**.

Odyssey Menus (Text Editor)

Block|Mark end

This command is used to tell Odyssey that you want a block to end at the current cursor position in the active text editor. No physical marker appears in the text, however the entire block will be highlighted once you have marked both beginning and end (in either order).

You may use **F8** or **Ctrl+K K** as keyboard shortcuts for this command.

☞ This is the WordStar™ method of marking a block. The Odyssey text editor naturally also supports the Windows (CUA) method of block marking using the cursor or **Shift+<*movement key*>**.

Odyssey Menus (Text Editor)

Block|Mark word

This command may be used if you wish to mark a single word as a block so that it may be copied or moved - the word marked is the one at the cursor position in the currently active editor. Alternatively, you can mark a word by double-clicking it with the mouse, or by typing **Shift+Ctrl+→** (although the latter command is slightly different - it marks from the cursor position to the end of the word).

You may use **Ctrl+K T** as a shortcut for this command.

Odyssey Menus (Text Editor)

Block|Mark line

This command may be used if you wish to mark the current line as a block so that it can be copied or moved. The line marked is the one under the cursor in the active text editor.

You may use **Ctrl+K L** as a shortcut for this command.

Odyssey Menus (Text Editor)

Block|Hide/Display

When a block is marked and highlighted, you may un-highlight it using this command. However, the editor does not forget where the block markers were, so selecting this option again will cause the block to become visible once more. The hide block command is most commonly used after a move or copy block operation, as this leaves a highlighted block at the destination position. If you intend no further operations on that block then this command will un-highlight it.

You may use **Ctrl+K H** as a shortcut for this command. You may also hide a block by clicking with the mouse anywhere inside the text edit window.

Odyssey Menus (Text Editor)

Block|Read from file...

This command is used to read text from a file on disk and insert that text at the cursor position in the currently active text editor, leaving it as the currently marked and highlighted block. The editor will prompt for the name of the file containing the text you want to read.

You may use **Ctrl+K R** as a keyboard shortcut for this command.

Odyssey Menus (Text Editor)

Block|Write to file...

Writes the currently marked and highlighted block to a file on disk. The editor will prompt for a name to give to the new file. If a file exists already with the same name then you will be asked whether the existing file should be erased.

You may use **Ctrl+K W** as a keyboard shortcut for this command.

Odyssey Menus (Text Editor)

Block|Append to file...

Appends the currently marked and highlighted block to an existing file on disk. The editor will prompt for the name of the file to which the block is to be appended.

You may use **Ctrl+K A** as a keyboard shortcut for this command.

Odyssey Menus (Text Editor)

Block|Print

This command may be used to copy the currently marked and highlighted block to the printer, which must be online and ready to receive data.

You may use **Ctrl+K P** as a keyboard shortcut for this command.

If no block is highlighted when this command is entered then the entire document is printed.

Odyssey Menus (Text Editor)

Block|Copy

Copies the currently marked and highlighted block to the cursor position of the currently active text editor. The new block then becomes the currently marked block. Attempts to copy a block onto itself are ignored.

The sequence **Ctrl+K C** may be used as the keyboard shortcut for this command.

Odyssey Menus (Text Editor)

Block|Move

Moves the currently marked and highlighted block to the cursor position in the currently active editor. The block remains highlighted at its new position. Attempts to move a marked block into the highlighted area (ie. onto itself) are ignored. Moving a block is equivalent to copying it to the destination position, and then deleting the original marked block.

You may use **Ctrl+K V** as a the keyboard shortcut for this command.

Odyssey Menus (Text Editor)

Block|Delete

Deletes the marked and highlighted block from the active text editor, moving remaining text in the document up to close the gap. A block once deleted is lost.

You may use **Ctrl+K Y** as a the keyboard shortcut for this command. You may also use **Ctrl+Delete**, if the editor is operating in CUA compatible mode.

Odyssey Menus (Text Editor)

Block|Indent

The block indent command increases the left margin offset of a block of text. For example, if the **Block|Indent** command is used on a selected paragraph which was indented to column five, then the indent is increased to column six. The **Block|Unindent** command would reverse this.

These commands are most useful when working with Odyssey script source files (or other program source files), in order to apply a uniform re-indentation on a bracketed section of program code.

You may use **Ctrl+K I** as a keyboard shortcut for this command.

Odyssey Menus (Text Editor)

Block|Unindent

The block unindent command decreases the left margin offset of a block of text. For example, if the **Block|Unindent** command is used on a selected paragraph which was indented to column six, then the indent is decreased to column five. The **Block|Indent** command would reverse this.

These commands are most useful when working with Odyssey script source files (or other program source files), in order to apply a uniform re-indentation on a bracketed section of program code.

You may use **Ctrl+K U** as a keyboard shortcut for this command.

Odyssey Menus (Text Editor)

Block|Paste to serial port

Transmits the currently marked and highlighted block (in the active edit window) through the serial port to a remote host, which must be ready to receive text at the time - note that Odyssey has no way of checking this first. The effective speed of transmission while pasting can be controlled by adjusting the ASCII character and line delays in the Setup|File transfer... configuration dialog.

Odyssey Menus

Text Editor Command menu

The editor Command menu provides further text editing commands.

Command|Find...
Command|Replace...
Command|Find again
Command|Go to line number...
Command|Compile script
Command|Convert word to upper case
Command|Convert word to previous case
Command|Set right margin at column
Command|Reformat paragraph

Odyssey Menus (Text Editor)

Command|Find...

This command allows you to search for any occurrence of a string of characters in the current document. The **Find String** dialog is displayed, prompting you for the string to find, the string you last searched for being offered as a default (blank if there was no previous search), and showing current values for several search options. See the description of the Find String dialog for more detailed information on these options. You should make any necessary changes to the dialog, then press <Enter> or click on OK to perform the search.

If the requested string was found, then the cursor will be moved to a point immediately after the string (if the search direction was forward), or on the start of the string (if the search direction was backwards).

You may use **Ctrl+Q F** as a keyboard shortcut for this command.

Odyssey Menus (Text Editor)

Command|Replace...

This command is used to search for a string of characters and then replace it with another string. This command is very similar to the Find command described above, except for the additional prompt in the **Find and Replace** dialog for the replacement string. There are also a couple of extra options used when replacing text - see the description of the Find and Replace dialog for further details.

You may use **Ctrl+Q A** as the keyboard shortcut for this command.

Odyssey Menus (Text Editor)

Command|Find again

This command repeats the last Find String, or the last Find and Replace, whichever was done most recently. For example, if a Find and Replace was used last then another Find and Replace will be performed.

You may use **Ctrl+L** as the keyboard shortcut for this command.

Odyssey Menus (Text Editor)

Command|Go to line number...

This command allows you to quickly jump to a specific line number within the current document. A dialog prompts you for the line number the editor should jump to.

You may use **Ctrl+Q G** as the keyboard shortcut for this command.

Odyssey Menus (Text Editor)

Command|Compile script

If the file loaded in the currently active editor is a source file for an Odyssey script (a .SRC or .SCR file), then this command allows you to compile that script within the editor. In the case of .SRC files, the compiled .SCR file is written to disk, assuming that the compilation was successful. In the case of .SCR files this command simply performs a syntax/semantics check of the script, but does not attempt to write a compiled version of the script to disk.

If a syntax error occurs while the script is being compiled then the compilation aborts, an error message is displayed on the status line, and the editor text cursor is set to the position of the syntax error. The error message is removed from the status line at the first keypress (note: that keypress is *not* discarded).

You may use **F9** as the keyboard shortcut for this command.

Odyssey Menus (Text Editor)

Command|Convert word to upper case

If the cursor is resting on a word in the currently active text editor, then this command forces that word to be all capitals.

You may use **Ctrl+U** as the keyboard shortcut for this command.

Odyssey Menus (Text Editor)

Command|Convert word to previous case

If the cursor is resting on a word in the currently active text editor, and that word has occurred previously in the document, then this command changes the case of that word so that it matches the most recent previous occurrence of that word. This command is most useful when editing Odyssey scripts or other program sources when you want to ensure that a consistant capitalisation is applied to a program symbol throughout.

You may use **Shift+F7** as the keyboard shortcut for this command.

Odyssey Menus (Text Editor)

Command|Set right margin at column

This command is used to set the right margin required for word wrap and paragraph reformatting operations. The right margin is set at the cursor column position in the currently active text editor - note that you are not asked for a column number. The right margin setting will be made permanent if you use the "**Save Setup**" option of the Setup menu.

Odyssey Menus (Text Editor)

Command|Reformat paragraph

This command causes the current paragraph in the currently active text editor to be reformatted such that the paragraph fits between the defined left and right margins. If "Right Justify" is enabled in the Options menu then the editor will format the paragraph such that each line in the paragraph is the same length, exactly meeting the right margin.

While the editor has a command to set a right margin, there is no equivalent command to set a left margin. Instead the editor will format every line to have the same indentation as the first line to be formatted. For example, to reformat a paragraph with a different left margin, move to the first character of the first line in the paragraph, type the space bar or backspace to adjust its indentation, then select **Command|Reformat paragraph** to reformat. All remaining lines in the paragraph will be given the same indentation.

The editor considers a paragraph to be any sequence of lines ending in a blank line. Since this is a pure ASCII editor it does not use special control characters to mark paragraph ends.

☞ The reformat paragraph command will not work unless word wrap mode is enabled.

This command leaves the cursor on the line *following* the blank line which ended the paragraph. This is hopefully where the next paragraph begins, and is intended to make it convenient to step through a document, reformatting each paragraph in turn.

You may use **Ctrl+B** as the keyboard shortcut for this command.

Odyssey Menus

Directory Viewer Menus

The Director Viewer menu bar contains the following submenus :-

Odyssey Menus (Directory View)

File menu

The following options are available on the Directory Viewer **File** menu.

File|Open...
File|Close all
File|Move...
File|Copy...
File|Delete...
File|Rename...
File|Show files of type...
File|Select...
File|Deselect all
File|View directory...
File|Create directory
File|DOS Shell
File|eXit from Odyssey

Odyssey Menus (Directory View)

File|Open...

If files are selected in the Directory Viewer file list panel, then the **File|Open** menu option causes those files to be opened. The meaning of "Open" depends on the file type (extension). If the file is a recognised bitmap type, then the enclosed image is displayed in a Bitmap Viewer wIDH_BTMAP_VIEWERindow. If the file is a Windows .HLP file then Odyssey runs the **WinHelp** application with that file as an argument. If the file is an Odyssey script (.SCR) file then Odyssey runs the script.   If the file is an application (.EXE) file then Odyssey asks Windows to run the program. Finally, if the file type is none of the above then Odyssey assumes it to be an ASCII text file and loads it into a text editor window.

Odyssey Menus (Directory View)

File|Close all

The **File|Close all** menu option closes all currently open document windows (directory viewers and bitmap viewers, text edit windows etc). This is quicker than closing each window individually.

Odyssey Menus (Directory View)

File|Move...

The **File|Move...** menu item may be used to move selected files to a new destination directory ("move" means that the originals are deleted after the copy is made). If no files are selected in the "File list" panel then Odyssey assumes that you wish to move the entire directory selected in the directory tree panel (however, Odyssey does not allow you to move the Odyssey directory!).

A dialog will prompt the user for the name of the destination directory, which must already exist. Use File|Create directory... first, if the destination directory does *not* already exist.

The destination directory for a move or copy must be different from the source directory - an item cannot be moved or copied onto itself, because DOS does not allow two files in the same directory to have the same name (copy), and moving a file to the same place it was moved *from* does not make sense.

Odyssey Menus (Directory View)

File|Copy...

The **File|Copy...** menu item may be used to copy selected files to a new destination directory. If no files are selected in the "File list" panel then Odyssey assumes that you wish to copy the entire directory selected in the directory tree panel.

A dialog will prompt the user for the name of the destination directory, which must already exist. Use File|Create directory... first, if the destination directory does *not* already exist.

☞ The destination directory for a move or copy must be different from the source directory - an item cannot be moved or copied onto itself, because DOS does not allow two files in the same directory to have the same name (copy), and moving a file to the same place it was moved *from* does not make sense.

Odyssey Menus (Directory View)

File|Delete...

The **File|Delete...** menu item may be used to delete selected files from the currently selected directory. If no files are selected in the "File list" panel then Odyssey assumes that you wish to delete the entire directory selected in the directory tree panel (however, Odyssey does not allow you to delete the Odyssey directory, nor does it allow you to delete the root directory of a drive or partition).

Odyssey always requests confirmation before it carries out the deletion. Unlike *File Manager*, Odyssey provides no option which disables the confirmation dialog.

Odyssey Menus (Directory View)

File|Rename...

The **File|Rename...** menu item may be used to rename selected files from the currently selected directory. A dialog will appear asking you for the new name to give to the file - if more than one file is selected then the "new name" must be a wildcard. If no files are selected in the "File list" panel then Odyssey assumes that you wish to rename the directory currently selected in the directory tree panel (however, Odyssey does not allow you to rename the Odyssey directory).

Odyssey Menus (Directory View)

File|Show files of type...

Odyssey defaults to displaying, in the file list panel, all the files in the current directory which match the search pattern ***.***. However, you can change this by selecting the **File|Show Files of Type...** menu item, or by clicking the "New file type" toolbar button.

In either case, you will be prompted for the new wildcard, and the files listed will then change to show only files matching the new pattern. For example, changing the pattern to *.TXT will cause the file list panel to be repainted, only listing files in the current directory which have the .TXT extension.

☞ The file list always includes a list of the subdirectories in the current directory, regardless of the current file type selection.

Odyssey Menus (Directory View)

File|Select...

Normally, you would select items in the Directory Viewer file list panel by clicking them with the mouse, or by using using standard Windows **Shift+<*movement key*>** sequences. However, the **File|Select...** menu item provides an alternative which may be quicker if you simply want to select all items in the directory which match a given wildcard - Odyssey will prompt you for the appropriate wildcard.

Note that this function is additive - ie. previously selected files are not deselected by this operation.

Odyssey Menus (Directory View)

File|Deselect all

The **File|Deselect all** menu item causes Odyssey to deselect all files in the file list panel of the currently active Directory Viewer window.

Odyssey Menus (Directory View)

File|View directory...

The **File|View directory...** menu item causes Odyssey to open another Directory Viewer window. It is possible to copy files by dragging them between opened viewer windows - this is useful if you want to copy files between different drives.

Odyssey Menus (Directory View)

File|Create directory

The **File|Create directory...** menu item may be used to create a new subdirectory of the directory currently selected in the Directory Tree panel of the currently active Directory Viewer. A dialog will appear, prompting the user to enter a name for the new directory.

Odyssey Menus (Directory View)

File|DOS Shell

When the **File|DOS Shell** menu item is used, Odyssey asks Windows to run a copy of the DOS command interpreter. This can be useful for users who want to quickly manipulate a few files before returning to Odyssey.

Odyssey Menus (Directory View)

File|eXit from Odyssey

Selecting this menu item tells Odyssey to shut itself down. The **ALT+X** keystroke can be used as a shortcut for this menu item, as can clicking on the "Exit Odyssey" toolbar button. However, Odyssey asks for confirmation if you use a shortcut method.

If there are any "modified" text editor windows opened then Odyssey will ask if you wish to save those files.

Odyssey Menus (Directory View)

Upload/Download menu

If files are selected in the "file list" panel of the currently active Directory Viewer then you can upload those files to a remote host by pulling down the **Upload** menu and selecting a protocol. If no files are selected then Odyssey assumes that you wish to upload all the files in the directory currently selected in the Directory Tree panel. Note that you must use a batch protocol if you wish to upload multiple selected files.

Items in the **Download** menu are always disabled. This submenu only appears for reasons of symmetry with the Terminal window menu.

If you wish to perform a Zmodem upload then Odyssey provides a toolbar button as a quicker way of selecting that option. You may also drag selected files onto the Zmodem button with similar results.

Odyssey Menus (Directory View)

Command Menu

The **Command** menu provides options which affects what Odyssey displays in the file list panel of the currently active Directory Viewer window.

Command|Show filenames only
Command|Show all file details
Command|Expand branch
Command|Collapse branch
Command|Sort by name
Command|Sort by type
Command|Sort by size
Command|Sort by date

Odyssey Menus (Directory View)

Command|Show filenames only

An Odyssey Directory Viewer can list files in two formats:

In ***Filename Only*** format, the file listing is a multicolumn format containing just the tail part of the name of each file matching the current file type mask. This allows you to see more files, but doesn't give you any detail about individual files apart from their names.

In ***All File Details*** format, the file listing switches to a single column format, with the attributes of each file listed in detail. The details include the size in bytes of the file, the file date and time stamp, plus the file permissions.

Odyssey Menus (Directory View)

Command|Show all file details

An Odyssey Directory Viewer can list files in two formats:

In ***Filename Only*** format, the file listing is a multicolumn format containing just the tail part of the name of each file matching the current file type mask. This allows you to see more files, but doesn't give you any detail about individual files apart from their names.

In ***All File Details*** format, the file listing switches to a single column format, with the attributes of each file listed in detail. The details include the size in bytes of the file, the file date and time stamp, plus the file permissions shown as a field of attribute characters. The characters and their meanings are as follows :-

***Attribute character*.**		***Meaning***
A	-	The file has the "archive" attribute bit set, meaning that it has been modified since the last time it was backed up.
H	-	The file is hidden (it does not appear in a normal DOS directory listing).
S	-	The file is a system file.
R	-	The file is marked read-only.

Odyssey Menus (Directory View)

Command|Expand branch

If a Directory Tree folder has an icon like this: 🗀 or this:
🗁, then this tells you that the associated directory has subdirectories which are not currently displayed in the directory tree (they *will* be displayed in the file listing). You can make the directory tree panel display those subdirectories by selecting the **Command|Expand branch** menu item.

You can do the same thing by double-clicking on the folder you want to expand, or by pressing the **+** key (the key on the numeric keypad is best for this), which expands the currently highlighted folder.

Odyssey Menus (Directory View)

Command|Collapse branch

If you no longer want to see the subdirectories of a particular folder in the Directory Tree panel, then you can hide those entries by highlighting the parent folder and selecting the **Command|Collapse branch** menu item.

You can get the same result by double-clicking on the parent folder or by pressing the **-** key (the key on the numeric keypad is best for this).

Odyssey Menus (Directory View)

Command|Sort by name

By default, the Odyssey directory viewer lists files sorted in alphabetical order of their name ("a..." first). However, other sorting methods are supported. To change the sorting method, pull down the directory viewer Command menu, and choose one of the **Sort by xxxx** options - a check mark will be shown beside the menu item relating to the current sorting method.

If **Command|Sort by name** is selected then files are listed in alphabetical order of their name. This is the default sorting order, as described above.

Odyssey Menus (Directory View)

Command|Sort by type

By default, the Odyssey directory viewer lists files sorted in alphabetical order of their name ("a..." first). However, other sorting methods are supported. To change the sorting method, pull down the directory viewer Command menu, and choose one of the **Sort by xxxx** options - a check mark will be shown beside the menu item relating to the current sorting method.

If **Command|Sort by type** is selected then files are sorted first by their extension, and then (within a group of files with the same extension) by name. This has the effect of ensuring that files with the same extension are listed next to each other.

Odyssey Menus (Directory View)

Command|Sort by size

By default, the Odyssey directory viewer lists files sorted in alphabetical order of their name ("a..." first). However, other sorting methods are supported. To change the sorting method, pull down the directory viewer Command menu, and choose one of the **Sort by xxxx** options - a check mark will be shown beside the menu item relating to the current sorting method.

If **Command|Sort by size** is selected then files are sorted so that the largest files appear at the top of the list, and the smallest files at the bottom. If your disk is nearly full then finding the largest files you no longer need, and deleting them, has the quickest payoff in terms of recovering disk space.

Odyssey Menus (Directory View)

Command|Sort by date

By default, the Odyssey directory viewer lists files sorted in alphabetical order of their name ("a..." first). However, other sorting methods are supported. To change the sorting method, pull down the directory viewer Command menu, and choose one of the **Sort by xxxx** options - a check mark will be shown beside the menu item relating to the current sorting method.

If **Command|Sort by date** is selected then files are sorted by age, with the oldest files at the top of the list. You might use this to find files which you haven't needed in a long time, and liberating the disk space they occupy.

Odyssey Menus

Bitmap Viewer Menus

These are the items available on the menu bar when a Bitmap Viewer is the active document window.

File menu
Edit menu
Setup menu
Command menu
Window menu
Help menu

Odyssey Menus (Bitmap Viewer)

File menu

Below are the options available on the **File menu** when the current document window is a Bitmap Viewer.

File|Open...
File|Close all
File|Save
File|Save as...
File|Print bitmap
File|Delete bitmap...
File|Text logging...
File|Printer logging
File|View directory...
File|DOS Shell
File|eXit from Odyssey...

Odyssey Menus (Bitmap Viewer)

File|Open...

The **File|Open...** menu item is disabled when a Bitmap Viewer is the active document window.

Odyssey Menus (Bitmap Viewer)

File|Close all

The **File|Close all** menu option closes all currently open document windows (directory viewers and bitmap viewers, text edit windows etc). This is quicker than closing each window individually.

Odyssey Menus (Bitmap Viewer)

File|Save

The **File|Save** menu item is disabled when a Bitmap Viewer is the active document window.

Odyssey Menus (Bitmap Viewer)

File|Save as...

The **File|Save as...** menu item is disabled when a Bitmap Viewer is the active document window.

Odyssey Menus (Bitmap Viewer)

File|Print bitmap

You can print a bitmap by reading the bitmap file into an Odyssey bitmap viewer and then selecting **File|Print bitmap...** from the Bitmap Viewer main menu, or by clicking on the "print" toolbar button.

Odyssey Menus (Bitmap Viewer)

File|Delete bitmap...

Sometimes, having viewed a bitmap image you have just downloaded, you will decide that you do not wish to keep that image. You can delete the bitmap file when the viewer is active by selecting **File|Delete bitmap...** or by clicking the "trashcan" toolbar button. You will be asked to confirm that you really want to delete the bitmap file.

You are not allowed to delete a FAX file if the bitmap viewer was opened in order to preview a FAX before transmission, because the FAX server always deletes such temporary fax files itself.

Odyssey Menus (Bitmap Viewer)

File|Text logging...

The **File|Text logging...** menu item is disabled when a Bitmap Viewer is the active document window.

Odyssey Menus (Bitmap Viewer)

File|Printer logging

The **File|Printer logging** menu item is disabled when a Bitmap Viewer is the active document window.

Odyssey Menus (Bitmap Viewer)

File|View directory...

The **File|View directory...** menu item causes Odyssey to open a directory viewer window, which allows the user to copy, rename and delete files etc. See elsewhere for a detailed description of the Odyssey Directory Viewer module.

Odyssey Menus (Bitmap Viewer)

File|DOS Shell

When the **File|DOS Shell** menu item is used, Odyssey asks Windows to run a copy of the DOS command interpreter. This can be useful for users who want to quickly manipulate a few files before returning to Odyssey.

Odyssey Menus (Bitmap Viewer)

File|eXit from Odyssey...

Selecting this menu item tells Odyssey to shut itself down. The **ALT+X** keystroke can be used as a shortcut for this menu item, as can clicking on the "Exit Odyssey" toolbar button. However, Odyssey asks for confirmation if you use a shortcut method.

If there are any "modified" text editor windows opened then Odyssey will ask if you wish to save those files.

Odyssey Menus (Bitmap Viewer)

Edit menu

These are the items available on the **Edit** menu when the active document window contains a Bitmap Viewer.

Edit|Review buffer
Edit|Cut
Edit|Copy
Edit|Paste

Odyssey Menus (Bitmap Viewer)

Edit|Review buffer

The **Edit|Review buffer** menu item is disabled when a Bitmap Viewer is the active document window.

Odyssey Menus (Bitmap Viewer)

Edit|Cut

The **Edit|Cut** menu item is disabled when a Bitmap Viewer is the active document window.

Odyssey Menus (Bitmap Viewer)

Edit|Copy

You can copy a bitmap image to the clipboard by first reading the bitmap file into an Odyssey bitmap viewer and then selecting **Edit|Copy...** from the Bitmap Viewer menu.

The bitmap is always copied to the clipboard in DIB (CF_DIB) format, in order that palette information is preserved for the target application.

Odyssey Menus (Bitmap Viewer)

Edit|Paste

The **Edit|Paste** menu item is disabled when a Bitmap Viewer is the active document window.

Odyssey Menus (Bitmap Viewer)

Command menu

The following are the items available on the **Command** menu when the active document window contains a Bitmap Viewer.

Command|First page
Command|Previous page
Command|Next page
Command|Last page
Command|Top of page
Command|Bottom of page

Odyssey Menus (Bitmap Viewer)

Command|First page

If the bitmap file contains more than one page (eg. a multi-page FAX file), then **Command|First page** can be used to move to the first page.

You may use **Ctrl+Home** as a keyboard shortcut for this command.

Odyssey Menus (Bitmap Viewer)

Command|Previous page

If the bitmap file contains more than one page (eg. a multi-page FAX file), then **Command|Previous page** can be used to move to the previous page, if any.

You may use **Ctrl+PgUp** as a keyboard shortcut for this command.

Odyssey Menus (Bitmap Viewer)

Command|Next page

If the bitmap file contains more than one page (eg. a multi-page FAX file), then **Command|Next page** can be used to move to the next page in the bitmap file, if any.

You may use **Ctrl+PgDn** as a keyboard shortcut for this command.

Odyssey Menus (Bitmap Viewer)

Command|Last page

If the bitmap file contains more than one page (eg. a multi-page FAX file), then **Command|Last page** can be used to move to the last page.

You may use **Ctrl+End** as a keyboard shortcut for this command.

Odyssey Menus (Bitmap Viewer)

Command|Top of page

The **Command|Top of page** command ensures that the top left of the bitmap page is aligned with the top left of the viewing window - ie. this command takes you to the top of the bitmap page.

You may use **Shift+Home** as a keyboard shortcut for this command.

Odyssey Menus (Bitmap Viewer)

Command|Bottom of page

The **Command|Bottom of page** command ensures that the bottom left of the bitmap page is aligned with the bottom left of the viewing window - ie. this command takes you to the bottom of the bitmap page.

You may use **Shift+End** as a keyboard shortcut for this command.

Odyssey Menus

Fax Server Menu

These are the items available on the menu bar when the Fax Server is the active document window.

File menu
Edit menu
Setup menu
Command menu
Window menu
Help menu

Odyssey Menus (Fax Server)

File menu

These are the items available on the Fax Server file menu. Note that many are not applicable to Fax Server mode and are therefore disabled.

File|Open...
File|Close all
File|Save
File|Save as...
File|Print screen
File|Text logging...
File|Printer logging
File|View directory...
File|DOS Shell
File|eXit from Odyssey...

Odyssey Menus (Fax Server)

File|Open...

The terminal window **File|Open...** menu item is used to load a text file into an Odyssey text editor. The user is presented with a standard File Selector dialog from which to make his choice of files, and a text editor window is then opened to display and edit that file. The filename entered into the dialog need not already exist - if the file does not exist then a new file of that name is created.

If you were looking for a way to open a Fax document, just double-click on the appropriate line in the "Received FAXes" window.

Odyssey Menus (Fax Server)

File|Close all

This menu item is not applicable to Fax Server mode, and is therefore disabled.

Odyssey Menus (Fax Server)

File|Save

This menu item is not applicable to Fax Server mode, and is therefore disabled.

Odyssey Menus (Fax Server)

File|Save as...

This menu item is not applicable to Fax Server mode, and is therefore disabled.

Odyssey Menus (Fax Server)

File|Print screen

This menu item is not applicable to Fax Server mode, and is therefore disabled.

Odyssey Menus (Fax Server)

File|Text logging...

This menu item is not applicable to Fax Server mode, and is therefore disabled.

Odyssey Menus (Fax Server)

File|Printer logging

This menu item is not applicable to Fax Server mode, and is therefore disabled.

Odyssey Menus (Fax Server)

File|View directory...

The **File|View directory...** menu item causes Odyssey to open a directory viewer window, which allows the user to copy, rename and delete files etc. See elsewhere for a detailed description of the Odyssey Directory Viewer module.

Odyssey Menus (Fax Server)

File|DOS Shell

When the **File|DOS Shell** menu item is used, Odyssey asks Windows to run a copy of the DOS command interpreter. This can be useful for users who want to quickly manipulate a few files before returning to Odyssey. The keystroke **ALT+O** can be used as a shortcut for this menu item.

Odyssey Menus (Fax Server)

File|eXit from Odyssey...

Selecting this menu item tells Odyssey to shut itself down. The **ALT+X** keystroke can be used as a shortcut for this menu item, as can clicking on the "Exit Odyssey" toolbar button. However, Odyssey asks for confirmation if you use a shortcut method.

If there are any "modified" text editor windows opened then Odyssey will ask if you wish to save those files.

Odyssey Menus (Fax Server)

Command menu

Below are the items available on the **Command** menu when the active document window is the Fax Server.

Command|Send Fax...
Command|View Fax...
Command|Print Fax...
Command|Export Fax to PCX/TIF...
Command|Delete Fax...

Odyssey Menus (Fax Server)

Command|Send Fax...

The menu item **Command|Send Fax...** is used after you have prepared a text file of PCX/TIFF image, and you want to send it to a Fax destination. This menu item leads to the Send Fax dialog, whose description you should read for further information.

You can use the "Send Fax" toolbar button as a shortcut for this menu item.

Odyssey Menus (Fax Server)

Command|View Fax...

The **Command|View Fax...** menu item leads to a standard File Selector dialog, which you can use to select the Fax file you wish to view. You can use the "View Fax" toolbar button as a shortcut for this command.

A quicker way to select a Fax document for viewing is to simply double-click on the document where it appears in the "Received FAXes" window - however this method only works in the FAX file is still in the Fax directory.

Odyssey Menus (Fax Server)

Command|Print Fax...

The **Command|Print Fax...** menu item leads to the Print Fax dialog, which allows you to select the Fax file and also the range of pages within that file which you wish to print. You can use the "printer" toolbar button as a shortcut for this menu item.

You can also print individual Fax pages from the Bitmap Viewer.

Odyssey Menus (Fax Server)

Command|Export Fax to PCX/TIF...

The **Command|Export Fax...** menu item leads to the Export Fax dialog, which allows you to select the Fax file and also the range of pages within that file which you wish to export, as well as the graphics file format (PCX or TIFF) you wish to use. You can use the "Export Fax" toolbar button as a shortcut for this menu item.

Odyssey Menus (Fax Server)

Command|Delete Fax...

The **Command|Delete Fax...** menu item leads to a standard File Selector dialog, which you can use to select the Fax file you wish to delete. You can use the "trashcan" toolbar button as a shortcut for this command.

Odyssey Menus

Archive Viewer Menus

These are the items available on the menu bar when an Archive Viewer is the active document window.

File menu
Edit menu
Setup menu
Command menu
Window menu
Help menu

Odyssey Menus (Archive Viewer)

Command Menu

Below are the items available on the **Command** menu when the active document window is an Archive Viewer :-

Command|Extract files...
Command|Set decryption password...
Command|Show files of type...
Command|Unsorted list
Command|Sort by name
Command|Sort by type
Command|Sort by size
Command|Sort by date

Odyssey Menus (Archive Viewer)

Command|Extract files...

The **Command|Extract files...** menu item is used when you wish to extract or unpack files from the archive file associated with the currently active Archive Viewer window. Note that you do not need to extract files simply in order to view them - if you merely wish to view a file in an archive then just double-click on the listbox entry for the compressed file you wish to view.

This menu item leads to the Extract files... dialog, which is described in detail elsewhere.

There is also an "*Extract files...*" toolbar button, which may be used as a shortcut for selecting this menu item.

Odyssey Menus (Archive Viewer)

Command|Set decryption password...

The **Command|Set decryption password...** menu item is applicable to ZIP archives only.

The ZIP format allows files to be encrypted with a password string. If that is done, then a person wishing to view or extract the encrypted file must know what that password is, since there is no way to decompress the file without it.

When you ask Odyssey to unpack an archive, it starts by displaying the Extract files... dialog, which includes a text field into which you can enter the password for decryption (if required). However, when viewing individual files no dialog is displayed, so in order to successfully view encrypted files you must first tell Odyssey the password, using this menu option to do so. Selecting this menu option causes Odyssey to prompt you for the password.

Note that an individual password must be set separately (when necessary) for every Archive Viewer window currently open. This password is forgotten when the window is closed (for obvious security reasons, Odyssey does *not* save entered passwords in order to display them as defaults later), Odyssey does however associate the entered password with an **Archive Viewer** window for as long as that window remains open.

Finally, note that while Odyssey can quickly determine whether the password entered is wrong, it is not possible for us to determine what your correct password is if you have forgotten it.

Odyssey Menus (Archive Viewer)

Command|Show files of type...

Odyssey defaults to displaying, in the file list panel of an Archive Viewer window, all the files stored in the associated archive which match the search pattern ***.***. However, you can change this by selecting the **File|Show Files of type...** menu item, or by clicking the "*Show files of type...*" toolbar button.

In either case, you will be prompted for the new wildcard, and the files listed will then change to show only files which match the new pattern. For example, changing the pattern to *.TXT will cause the file list panel to be repainted, only listing files in the archive which have the .TXT extension.

☞ When you ask Odyssey to "Unpack all files" from an archive, it will only ever unpack files which are currently listed in the **Archive Viewer** window. In other words, following the example above, unpacking all files would mean "Unpack all files in the archive which have the '.TXT' extension".

Odyssey Menus (Archive Viewer)

Command|Unsorted list

By default, the Odyssey Archive Viewer lists files unsorted, ie. listed in the order in which the files are stored in the archive. However, a number of sort order options are supported. To change the sort order, pull down the **Archive Viewer** Command menu, and choose one of the **Sort by xxxx** options - a check mark will be shown beside the menu item relating to the currently selected sorting method.

If **Command|Unsorted list** is selected then files are listed in the order in which they are stored in the archive. This is the default sorting order, as described above.

Odyssey Menus (Archive Viewer)

Command|Sort by name

By default, the Odyssey Archive Viewer lists files unsorted, ie. listed in the order in which the files are stored in the archive. However, a number of sort order options are supported. To change the sort order, pull down the **Archive Viewer** Command menu, and choose one of the **Sort by xxxx** options - a check mark will be shown beside the menu item relating to the currently selected sorting method.

If **Command|Sort by name** is selected then files are listed in alphabetical order of their name, ie. "Axxxxx" first.

Odyssey Menus (Archive Viewer)

Command|Sort by type

By default, the Odyssey Archive Viewer lists files unsorted, ie. listed in the order in which the files are stored in the archive. However, a number of sort order options are supported. To change the sort order, pull down the **Archive Viewer** Command menu, and choose one of the **Sort by xxxx** options - a check mark will be shown beside the menu item relating to the currently selected sorting method.

If **Command|Sort by type** is selected then files are sorted first by their extension, and then (within a group of files sharing the same extension), by name.

Odyssey Menus (Archive Viewer)

Command|Sort by size

By default, the Odyssey Archive Viewer lists files unsorted, ie. listed in the order in which the files are stored in the archive. However, a number of sort order options are supported. To change the sort order, pull down the **Archive Viewer** Command menu, and choose one of the **Sort by xxxx** options - a check mark will be shown beside the menu item relating to the currently selected sorting method.

If **Command|Sort by size** is selected then files are listed in decreasing order of their uncompressed size (the column labeled as "Length" in the Archive Viewer window). Ie. the largest file appears first in the list.

Odyssey Menus (Archive Viewer)

Command|Sort by date

By default, the Odyssey Archive Viewer lists files unsorted, ie. listed in the order in which the files are stored in the archive. However, a number of sort order options are supported. To change the sort order, pull down the **Archive Viewer** Command menu, and choose one of the **Sort by xxxx** options - a check mark will be shown beside the menu item relating to the currently selected sorting method.

If **Command|Sort by date** is selected then files are listed in date/time order, the oldest file being listed first.

Odyssey Help

Reference

Choose one of the reference topics listed below:

Odyssey Help

Cabling Requirements

The cabling requirements for successful communications depends very much on exactly what kind of devices you are trying to connect. The topics below list precise requirements for a variety of device types which you are likely to encounter :-

Modem Cables:

- PC (25 pin) to Modem (25 pin)
- PC (9 pin) to Modem (25 pin)

Null Modem Cables (ie. PC-PC):

- PC (25 pin) to PC (25 pin)
- PC (25 pin) to PC (9 pin)
- PC (9 pin) to PC (9 pin)

When selecting a cable to use for modem communications you should be extremely wary of using a ready made one designed for use with a serial printer. These are often totally unsuited for use with a modem. For example, they often have the carrier pin tied high to satisfy the printer or IBM PC BIOS, which is totally wrong for a modem cable, if you want the modem to be able to signal the presence or not of a carrier to the PC (as Odyssey requires).

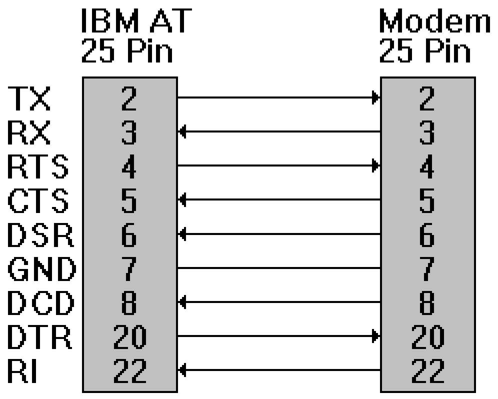
If you are not sure how a particular cable is wired up then open it up and check it. If it has more pins connected than are shown in these diagrams, that is no problem. If however you see pins connected together at one end then do not use that cable for comms unless the connections match the appropriate cable outlined in one of the above topics. If the cable has moulded plastic connectors that cannot be opened, then do not trust it unless you know that it was designed for modem rather than printer use.

The important signals to check are DTR, RTS and TX coming from the PC, and DSR, CTS, DCD, RI, and RX coming from the modem. All of the cable diagrams presented in this appendix are designed to make sure that these signals are presented properly to each side of the connection.

Odyssey Cable Requirements

PC (25 pin) to Modem (25 pin)

Connecting a 25 pin PC port to a modem uses the simplest cable there is, a straight ribbon cable with snap on connectors. If you cannot provide yourself with this then the following cable should be the minimum full specification cable for a PC. The arrows on the connectors show the direction of the signal.

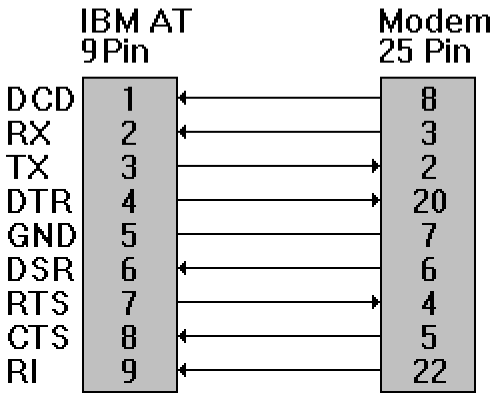


TX	-	Transmit
RX	-	Receive
RTS	-	Request to send
CTS	-	Clear To Send
DSR	-	Data Set Ready
GND	-	Ground
DCD	-	Data Carrier Detect
DTR	-	Data Terminal Ready
RI	-	Ring Indicator

Odyssey Cable Requirements

PC (9 pin) to Modem (25 pin)

Connecting an 9-pin PC port to a modem is more work, because few (if any) modems on the market come with AT style 9 pin connectors. The following cable will do the trick.

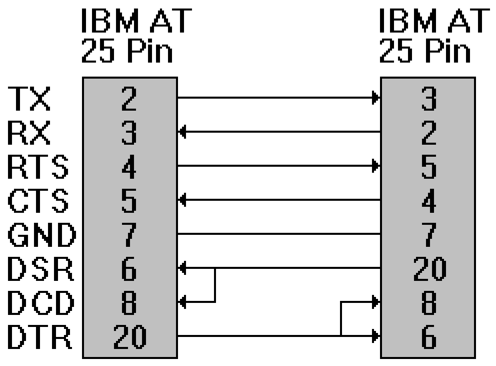


TX	-	Transmit
RX	-	Receive
RTS	-	Request to send
CTS	-	Clear To Send
DSR	-	Data Set Ready
GND	-	Ground
DCD	-	Data Carrier Detect
DTR	-	Data Terminal Ready
RI	-	Ring Indicator

Odyssey Cable Requirements

PC (25 pin) to PC (25 pin)

The remaining cables are all "null modem" cables, for directly connecting computers together, starting with PC-25pin to PC-25pin.

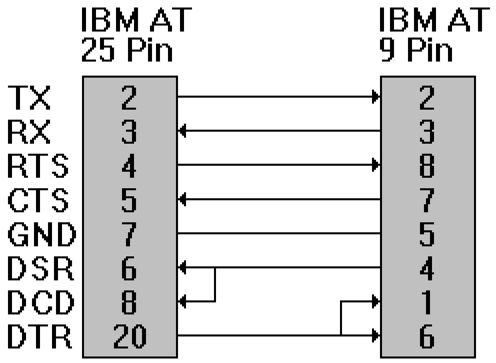


TX	-	Transmit
RX	-	Receive
RTS	-	Request to send
CTS	-	Clear To Send
DSR	-	Data Set Ready
GND	-	Ground
DCD	-	Data Carrier Detect
DTR	-	Data Terminal Ready
RI	-	Ring Indicator

Odyssey Cable Requirements

PC (25 pin) to PC (9 pin)

This null modem cable connects PC-25pin to PC-9pin.

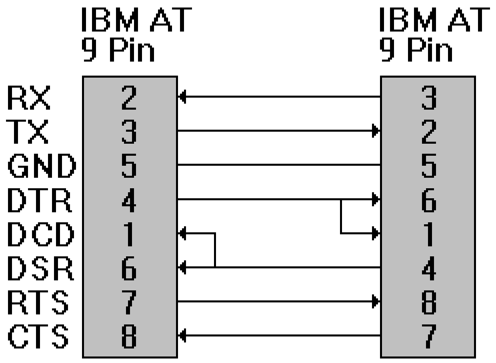


TX	-	Transmit
RX	-	Receive
RTS	-	Request to send
CTS	-	Clear To Send
DSR	-	Data Set Ready
GND	-	Ground
DCD	-	Data Carrier Detect
DTR	-	Data Terminal Ready
RI	-	Ring Indicator

Odyssey Cable Requirements

PC (9 pin) to PC (9 pin)

This null modem cable connects PC-9pin to PC-9pin.



TX	-	Transmit
RX	-	Receive
RTS	-	Request to send
CTS	-	Clear To Send
DSR	-	Data Set Ready
GND	-	Ground
DCD	-	Data Carrier Detect
DTR	-	Data Terminal Ready
RI	-	Ring Indicator

Odyssey Help

Acknowledgments, Trademarks etc

The Odyssey ZIP, ARC and LZH archive readers were written from scratch by the Odyssey author, and are definitely ***not*** derived from any freeware or public domain sources you may be aware of. However, the author wishes to acknowledge his gratitude in particular towards the Info-ZIP group (who distribute copyrighted, but freeware sources for ZIP) - while I didn't use their code, it did serve an extremely valuable role as additional documentation for the ZIP algorithms, over and above the rather terse APPNOTE.TXT provided by PKWare. There was certainly nothing wrong with their code, I simply prefer to write such complex stuff myself, since that is the only way I can be sure of fully understanding what it does. I would also, naturally, like to thank Phil Katz of PKWare for making APPNOTE.TXT available, and hope that he is able to continue doing so in future.

Likewise, public domain or freeware sources for **ARC** and **LZH** readers served as valuable documentation for those formats, even though I didn't actually need to use their code. The author of the ARC sources I used is not known (DEARC5.PAS, readily available from many hosts). The LZH sources I used initially came from the IBMPRO forum on Compuserve, and were written by Haruyasu Yoshizaki, who also wrote LHARC and LHA. Another important source of LZH documentation was some sources I found on the Internet, written by Thomas Quester in Germany, who had done a lot to work out the LHA -lh5- algorithm. Thomas had done me a favor by recoding the latter algorithm in C, making it far easier for me to understand than Yoshi's assembler equivalents.

A word about my JPEG reader as well: that too is entirely original, based on the **JPEG** standard as printed in the book "*The JPEG Still Image Data Compression Standard*", by Pennebaker and Mitchell (published in 1993 by Van Nostrand Reinhold, ISBN: 0-442-01272-1). That book is thoroughly recommended to anyone interested in JPEG - not only does it publish the standard in full, it also contains a wealth of information on how to write the most efficient DCT and inverse **DCT** (IDCT) functions possible (ie. those involving the least number of multiplications).

Odyssey Features

Viewdata (PRESTEL) Emulation

For compatibility with previous versions of Odyssey, the Viewdata terminal emulation module is called PRESTEL.TRM, and dialing directory entries or scripts which wish to select Viewdata emulation should select the **PRESTEL** emulation as before.

You may choose to use the Viewdata emulation in either *full screen* or *split screen* modes. In the latter mode the right hand side of the split screen is used to display a help screen at all times. In full screen modes the help screen can be viewed by pressing a key (described fully later).

Viewdata systems are based on *frames*. That is, the data is displayed using a complete screenful at a time. When we mention the word frame in this supplement we refer to what you can see on your screen.

Odyssey Viewdata Emulation

Systems you can Access

The standard supported by the Odyssey Viewdata emulation is originally that of the British Telecom PRESTEL system, but is now used in many countries. Some of these countries are :-

Country	**Service**
Australia	Discovery 40
UK	Prestel
Denmark	Teledata
Finland	Videotex
Italy	Videotel
Netherlands	Viditel
Norway	Teledata
Sweden	Videotex

In addition, a number of private Viewdata services have appeared, such as private databases and home banking systems. In this supplement we refer to all such services as ***Viewdata*** services.

Some countries use a Viewdata system of another type. The Odyssey Viewdata emulation is *not* compatible with the French Teletel (Minitel) system, or with the German, Austrian, Swiss or Spanish systems (BTX, VTX, IBERTEX).

Odyssey Viewdata Emulation

Split Baudrates

Some of the access numbers for Viewdata services use only V.23 split baudrate, ie. you receive data at 1200 bps but send at 75 bps. Odyssey can use that type of system only if your modem offers *speed buffering*, that is, the modem will handle the V.23 part, but will talk to Odyssey at 1200 baud in both send and receive, buffering 1200 bps data from Odyssey and sending it on at 75 bps to the service.

☞ The UK Prestel system has in recent years starting upgrading their access numbers to 2400 bps full duplex, and perhaps even higher speeds by the time you read this. You would be well advised to start using those numbers, if your modem is capable of supporting the higher speeds.

Odyssey Viewdata Emulation

Running the Viewdata Emulation

See the Terminal Emulation topics for information on how to select Viewdata emulation. When the emulation first loads, the screen will be divided into two distinct parts; the left hand side is where you will see the data coming from the Viewdata service. If you wish, this data can be displayed across the full screen by pressing **CTRL+F10**.

The right hand side of the display shows the main help menu. In full screen mode you can view this help screen by pressing **ALT+Z** (press any key to restore the terminal screen). In split screen mode the help screen is always visible.

Odyssey Viewdata Emulation

Connecting to the Viewdata Service

Before connecting to a Viewdata service you should first enter the details of the service into an Odyssey Dialing Directory record. Remember in particular to set the correct telephone number, baud rate and parity (most Viewdata services currently expect even parity), and make sure you have selected the PRESTEL terminal in the "emulation" field.

Some Viewdata services send the **Enquiry response** character (ENQ, or ascii code 5) to your terminal, which requests the terminal to send some sort of identification sequence (usually a user number, password or account number). When the Odyssey Viewdata emulation receives this character it will transmit whatever is defined as macro 0 in the Setup|Macros dialog. You can define this macro to be anything you like - including nothing, if you prefer to enter this reply manually.

If you want to send commands to your modem while offline, and while running the Viewdata emulation, please remember that the emulation maps the enter key as '#', which will prevent you from using it to complete modem commands. You can however use the alternative enter key on the numeric keypad, which will still send the normal carriage return character. We recommend that unless you are using the Viewdata emulation offline for a specific reason (perhaps to use the frame editor) that you have Odyssey normally default to a "standard" emulation offline (such as TTY), and have Odyssey switch to Viewdata emulation only when it connects to a Viewdata service. Odyssey will perform this switch at the right time automatically, provided that Prestel emulation is selected in the dialing directory entry.

To access specific pages or frames on a Viewdata service you must select it by number. This is done by keying '*', followed by the number, followed by hash (#) or underscore (_). As mentioned above, the Odyssey Viewdata emulation will send hash for you when you press the enter key. Viewdata services were not originally designed to be called from PC's, which is the reason for the unusual, if not awkward, choice of keys.

Odyssey Viewdata Emulation

The User Help Screen

The main help screen is not the only help you can get in the Odyssey Viewdata module. There is also a help screen which you can view when using the frame editor, and you can even create your own screen, and access it by keying **ALT+U**.

You may want to create a user help screen if there is some information you would like to have at hand when necessary. It may for instance be a list of interesting pages on your Viewdata service, or a table of semi-graphic characters if you design frames or graphic mail.

The User Help screen behaves like any other help screen: in split screen mode it will stay on the screen until you decide to remove it. In full screen mode the user help goes away when you press a key. You can view the user help screen at any time while using the Odyssey Viewdata module - even while using the Viewdata frame editor.

If you wish to restore the standard help screen for whichever Viewdata mode you are currently using, just press **ALT+Z**.

To modify the User Help Screen you should :-

- Load the file **VDUSER.PIC** into the frame editor and edit it.

- When you are done you should save the edited picture as VDUSER.PIC (no other name will do), and you must also save it in the Odyssey program files directory.

- Drop out to DOS, go to the Odyssey directory and run the **PIC2HLP** program. This will convert the VDUSER.PIC format into a new file VDUSER.HLP which is suitable for use as a Viewdata help file.

{mbc finger.mrb}Note that the Viewdata HLP files are compatible with the DOS version of the Odyssey Viewdata emulation, and are NOT compatible with Windows HLP files.

Odyssey Viewdata Emulation

Creating Keyboard Macros

You can use the standard Odyssey keyboard macro facility (see Setup|Edit Macros) to create macros for use on a Viewdata system. Remember however that macro 0 is reserved for use as the enquiry response string; the remaining nine macros may be used for your own purposes.

One other point you should be aware of is that the Odyssey macro menu assumes the ASCII character set, which is suitable for all services - except Viewdata. In particular, if you want a macro to send the Viewdata hash (#) character your macro should actually contain underscore (_), which is the ASCII character whose code corresponds to the Viewdata code for a hash.

Odyssey Viewdata Emulation

Logging and then Viewing a Session

You can use the standard Odyssey text logging feature to log a Viewdata session. However, you should make sure that you select *raw* logging mode. This will make sure that Odyssey retains all the necessary Viewdata terminal control sequences and characters which will allow you to play back the session offline using the Viewdata *View Log* feature. Odyssey will automatically enable raw logging when you select PRESTEL as the terminal type in the emulation field of a dialing directory entry.

You can view (play back) any Viewdata log file after it has been saved (see the Redisplay frame feature if you just want to view a couple of recent frames without closing the log file).

To view a log file press **ALT+V**, and then enter the name you gave to the log file. As you view the log, each frame will be shown one at a time, each staying on the screen for a short while before being replaced by the next one in the file, and so on. If you want to pause at a particular frame then just press the <**space**> bar when it appears. When the frame is paused like that you will also be given the opportunity to print the frame, or save it to disk as a picture file which you can edit using the frame editor.

You can also use the standard Odyssey "Playback" feature (accessed from the Command menu) to play back a Viewdata log. The standard playback mode is different from the specialised Viewdata viewer in that it updates the display on a character by character basis, rather than frame by frame. This is quite useful if you want to play back a Prestel "dynamic" frame (frames containing animation etc), and be able to see the dynamic aspect.

There is a trick you can use if you want to convert a raw Viewdata log into an ASCII log (ie. a file you can edit in an ASCII editor). The trick is to use the standard Odyssey playback mode to play back the raw log, *while at the same time logging the output of the playback!*. The new log should naturally be creating with raw logging disabled, otherwise you will have simply discovered a rather slow way of copying the file! You must also remember to give a different name for the second log file, otherwise the best that will happen is that the new log will overwrite the first as it is being read, producing non too wonderful results.

Odyssey Viewdata Emulation

Screen Snapshots

If you only want to save a single frame, you can do so more simply than creating a log of the entire session. You can instead save a *screen snapshot* to disk. To save a snapshot you simply press **ALT+D** when the frame you wish to save appears on the screen, while online to the service. You will be asked for a name to give to the snapshot file, which will be given a .PIC extension by default. You can load this picture file into the frame editor later when you want to look at it.

Odyssey Viewdata Emulation

Printing the Viewdata Screen

You can print the terminal screen at any time by pressing **Ctrl+P**. You must first ensure that the printer is online, and of course if the printer is attached to a serial port it must not be sharing the same port as the modem, if the modem is online at the time. The Viewdata print screen routine maps many Viewdata mosaic characters to IBM PC characters before printing. This can produce quite acceptable results, but does of course require that your printer can cope with IBM PC non-ASCII characters. If your printer cannot handle that then you can force the Odyssey Viewdata module to print using ASCII characters only by toggling the printer type to non-IBM, by keying **ALT+P**.

Of course, if your printer is capable of handling bitmaps then you can just print the screen using normal Windows procedures.

Odyssey Viewdata Emulation

Redisplaying Frames

The Odyssey Viewdata emulation allows you to redisplay the last twelve frames which appeared on the terminal screen. You can use this feature to check data that has disappeared from your screen, or to save a particular frame which you accidentally skipped past.

To redisplay a frame just press **ALT+F6**. The previous frames will be redisplayed, starting with the frame which just disappeared. The status line will indicate that you are in redisplay mode, and you will have the opportunity to page backwards and forwards over the recent frames, print them, save them to a picture file, etc. When you want to leave redisplay mode just press the **Esc** key.

Odyssey Viewdata Emulation

The Frame Editor

The Odyssey Viewdata module makes it very easy for you to design and send messages in the form of Viewdata frames. The built in Frame Editor is of the most powerful and easy to use Viewdata editors on the market, allowing you to design sophisticated frames with a variety of graphics and colors. At the same time, this section discusses how Viewdata frames are designed and why they are displayed as they are on your screen.

The Upload Mail function, explained elsewhere, lets you include your edited frame as private mail, or for public viewing (like the "Gallery" on the UK Prestel system).

Before planning to include graphics in mail messages however, please check whether your Viewdata service allows it. Some systems do not yet allow graphics in mail messages.

To start the frame editor press **ALT+A**. You will see an editor frame, which is empty except for the bottom line. If you use the frame editor in split screen mode then the right hand side of the display will show the frame editor help screen. If you work in full screen mode then **ALT+Z** shows the help screen, just like normal terminal mode.

Odyssey Viewdata Frame Editor

Editing Text

You can edit your frame on 24 lines - Viewdata services use only the first 24 lines of the screen. The 25th line is used in the frame editor to display information on the frame you are editing, such as the cursor position and the character under the cursor, or the meaning of the code, if it is a special display mode character, and whether or not you are currently in insert mode. The frame editor can hold up to three separate frames at once, and you can move between those frames by pressing the **F10** key.

To make a change to a line, simply start typing characters. If you are in insert mode (**IN** appears on the status line) then characters from the current cursor location onwards are moved right, and the new character is inserted - characters moved outside the right margin are lost - the frame editor does not line wrap. In overtype mode (the default) characters typed replace the character previously under the cursor. While editing you can use the arrow keys and tab key to move around the frame, **PgUp** and **PgDn** take you to the first and lines lines on the frame, **Ctrl+PgUp** and **Ctrl+PgDn** move you to the first or last *character* on the display, **Home** and **End** to the first and last column, and **Enter** moves you to the first column on the next line. You can use **Backspace** and **Delete** to delete characters, and the **Insert** key to toggle insert/overtype mode.

Unless you include special attribute codes before the text, characters typed will always appear white on black, in normal height.

You can delete an entire line. To do so, position the cursor on the line to be deleted and press **ALT+D**. To erase from the cursor position to the end of the line press **Ctrl+End**, and to erase from the cursor to the beginning of the line press **Ctrl+Home**. To erase the entire display simply move the cursor to the first line (PgUp), then hold down **ALT+D** until all lines are erased.

You can insert a new line. To do so, position the cursor where the new line is to appear, and press **ALT+I**. All lines from the current line onwards will be moved down, leaving a blank line at the cursor position. The line moved off the display is lost.

Odyssey Viewdata Frame Editor

Editing Frame Blocks

You can move, copy, save, load, delete and insert frame blocks. This is done by a "cut and paste" method. To use a block you must first *mark* it, and then decide what you want to do with it.

Sizing and Marking a block.

Position the cursor on the first character of the block (the upper left corner), then press **ALT+M**. The screen will be blanked, the status line will display the cursor position and your options. As you move the cursor the block will grow, and data corresponding to that block will appear on the screen. The following keys have an affect while marking a block :

<Arrow keys>	- adjust block size
PgUp/**PgDn**	- change block to maximal/minimal height.
Ctrl+PgUp	- change block to maximal size, i.e. from the upper left corner of the box to the bottom right corner of the screen.
Ctrl+PgDn	- change block to its minimal size, i.e. one character cell at the upper left corner.
Home/**End**	- change block to maximal/minimal width.

The following keys can only be used if you have an enhanced keyboard:

Alt+Home	- Moves the upper left corner of the block to the left margin.
Alt+End	- Moves the lower right corner of the block to the right margin.
Alt+PgUp	- Moves the upper left corner of the block to the top margin.
Alt+PgDn	- Moves the lower right corner of the block to the bottom margin.
Alt+Up	- Moves the entire block one line up.
Alt+Down	- Moves the entire block one line down.
Alt+Left	- Moves the entire block left one column.
Alt+Right	- Moves the entire block right one column.

When you have finished marking your block you have three possible options:

- Press **Esc** to abandon the block operation.
- Press **Delete** to erase the block as marked.
- Press **Enter** to accept the block as marked. This causes the block to be copied to an editor scratchpad.

Pasting a Block.

To paste the block that you have marked, move the cursor to where the upper left corner of the pasted block should appear, and press **ALT+P**. The block will appear at that position, however you can still move it around the screen using the arrow keys. Two types of block paste are possible: *Solid* and *Transparent* paste. The current mode is shown on the status line, and you can toggle between the two using the space bar. *Solid* means that the characters in the block will completely replace the characters previously under the block. *Transparent* means that only non-space characters in the block replace characters under the block; where there are spaces in the block the previous characters will remain. Press **Esc** to abandon the paste operation, or **Enter** to complete it.

Saving the block to disk.

You can make your own library of standard pictures or text, save them on disk, and then load them later to include them in a frame. To save a block to disk you should mark the block, then press **ALT+W** to save it to disk. You will be prompted for a name to give to the file.

Reading a block from disk.

To load a previously saved block press **ALT+R**, and give the name of the block file. The block is read from the file into the *scratchpad* (not the screen). To put the block on the screen you should use the *paste* operation described above.

Odyssey Viewdata Frame Editor

Drawing Boxes

The frame editor makes it very easy for you to draw boxes. The boxes will be drawn using mosiac characters. However, in order to display these mosaic characters correctly, you should remember to first place the proper mosaic attribute correctly on each line where the box will be displayed.

To Draw a Box:

- Place mosaic attributes on all lines where the box will be drawn. You can use the **ALT+C** key combination to type the mosaic attribute once and then copy it to the following lines.

- Position the cursor where you want the top left corner of the box to be.

- Press **ALT+B** to start drawing the box.

A small box will initially be displayed, with the cursor positioned at the bottom right corner. By moving the cursor with the arrow keys you can adjust the size as required. The following table lists the keys which can be used for adjusting box size and position:

Arrow keys	- adjust box size
PgDn/PgUp	- change box to maximal/minimal height.
Ctrl+PgDn	- change box to maximal size, ie. from the upper left corner of the box to the bottom right corner of the screen.
Ctrl+PgUp	- change box to its minimal size, ie. one character cell at the upper left corner.
Home/End	- change block to minimal/maximal width.

The following keys can only be used if you have an enhanced keyboard:

Alt+Home	- Moves the upper left corner of the box to the left margin.
Alt+End	- Moves the lower right corner of the box to the right margin.
Alt+PgUp	- Moves the upper left corner of the box to the top margin.
Alt+PgDn	- Moves the lower right corner of the box to the bottom margin.
Alt+Up	- Moves the entire box one line up.
Alt+Down	- Moves the entire box one line down.
Alt+Left	- Moves the entire box left one column.
Alt+Right	- Moves the entire box right one column.

Once you have completed adjusting the box size and position, you have two choices.

- Press **Enter** to accept the box as it appears.

- Press **Esc** to cancel the box.

Box Types. While drawing the box you can select any of eight different box types, of various line thicknesses and line styles (ie. broken and solid). Press the **space** bar to cycle through the box styles.

Odyssey Viewdata Frame Editor

Copying Characters

If a line on the screen will be similar to the one above it, it is sometimes useful to be able to copy characters from the line above. You can do that by using the character copy feature. Position the cursor on the character you wish to copy and press **ALT+C**. The character under the cursor will be copied to the same column on the next line. ALT+C is ignored if you position the cursor on the last line. Note that ALT+C copies the character only, not its attribute. If you wish to preserve the attribute then you also need to copy the attribute characters which precede the copied character on the line.

Odyssey Viewdata Frame Editor

Special Effects

You can add special effects to a character such as double height, semi-graphics, color etc. If you intend to use the frame for mailing, be aware that some Viewdata services do not allow graphics in mail frames. To change the attributes of a character simply insert an attribute character into the line before the character whose attribute you wish to affect - the help screen shows clearly what keyboard combinations produce which effects.

Odyssey Viewdata Frame Editor

Loading and Saving a Picture

When you have finished editing a picture, you should remember to save it! To do so you should press **ALT+S**. You can load a picture by pressing **ALT+L**. In either case you will be prompted for a file name, usinga standard file selector dialog.

Odyssey Viewdata Frame Editor

Leaving the Frame Editor

To leave the frame editor just press **ALT+X**. A picture loaded into the frame editor is not lost when you return to terminal mode. The picture will be displayed once more when you re-enter the frame editor.

Odyssey Viewdata Emulation

Uploading Mail

When you wish to send mail you can either type the message directly while online, or you can prepare the message offline using the Odyssey Viewdata frame editor, then upload the mail in one go.

Sending a frame to a mailbox is very easy and is done by a simple key combination. Before sending a prepared frame you must first have saved it to disk - you cannot send the frame directly from the frame editor.

To send a frame you should first make sure that you have selected the correct page on the host service, then press **PgUp**. You will be asked for the file name. Odyssey Viewdata supports several mailbox formats which you can switch between using **ALT+M**.

Odyssey Viewdata Emulation
Downloading Files
Odyssey Viewdata supports the CET Telesoftware protocol for file downloads. To download software you must *first tell the host* software that you wish to download software or data. You will not be able to download anything if the host doesn't know it is supposed to be sending something at the time!

First, access the Viewdata system. Once you are in the system you must access the telesoftware area. Then, often after several menus, you must go to the appropriate "page" and indicate the name of the file you wish to download. Sometimes, only specially registered users are allowed to download files.

Once you have selected a file to download, the Viewdata service will tell you to start downloading the file. The strange characters you see at the bottom of the screen are actually the first bytes of the file trying to make their way to your PC - they show that the Viewdata service has started sending the file to you, and is now waiting for you to start downloading, so you had better do so! - just press **PgDn**. A download window will appear on your screen to keep you informed of progress. An alarm will sound when the download is completed.

Odyssey Help

Glossary

ARC	Baud	BBS	BPS
CCITT	Checksum	CPS	CRC
DCT	DIB	Download	DPI
EIA	Escaped	LZH	MNP
OCR	RGB	UART	Upload
V.23	V.42	YcbCr	ZIP

ARC: was originally the format used by the utility of the same name published by *System Enhancement Associates* (SEA). This was the de-facto standard BBS archive format for several years, and so you still find a great many BBS files in this format, even though the ARC format has largely been supplanted by ZIP etc for new uploads. The SEA ARC utility supported a number of compression methods, starting with none (ie. file stored with no compression), then run-length encoding, and then several variations on the Lempel-Ziv-Welsh (LZW) scheme, each later version of the scheme slightly increasing in sophistication (eg. moving from fixed length codes to variable length codes). Phil Katz of PKWare also produced an ARC compatible utility called PKARC, and also added a new "Squashed" compression method - another LZW variant. The Odyssey ARC format reader handles all ARC and PKARC compression methods.

BAUD: One of the properties of a serial connection is the data rate, meaning the speed, in bits per second, at which data is transmitted. This data rate is commonly referred to as the "**baud rate**", a term which is convenient, but technically inaccurate. However, to avoid confusion, we adopt that misnomer throughout this help system. When we say that data is transferred at "*2400 baud*", we really mean that data is transferred at 2400 bits per second (bps).

BBS: Short for **B**ulletin **B**oard **S**ystem. An electronic meeting place for modem users, a BBS is generally a small system run by an individual, with basic facilities for public and private text messages, and file transfer.

BPS: **B**its **P**er **S**econd. A unit of measurement used when recording how quickly data is being transferred, eg. by a modem or serial link. The word *baud* is usually, and incorrectly, used as a synonym for **bps**.

CCITT: **C**onsultative **C**ommittee on **I**nternational **T**elegraphy and **T**elephony. A body responsible for the development and promotion of standards related to the communications field. Recently absorbed into the **ITU**.

A **checksum** is used when a device has data to send, and wants to ensure that it reaches the destination undamaged. It does this by forming the data into packets, and then adding a checksum to each packet, where the checksum is simply the sum of the values of all the data bytes in the packet. The receiver recalculates the sum and compares the result to the checksum received with the data. If the two values do not match then the data packet must have been corrupted. A checksum is simple to carry out, but considerable less reliable than CRC checking.

CPS: **C**haracters **P**er **S**econd. A unit of measurement used when recording how quickly data is being transferred, eg. using a file transfer protocol.

CRC is an acronym for *Cyclic Reduncancy Check*. It is similar to a checksum in that it is used to check the integrity of a block of data. Roughly speaking, it involves forming all the data in the block into one huge number, and dividing that huge number by a smaller value (the CRC polynomial). The remainder after the division is the CRC. Typically a sending process (such as a file transfer protocol) will calculate the CRC on a block of data, and then send the CRC along with the data. The receiving process recalculates the CRC on the block of data it has received, and if the receivers calculated CRC matches the one received with the data packet then the chances are very good that the packet contains no transmission errors (with a sixteen bit CRC that means less than one chance in 65000 of arriving at the right CRC by accident).

DCT is an acronym for Discrete Cosine Transform, a real-function relative of the Fourier Transform. The DCT is primarly used in image compression (such as in JPEG). In fact, the DCT does not compress an image directly, but instead transforms the image data into a format which is easier to compress in a lossy fashion. Eg. the DCT converts pixel data into a "rate of change" function, which is then quantized (scaled down by a constant, losing a little precision). This results in many frequency components having a zero coefficient, which lends itself well to compression using run length or Huffman encoding.

DIB: Device **I**ndependant **B**itmap, a portable bitmap format invented by Microsoft. When saved to disk a descriptive header is attached and the result is a Windows **BMP** file. There are also run length encoded variants of this file format, called **RLE** files.

DPI: Dots Per Inch, a measurement of image resolution.

EIA: **E**lectronic **I**ndustries **A**ssociation. Yet another of the plethora of standards bodies involved in devising standards for various aspects of communications. Our main interest in the EIA is that it is they who were originally responsible for the FAX-modem interface standards, Class I and II.

Escaped: A file transfer protocol sometimes needs to "escape" certain characters which occur in the file, which means to replace the special characters with a code which is harmless. For example, suppose you were transferring data to a host which treated ^Z as an instruction to disconnect the line - you obviously would not want that character to occur in the file data! The solution is normally to replace any occurrences of ^Z in the data with something like <character>Z, where <character> is called the "escape code" (ASCII 16, or DLE is often used for this). The receiver sees the escape code and so knows to replace whatever character follows with the control code equivalent, so Z becomes ^Z. Of course, this makes DLE itself a sensitive character, so it also must be escaped.

LZH: an archiving format used by the LHARC (later called LHA) utility, LZH has attained a respectable following among BBS users. Initially this was because, while LHARC was significantly slower at compression than PKZIP a) decompression wasn't *much* slower, b) compression ratios were higher, c) LHARC was free, d) the source was also available for free, which made it easy to port the format to non-DOS platforms. LHARC also had the ability to create self extracting archives with surprisingly little file size overhead. Subsequent PKZIP versions pretty much matched LHA on compression ratio, though the other attractions of LHA remained.

MNP: Microcom Network Protocol. This was the de-facto standard error correction protocol used in modems prior to the introduction of V.42bis. Several MNP flavours (or classes) are in common use, the most important being **MNP5**, which includes data compression. Odyssey supports MNP5 in software.

OCR: **O**ptical **C**haracter **R**ecognition. An algorithm or software package which is used to recognise and extract text from a document stored in image form - eg. a FAX, or an article scanned from a magazine. This kind of pattern recognition is incredibly difficult for a computer to do, and current OCR packages are still quite primitive - their output usually needs to be hand edited by a human being to bring it up to acceptable accuracy. The difference between a good and a bad OCR is how much editing is left to the user.

RGB (Red-Green-Blue) is the most popular system for specifying color on a computer graphics display. RGB controls color by describing the intensities of the three primaries which should be added to produce the desired color. In theory each primary would range between 0 and 1, with 0,0,0 specifying black, and 1,1,1 specifying white. 0.5,0.5,0.5 would be a mid-intensity gray, 1,0,0 would be a high intensity pure red, etc. In actual fact, most PC implementations scale each component into an integer range, such as the 0-255 range used in Microsoft Windows.

UART: *Universal Asynchronous Receiver/Transmitter*. The UART is the chip inside your personal computer which is responsible for sending and receiving bytes via your serial ports. For example, when sending data your computers CPU passes a byte to the UART, which then transfers the bits in that byte one at a time out through the serial port transmit wire. The UART used in the original PC was the National Semiconductor **8250**, which could handle speeds up to 9600, provided your PC wasn't doing too much else. Later PCs and ATs used the **16450** UART, which was somewhat faster, and could handle speeds of 19200 or 38400, again depending on how busy your PC was. The best equipped PCs now use the **16550AFN**, which can handle much higher speeds (up to 115200 bps in many cases), which it manages by providing extra on-chip buffering.

UPLOAD/DOWNLOAD: are standard jargon terms for, respectively, sending and receiving data (file transfer). The jargon words are instead instead of words such as *send* because the latter is ambiguous, since the direction of the transfer depends on whose point of view is implied, ie. does *send* mean "send from the host", or "send to the host"?   The Upload/Download terms are always used from your point of view (the local machine), ie. Upload always means "send from the local machine to the remote host". This point of view is maintained regardless of whether it is the host machine or Odyssey which is offering you the chance to upload (or download).

V.23: A CCITT standard for modems which transmit at 1200 bps in one direction, and 75 bps in the other, used mostly by European telephone authorities for providing various database services. Designed when 300 bps was the most common speed for personal-use modems, it provided a cheap way to allow callers to read data at 1200 bps, without requiring a much greater bandwidth than a 300 bps modem did. V.23 was already obsolete when it was introduced, but you can still find services which require it. V.23 never took off in the US, which at that time was still using Bell standards for 300 and 1200 bps modems.

V.42 is the ITU (formerly called CCITT) standard for error correction. An important extension to V.42 is ***V.42bis***, which provides data compression superior in performance to that found in MNP5 (though it never gets the 4:1 performance typically claimed by modem manufacturers).

YCbCr is an system for specifying color, an alternative to the RGB model. YCbCr is quite similar to system like YUV, used in color television. The idea is to specify the color in terms of Y - the intensity or grayscale value - plus the blue and red chrominance values. In television this has the advantage that a black and white TV receiver simply ignores the Cb and Cr components, while a color set uses all three components. If RGB was used by color television then a separate grayscale channel would need to be transmitted for the benefit of black and white sets.

Also, the human eye is very much more sensitive to the intensity (Y) component than it is to the color components. This is taken advantage of in certain computer graphics file formats, such as **JPEG/JFIF**, which usually stores the color chrominance components using a lower precision (fewer bits), thus saving space.

ZIP: this is the current de-facto standard BBS archive format. ZIP improves on ARC/PKARC by providing faster decompression, better compression ratios, and the ability to store complete directory trees in the archive, instead of the simple list of unqualified file names used by ARC. Although ZIP supports a number of different compression methods, the most important algorithms are derived from the "Sliding Dictionary" Lempel-Ziv schemes (ie. where a string which has previously occurred in the file is replaced by a code telling the reader the position and length of the previous occurrence). ZIP also uses a secondary static Huffman encoding to further compress the character, length and position codes. Later ZIP algorithms find efficient ways to handle large (32k) dictionary sizes.

Odyssey Script Language

Script Tutorial

You may read the Odyssey Script Tutorial by selecting from the list of topics below, or you can select the first topic, and then use the WinHelp "browse buttons" to step between topics.

Odyssey Script Tutorial

Introduction to Syntax

Odyssey Script in most respects resembles a traditional computer programming language. As such, and in common with most languages, it is formed from words, and has a specific grammar whose rules much be followed in order to compose a correct sentence. Unlike most spoken languages however, the grammar of a programming language is normally much simpler and more rigidly defined. This is necessary, since technology has not yet advanced to the stage where it is possible to specify computer programs using human language (and it isn't entirely clear that it would even be desirable).

When describing a human language, it is normal to refer to words, sentences and grammar. In computer languages those same features are normally called *symbols*, *statements*, and *syntax*.

When you tell Odyssey to run a script, it first loads the text file containing that script into memory, and then it checks the script for errors. In order to do this it scans the file, extracting symbols one at a time, forming those symbols into a sentence (statement),and then validates the newly formed statement using its knowledge of the rules of grammar (syntax). If the statement correctly conforms to the syntax rules then the script checker will continue with the next statement, otherwise an error message describing the nature of the error will be issued and the checking will stop.

The script language is designed to make it as easy as possible for the syntax checker to extract symbols from the source file. For example, a symbol is never made up of more than one word, unless it is surrounded by quotes. Also, a symbol never starts with a digit unless the entire symbol is a number. A symbol can contain digits and not be a number, but that symbol cannot start with a digit. In most cases, symbols are separated from other symbols by punctuation marks, spaces or line ends. In some languages a newline is used to terminate statements, but in Odyssey script language a semicolon is used for that purpose, while a newline is treated like a space. This gives you the freedom to format sentences as you like, without reference to the number of lines the sentence occupies - you can place several statements on one line, or spread one statement over several lines. The number of spaces between symbols is not important, provided that (where necessary) there is at least one.

Where necessary means, where there would otherwise be ambiguity. If a script contained HELLOTHERE, this would be seen as a single symbol. If you wanted it to treated as two then you would have to insert a space. On the other hand HELLO,THERE would be treated as three symbols - <HELLO> <COMMA> <THERE>, since names cannot contain commas. There is no ambiguity here, so spaces are not required, although you can still insert them if you wish.

The process of comparing the symbols of a program text against a defined syntax is called **PARSING**. After it has been checked by the parser, Odyssey converts the script into a coded form, which is much easier and faster for it to manipulate during execution. The entire process of starting from the human readable text, parsing, and translating into a machine readable executable form is called **COMPILING**. In Odyssey, scripts are normally compiled each time they are loaded. However with large scripts this might be considered wasteful, and so you are provided with the **Command|Compile script...** menu option, which is able to compile a script and store it on disk in precompiled form. Odyssey can run such scripts immediately after loading them, and does not need to compile them again. You can also compile scripts you have loaded into an Odyssey text editor by pressing **F9** when the editor window is active.

It was mentioned in a previous paragraph that the parser breaks the input text file into symbols. The basic symbol types that the script language knows about are **words**, **strings**, **numbers** and punctuation characters. These are described in the following following topics (use the browse buttons above to step between topics).

Odyssey Script Tutorial

Words

WORDs are script language symbols which start with a letter, and then contain a mixture of letters, digits, or the underscore character. For example:-

```
HELLO    ABC123      HELLO_AGAIN
```

TODAYS DATE cannot be treated as a single symbol because it uses an illegal character, a space. The syntax checker would in fact see this as two separate symbols. Words are used in the script language to describe operations, and also to name variables and other objects. Some words are known to the parser before checking starts, while others are defined within the script. The script language does not differentiate words by case, ie. HELLO, Hello and hello are all regarded as identical, and refer to the same object. The script language treats all characters in a word as significant (this differs from previous Odyssey versions, in which only the first ten characters of different symbols needed to be unique).

Odyssey Script Tutorial

Strings

A string is a series of characters which begins and ends in either single or double quotes. The string must use the same type of quote at both ends. This string is then treated as a single entity, even when it includes spaces. These are all examples of valid strings:-

```
   "Hello, World"
   'Hello, World'
   'The boss said "hello" to me today'
   "I prefer Mike's car to my own"
```

These are not considered to be valid string symbols:-

```
   "The boss said "hello" to me today"
   'I prefer Mike's car to my own'
   "Hello,
   World"
```

The first two strings are incorrect because they contain quote marks embedded in the string which are the same as those used to mark beginning and end of the string. When the checker examines the first string it would take "**The boss said** ", **hello**, and " **to me today**" to be separate symbols. A string can contain double quotes only if it is itself surrounded by single quotes, and vice versa. The third string is incorrect because it is on two lines. This is a language limitation - the entire string must be on one line.

Odyssey Script Tutorial

Numbers

A number is a sequence of one or more digits, forming a decimal or (ie. base ten) or hexadecimal (ie. base sixteen) value. For example, these are some decimal numbers:-

```
123
0123
1
```

The script language allows the use of hexadecimal numbers if you prefix the number with a dollar sign. The letters 'A' to 'F' are then used to indicate single digit values between ten decimal and fifteen decimal. For example:-

```
$123
$0123
$AB23
$23ab
```

are all valid hexadecimal numbers. Note: hexadecimal is often written in abbreviated form as "**hex**".

Odyssey Script Tutorial

Punctuation Characters

These are characters such as brackets, semicolon, colon etc., and are used to separate or group symbols, or as arithmetic operators. There are a large number of punctuation characters, and these will be described in later sections of this chapter.

Odyssey Script Tutorial

Comments

The script language allows the insertion of comments into the program text. When the parser finds the start of a comment it simply skips characters until it finds the end, and then extracts the symbol (if any) which follows. The entire comment is treated as if it were a single space. The language supports three comment styles, the use of which is entirely a matter of preference. Here are examples:-

```
SCRIPT myscript;
BEGIN
   Dial("MICROP");          -- start dialing
   IF WaitFor("User name?",10) THEN
      (* ok, we appear to be logged on *)
      Transmit("John Smith|"); -- send my name
      WaitFor("Password:");
      Transmit("smithy|"); -- send my password
   END;
END /* of script */;
```

The above script uses all three comment styles, as described below:-

Anything on a line following a "**--**" is a comment, and is ignored. The comment is terminated by the end of line. This is the only example in the Odyssey script language in which the end of a line affects syntax.

Pascal style comments start with **(*** and end with ***)**. These symbols, plus everything in between, are discarded. This style of comment can be spread over multiple lines.

C style comments use the symbols **/*** and ***/**, but are otherwise identical to Pascal style comments.

The script language allows comments to be nested, so it is possible to comment out a section of script which itself contains comments, without the parser becoming confused.

Beware of opening comments without closing them, as this will cause the parser to skip the remainder of the file. If Odyssey gives you an "Unexpected End of File" error when parsing a script then it is almost certainly due to an unclosed comment.

Unlike interpreted languages such as Basic, there is no penalty for inserting comments into an Odyssey script, since comments do *not* increase the size of the compiled program and thus reduce the memory available for executable code. Comments have no effect one way or the other on the code generated by the script compiler. However, do not feel that this means that you should supply great volumes of comments. Over commenting is as much a sin as under commenting.

Odyssey Script Tutorial

Script Syntax Definitions

The remaining topics of this help chapter will go into detail about each important aspect of the script language. In order to focus the discussion a simple skeleton structure will often be used to explain the syntax accepted by the compiler for a particular statement type. Where a symbol is intended to be literal (ie. will appear exactly as shown in your script) then it will be shown in bold upper case, otherwise a non-literal symbol will be lower case and enclosed in angle brackets, to show that this will be expanded into literal form by the programmer. For example, here is a definition of a script "while" looping statement:-

```
WHILE <logical expression> DO
   <statement sequence>
END ;
```

In this example the WHILE, DO and END symbols should literally appear as shown in your script, whereas <logical expression> and <statement sequence> stand for any suitable operation matching that symbol. The meaning of those symbols is discussed elsewhere in this chapter.

Odyssey Script Tutorial

Keywords and Naming

The section Introduction to Syntax described how the script parser recognises symbols in four basic classes - words, numbers, strings and punctuation marks. However, the parser really does more than that, because it differentiates between words which belong to the language, and those which belong to the person writing the script, for example, when used to name variables.

Words which are belong to the language are called **KEYWORDS** and are reserved, meaning that you cannot reuse such a word as a name for another object. Some programming languages do not employ the concept of reserved words, and instead interpret the word differently depending upon the context in which it is used. This may sound rather convenient, but in fact it simply leads to silly tricks which make life very confusing for programmer and parser alike, as the following example will show:-

```
IF THEN<ELSE THEN
    IF:=THEN
  ELSE IF THEN>ELSE THEN
    IF:=ELSE
  ELSE
    IF:=END
END
```

This sort of trick is not only silly, the fact that it is allowed makes it very hard for the parser to detect errors quickly, or to accurately detect the nature of an error. The use of reserved keywords avoids this problem entirely, and makes it possible for the parser to detect an error at the first incorrect symbol, for example:-

```
IF a< THEN
    a := 2
  ELSE
    a := 1
END;
```

The above statement has an error, in that the expression is incomplete (the term which should follow the less-than symbol '<' is missing). The Odyssey script parser will spot this error as soon as it sees the "THEN" symbol, because it knows that THEN is a keyword which cannot possibly be a term in an expression.

The following table lists all of the reserved words in Odyssey script language.

AND	**ELSIF**	**IF**	**PROC**	**THEN**
BEGIN	**END**	**NOT**	**REPEAT**	**UNTIL**
CASE	**FILE**	**LONG**	**NUMBER**	**RETURN**
VAR	**DO**	**FLAG**	**OF**	**SCRIPT**
WHILE	**ELSE**	**FUNC**	**OR**	**STRING**

In addition, the names of built in commands, and certain symbols used in conjunction with those commands (such as the names of file transfer protocols, parity types etc) may not be used to name your own objects. These predefined names can be found along with the description of the command which uses them. The script parser *will* point out any attempt to reuse a name.

Odyssey Script Tutorial

Overall Script Structure

Every text conforming to Odyssey Script Language syntax has an overall structure as follows:-

```
SCRIPT <name>;

<Optional: Variable and Procedure Declarations>

BEGIN
   <statement sequence>
END;
```

So, following the above definition, here is an example of the smallest script that will be accepted by Odyssey.

```
script smallest;
begin
end;
```

Naturally, since there are no commands, this script does nothing except exit immediately. Notice that we used lower case for the keywords in this example. This highlights a point mentioned earlier, which is that use of upper or lower case is entirely a matter of preference. Having made that point we will from now on return to using upper case, because that does make the use of keywords more obvious.

Now suppose we wanted to make the script display "Hello, World!" on the screen. Commands go between the BEGIN and END symbols, and so the necessary script would be:-

```
SCRIPT hello;
BEGIN
   Write("Hello, World!");
END;
```

also, if we wanted to declare a variable, you can see by the definition that it should go before the begin, like so:-

```
SCRIPT hello2;

VAR my_variable : Number;

BEGIN
   Write("Hello, World!");
END;
```

Finally, if we want to define a procedure, then it also goes before the main BEGIN, as in:-

```
SCRIPT hello3;

VAR my_variable : Number;

/*...........................................*/

PROC My_Procedure();
BEGIN
   Write("Hello, World!");
END;
```

```
/*.........................................*/

BEGIN
   My_Procedure();
END;
```

The /*... lines are simply comments, inserted to make the user defined procedure stand out a little better. The main BEGIN - END body of the script now makes a call to our procedure, which just displays "Hello, World!" as before.

By now you will be panicing, because I have suddenly introduced variables and user procedures. Don't worry, you are not intended to understand that now, the intent here is simply to let you see what a script looks like as it grows. "Write" is a built in command which you may have met in the Creating your first Script by hand section - this and all the other commands are documented later in the chapter.

The last example declared a variable and a procedure, yet the definition at the beginning of the section did not mention the order that they should be in - do variables have to come before procedures? The answer is no, you can mix procedure and variable definitions in any order you like, provided that they all come before the main BEGIN in the script, and provided that they are not used before they are defined. Most scripts in this help chapter will however group variables at the beginning of the script, to make them easier to find.

Odyssey Script Tutorial

Types and Variables

In previous sections you saw how the script checker recognises symbols as either words, numbers, strings or punctuation. Later on you were told that certain words are treated specially, in that they are reserved by the language for use as keywords. Now it is time to learn something else about words, which is that the checker also associates, with each word, a particular TYPE (expressions and functions also have types, but we will return to that later). These are the different word types that the checker recognises:-

KEY WORDS:	The word is a reserved word used by the language (you have already met these).
NUMBER:	The word is the name of a number variable with a range -32768 to 32767 (a sixteen bit signed integer).
LONG:	The word is the name of a number variable with a range of 0 to 4294967295 (a thirty-two bit unsigned integer).
FILE:	The word is the name of a file variable
FLAG:	The word is the name of a true/false logic variable.
STRING:	The word is the name of a string variable.
PROC:	The word is the name of a built in or user defined procedure.
FUNC:	The word is the name of a built in or user defined function. Functions also have a secondary type of Number, File, Flag or String which allows the word to be used as if it was a variable.

We will return to the subject of procedures and functions later. For now we will look at the definition of variables.

The checker knows the type of a variable because you tell it what the type is when you define the variable. Here is the format of a variable definition:-

```
VAR <list of names> : <type> ;
    <optional: another list of names> : <type> ;
```

Where <type> can be either **NUMBER**, **LONG**, **FILE**, **FLAG** or **STRING.** The optional line can be repeated as many times as you like. For example, the following lines define two number variables called A and B, two string variables C and D, two file variables E and F, and two long number variables G and H :-

```
VAR a,b : Number;
    c,d : String;
    e,f : File;
    g,h : Long;
```

You declare variables because you want to be able to store and then manipulate data for some part of the lifetime of a script (ie. while it is running). The variables are given types so that the checker can spot when you try to do something silly, like adding a file variable to a number variable. The following is a short script which takes two numbers and adds them together, then displays the result:-

```
SCRIPT Add;

VAR a,b,c : Number;

BEGIN
    a := 1234;
    b := 4321;

    c := a + b;
    Write("A plus B equals - ");
    Write(c);
END;
```

The above script was written for clarity, not necessarily because that was the best way to do it. In fact, there is no reason to have the variable C, since we don't need to store the sum, only display it. Also, we don't need two separate Write() commands, because a single **Write()** command can display more than one item if we separate the items with a comma, so here is a second attempt at that script:-

```
SCRIPT Add;

VAR a,b : Number;

BEGIN
    a := 1234;
    b := 4321;

    Write("A plus B equals - ", a+b, "|");
END;
```

Note that we have added a "|" symbol in the list of items for the Write command. This tells the script processor to move the cursor to the next line after it has displayed the first two items.

We can do a similar thing with string variables, that is, take two string variables, add them together and display the result:-

```
SCRIPT String_Add;

VAR c,d : String;

BEGIN
    c := "Hello ";
    d := "World!";

    Write(c+d, "|");
END;
```

If you run this script it will display "Hello World!" on the screen. The script language allows you to add numbers and strings (addition of strings is called "concatenation"), but there is no sensible way to add FILE and FLAG variables. You use these types of variables for other things - File variables are used when you want to access a DOS text file on disk, and a command is provided which allows you to associate a file variable with a physical file:-

```
SCRIPT Open_Close;

(* open a file and then close it again *)

VAR f : File;

BEGIN
    Fopen(f, "TESTFILE.TXT");
    Fclose(f);
END;
```

The file variable is simply used as a "handle", i.e. a convenient way to refer to an actual file, without having to type the file name every time.

Flag variables are used to record TRUE/FALSE values, usually the result of an earlier test your script did, and they are also used in loops. You will meet a lot of these later.

Odyssey Script Tutorial

Note on String Variables

One problem with defining a lot of string variables is that strings take up a lot of memory, and memory is not an unlimited resource for a script. The amount of memory occupied by a string therefore has to be limited in some way. In the Odyssey script language strings are normally allocated enough memory to store eighty characters. Note that this memory is allocated to the string when it is defined, before any value is assigned to it, and therefore means in some cases a waste of memory, because you may not need to store that many characters in that particular string variable.

On the other hand, sometimes eighty characters is not enough. For example, when you are writing a script to read lines from a text file and paste those lines to a BBS, you naturally want the script to be able to handle lines of any reasonable length, and some may quite easily be longer than the default eighty character maximum length of a string variable.

The way to get round this is to use a **STRING LENGTH OVERRIDE** in order to give the compiler a "hint" as to how much memory it should allocate to a particular string variable. You do this by including the size you need in square brackets after the string definition. The exact syntax for
a string variable declaration is :-

<var_list> **: STRING ;**

or :-

<var_list> **: STRING [** <integer constant expression> **] ;**

For example:-

```
VAR s1,s2 : String[20];
    s3 : String;
    s4 : String[255];
```

In the above example, strings "s1" and "s2" are each allocated enough memory to hold a twenty character string, so here we are saving memory by allocating less than the default. String "s3" did not use an override, and so it gets the default memory allocation, enough for an eighty character string. String "s4" needs more memory than the default, and so the override is used to allocate enough for a 255 character string. Judicious use of string overrides should allow you to make more efficient use of memory by reducing allocated space when you don't need it, and allocating more in specific instances when you do.

If you try to assign a long string to a shorter one, then the string will be truncated. For example:-

```
VAR s1 : String[10];
    s2 : String[20];

BEGIN
    s1 := "Hello, World!!"; -- string truncated

    s2 := "Hello, World!!";
    s1 := s2;               -- string truncated
    ....
```

If you count the characters between the quotes in the above examples you will see that there are fourteen characters. Yet string "s1" has been declared with a maximum length of ten characters, so it obviously cannot hold the whole string. This will not cause an error, it simply means that only the first ten characters will be assigned. In the second attempt, the string literal was initially assigned to a longer string variable, and then an attempt was made to assign it to the shorter variable again. This makes no difference. The string is still too long and will be truncated at ten characters. Notice that the fact that "s2" is declared as being longer than "s1" is not the cause of this problem. It is simply that the actual string being assigned is in this case too long for the destination variable.

Odyssey will not allow strings larger than 255 characters, even if you try to allocate more using a string override.

Odyssey Script Tutorial

Expressions

Expressions are the driving force in any script, and as such is a complex subject which requires careful explanation, and just as careful reading. Understanding of expressions is vital in order to progress to a fuller understanding of programming using the Odyssey Script Language.

An expression defines a calculation for the script processor to evaluate. A calculation need not be numeric - string and logical operations are also supported. The script processor evaluates the expression and produces two items of information, a "**value**" which is the result of the evaluation, and the "**type**" of that result.

For example; if you add two numbers, then the value is the resulting sum, and the type is a number, just as the terms were. Here are some sample expressions, with the evaluated result and type shown for each case:-

Expression	**Expression Result**	**Expression Type**
`2 + 2`	`4`	`Number`
`"Hello "+"World"`	`"Hello World"`	`String`
`TRUE OR FALSE`	`TRUE`	`Logical`
`2 = 4`	`FALSE`	`Logical`
`2 < 4`	`TRUE`	`Logical`
`NOT (2 < 4)`	`FALSE`	`Logical`
`"ABC" < "ABD"`	`TRUE`	`Logical`
`2 AND 2`	`2`	`Number`
`1 OR 2`	`3`	`Number`
`2`	`2`	`Number`
`Long(2)`	`2`	`Long`
`1234567`	`1234567`	`Long`
`Number(1234567)`	`-10617`	`Number`

Expressions only really become interesting when you use variable names instead of constants, and when you use the expression for some useful purpose, perhaps by storing the result in another variable:-

```
VAR a,b,c : Number;

BEGIN
   a := 2;
   b := 2;
   c := a+b;
```

This above sequence demonstrates several uses of an **ASSIGNMENT STATEMENT**, meaning that a value is assigned to a variable. Assignment statements have the following definition:-

<variable name> **:=** <expression> **;**

When the script processor sees an assignment statement it evaluates the expression, confirms that the type of the result matches the type of the variable, and assigns the value of the expression to the variable if so.

We now need to know more about what an expression is. Here is the definition. An expression is either:-

<term>

or: <term> <operator> <term>

or: **NOT** <term>

In this discussion of expressions, the word "term" is used to mean the same as the word "operand", with which some of you may be more familiar. Operand simply means an object which is the subject of an operation, in this case an arithmetical or logical operation. The word "term" is used because it is slightly less distracting than the more obscure "operand".

In the three examples of "assignments" given above, the first two showed examples of the simplest form of expression, consisting of a single term, with no operation, ie. :-

```
a := 2 ;
b := 2 ;
```

note that in both these cases, the expression part is the "2". The remainder of the line forms the syntax of an assignment. The third assignment gave a sample of the second definition of an expression, using an operator and second term:-

```
c := a + b;
```

So in this example the terms are "a" and "b" and the operator was "+", the addition operator. Addition is not the only operation allowed in the script language, the following paragraphs describe them all.

---

a + b (Addition)

Add A and B. Terms A and B must be either both be numbers or both be string types. Addition of strings produces a concatenation.

---

a - b (Subtraction)

Subtract B from A. A and B must be numeric types.

---

a * b (Multiplication)

Multiply A and B, both of which must be numeric.

---

a / b (Division)

Divide A by B. Both terms must have numeric types.

---

a % b (Modulus)

The remainder after dividing A by B. Both terms must have numeric types.

---

a OR b (Logical OR)

IF A or B is true, then the expression is true, else the expression is false. A and B must be logical (flag) types.

---

a OR b (Bitwise OR)

The bitwise OR of A and B, which are both numeric. A bitwise OR produces a result in which bits which are 1 in either A or B are also 1 in the result.

---

a AND b (Logical AND)

IF A and B are both true, then the expression is also true, else it is false. A and B must both be logical (flag) types.

a AND b (Bitwise AND)

The bitwise AND of A and B, which are both numeric. A bitwise AND produces a result in which the 1 bits in A which are common to B are set to 1 in the result.

a = b (Equality)

If A is equal to B then the expression is true, else it is false. A and B can be of any type, but both must be of the same type.

a <> b (Inequality)

If A is not equal to B then the expression is true, else it is false. A and B can be of any type, but both must be of the same type.

a < b (Less than)

If A is less than B then the expression is true, else it is false. A and B must be both numeric or both string types.

a <= b (Less than or Equal)

If A is less than or equal to B then the expression is true, else it is false. A and B must have numeric or string types.

a > b (Greater than)

If A is greater than B then the expression is true, else it is false. A and B must be numeric or string types.

a >= b (Greater than or Equal)

If A is greater than or equal to B, then the expression is true, else it is false. A and B must be numeric or string types.

NOT a (Logical Inverse)

A must be of a logical type. If A is TRUE then the expression is false, else it is true.

NOT a (Bitwise Inverse)

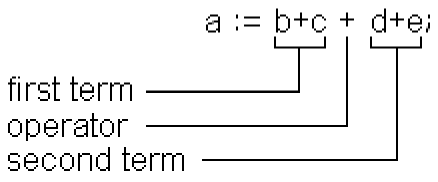
A must be of a numeric type. The value of each bit in A is inverted in the result.

☞ The important point to note about any expression is that the terms must be compatible, ie. they must be of the same type.

Odyssey Script Tutorial

More Complex Expressions

You may be wondering how you handle more complex expressions, that is, with more terms. Well, you will see how that is handled when you discover that a term in an expression may, if you wish, also be an expression in its own right. And so you can have compound expressions such as:-



Notice how the overall expression fits the definition, and how each term, which is also in fact an expression in its own right, also fits. Here are further examples of compound expressions:-

```
a := b*c + d*e;

a := b + c * d;

x := (a OR b) AND (c OR d)
```

and this is not as far as it can go. Each of the terms in a compound expression can also be an expression, and so on ad infinitum. A function call can also be a term in an expression, but we will return to that later, once functions have been properly discussed.

Odyssey Script Tutorial

Operator Precedence

Take a look at the following two expressions:-

```
x := 2 * 4 + 6 ;

y := 6 + 2 * 4 ;
```

Your puzzle for today - what are the values of x and y? Easy, you might say, and give the correct answer, which is 14 in both cases. If you got 20 for x, or 32 for y, then you have forgotten a lesson from mathematics class at school, which is that some operations always take precedence over others, regardless of where they appear in the expression. In the above examples, you must always perform the multiplication first, since multiplication has higher precedence than addition. Precedence rules are a natural consequence (and complication) of the Western convention of infix expression notations (postfix or "reverse Polish" notations do not require precedence rules).

You can override operator precedence by using parentheses, as follows:-

```
y := (6 + 2) * 4;
```

Now y does indeed equal 32, because subexpressions in parentheses have a higher precedence than multiplication. The following is a description of the different orders of precedence in the Odyssey script language. Level 1 has the highest priority, while level 5 has the least:-

1. Subexpressions in parenthesis.
2. The NOT operator.
3. Multiplication, division, modulus, and (*,/,%,AND).
4. Addition, subtraction, or (+, -, OR).
5. Relational operators (=, <=, <> etc).

Now look at the following expression:-

```
a := 6 + 2 - 4;
```

According to the precedence rules given above, addition and subtraction have equal priority, so we could choose to evaluate either subexpresson first (6+2 or 2-4). In mathematics this is not a problem, because the answer is the same (4), regardless of the order in which we do the calculation. In computer languages however, some or all of the terms may be function calls, e.g.:-

```
a := b + c - FuncCall(4);
```

Now suppose that the function "FuncCall" has a **SIDE EFFECT**, which is that it modifies the value of "b". Now it becomes *very* important to know in what order the expression is evaluated, since if "c - FuncCall" is evaluated first, b may have a different value than if "b + c" were evaluated first.

Odyssey script language makes an arbitrary decision to cope with this problem, which is that subexpressions of equal priority are evaluated from left to right, in other words in the previous example, "b+c" would be evaluated first.

Odyssey Script Tutorial

Long data type

Sharp eyes may have noticed a data type of **Long** being used in examples given previously. This is a 32 bit *unsigned* long integer quantity. Longs and numbers (16 bit signed integers) may be mixed in an expression, in which case the integers will be promoted to longs in order to evaluate the expression. One must be careful however to understand exactly when an implicit type promotion will occur. For example :-

```
VAR  a:Number;
     b,c:Long;
Begin
     c := a*b;
```

Multiplying a number (a) by a long (b) is fine, because a is automatically promoted to type long. The following however would be wrong :-

```
VAR  a:Number;
     b,c:Long;
Begin
     a := 1234;
     b := a * a;

     c := 1234 * 2345;
```

The values assigned to 'b' and 'c' above are wrong in both cases because the expression on the right hand side is completely evaluated before the *result* is promoted to a long and assigned to the long variable on the left. In both cases the result (prior to promotion) is an integer, and in both cases that result overflowed (eg. 1234 * 2345 = 2893730 - far too big a number for a 16 bit signed integer result to hold).

This one is slightly more subtle :-

```
VAR  a:Number;
     b,c:Long;
Begin
     a := 1234;
     b := 3456;

     c := b + a * a;   /* note this line */
```

You might think that since the expression on the right hand side contains a long that the result would be promoted to long (correct), and therefore it will not overflow (wrong!). The problem is the order of evaluation within an expression - multiplications have a higher priority than addition, and in the case of the expression shown above, this means that (a*a) is evaluated first, and since a is an integer, the sub-expression result is also an integer, and thus overflows. The remainder of this overflow is promoted to long and added to 'b', and the final result is assigned to 'c'. That result will of course be completely wrong.

In case of doubt, you can avoid these type promotion problems by using explicit "type conversion" to promote a number to a long. The following examples all produce the correct result :-

```
VAR  a:Number;
     b,c:Long;
Begin
     a := 1234;
     b := Long(a) * a;
     c := b + Long(a) * a;
```

You could have explicity promoted every occurrence of 'a', but that isn't really necessary, since in an expression like "Long(a) * a" the second a is implicitly promoted in order to perform the multiplication.

You can also demote a long using the type conversion function 'Number', eg:-

```
    a := Number(c);
```

which yields the lower sixteen bits of the long variable c.

☞ A literal integer constant (such as the '1234' term used in the examples above) is always assumed to be of type 'Number', if it is small enough to fit in sixteen bits. You must explicitly promote it to long (eg. using Long(1234) if you want to override this behaviour).

Odyssey Script Tutorial

Decision Statements

The basics of Odyssey script language have now been dealt with, and it is time to start learning how to put it all together to form complete scripts. One of the things you need to learn is how to control the flow of execution in your script, that is, how to control when a given sequence of commands is executed, and when it is not.

One way to handle flow of control is by means of decision statements, of which the simplest is the **IF STATEMENT**. A basic IF statement has the following form:-

```
IF <logical expression> THEN
   <statement sequence>
END;
```

If you read the previous section on expressions then you will know that a **<logical expression>** means an expression which evaluates to either **TRUE** or **FALSE**. If the expression evaluates to TRUE then the <statement sequence> is executed, otherwise it is not. The purpose of the END is to mark the end of the sequence of statements which is dependent on the result of the test. If the above expression evaluated to FALSE then the flow of control would move directly to the statement following the END, without executing any of the commands in <statement sequence>. The following is an example of the use of the above form of IF statement:-

```
IF Dial("MICROP") THEN
   WaitFor("User name?");
   Transmit("John Smith|");
END;
```

which is the same example you may have discovered in the section on "Creating your first Script by hand". The thing that was not mentioned in the earlier section is that Dial() is a built in function which produces a logical result (i.e. it returns either TRUE or FALSE depending on whether the dial attempt succeeded), hence it can be used directly as a condition in a decision statement, as is shown above.

The logical condition in a decision statement can also be made a compound expression, by linking smaller logical expressions with logical operators, e.g.:-

```
IF Dial("MICROP") AND OnLine() THEN
   WaitFor("User name?");
   Transmit("John Smith|");
END;
```

where OnLine() is another built in function returning a logical result. The above statement means that IF the dial attempt was successful, AND the carrier is still high, then the script processor should execute the two dependant commands, otherwise it should not.

Sometimes you want the script to select one of two sequences, depending on the result of the condition. This is handled by making use of an optional **ELSE** clause in the IF statement. Here is the definition of an IF statement using an ELSE clause:-

```
IF <logical expression> THEN
   <statement sequence>
  ELSE
   <statement sequence>
END;
```

and here is an example using that format:-

```
IF Dial("MICROP") THEN
    WaitFor("User name?");
    Transmit("John Smith|");
  ELSE
    Alarm(1);
    Write("Could not connect to MICROP BBS.|");
END;
```

notice that the first sequence of commands (between the THEN and the ELSE) is executed if the condition evaluates to TRUE, and the other sequence of commands (between the ELSE and the END) is only executed when the condition evaluates to FALSE. In either case, only *one* of the two sequences of commands is executed, and the other sequence is skipped.

There is a final form of the IF statement which is best explained by a substantial example showing why the above formats can be inconvenient in some circumstances. Take a look at the following script, the purpose of which is to log on to a BBS which provides three different access numbers. The script dials the first number, if that fails it tries the second, and if that fails it tries the third. If the third number fails then the script gives an error message and halts. If the script managed to connect to the BBS using any of the numbers then it will proceed to log on. The script assumes that "FIRST", "SECOND" and "THIRD" match keys of three different entries in the dialing directory. Here goes:-

```
SCRIPT Login;

VAR Success : Flag;

BEGIN
    Success := FALSE;
    IF Dial("FIRST") THEN
        Success := TRUE;
      ELSE
        IF Dial("SECOND") THEN
            Success := TRUE;
          ELSE
            IF Dial("THIRD") THEN
                Success := TRUE;
              ELSE
                Write("All numbers were engaged|");
            END;
        END;
    END;

    IF Success THEN
        WaitFor("User name?");
        Transmit("John Smith|");
        WaitFor("Password: ");
        Transmit("smithy|");
    END;
END;
```

This script will work perfectly well, but it looks rather awkward and long winded because of all those ELSEs and ENDs. The need to have a second condition in the ELSE part of an IF statement is so common that in fact the script language provides a short form way of doing it, like so:-

```
SCRIPT Login;
```

```
VAR Success : Flag;

BEGIN
    Success := FALSE;
    IF Dial("FIRST") THEN
        Success := TRUE;
      ELSIF Dial("SECOND") THEN
        Success := TRUE;
      ELSIF Dial("THIRD") THEN
        Success := TRUE;
      ELSE
        Write("All numbers were engaged! |");
    END;

    IF Success THEN
        WaitFor("User name?");
        Transmit("John Smith|");
        WaitFor("Password: ");
        Transmit("smithy|");
    END;
END;
```

Notice that the "ELSE .. IF" words have become a single "**ELSIF**", and that this form is treated as a clause in the first IF statement, and so does not require an END to terminate it. The following is the final definition of our IF statement.

Note that we have introduced a refinement to our definition notation in order to cope with the idea of repetition, and with optional features. If a section of the definition is in curly braces like so **{ }**, then this part of the definition can be repeated as many times as you wish, including zero times. If part of the definition is shown in square brackets like so **[ ]**, then this part is optional, but if used can only appear once.

```
    IF <logical expression> THEN
         <statement sequence>
   {    ELSIF <logical expression> THEN
         <statement sequence>             }
   [    ELSE
         <statement sequence>             ]
     END ;
```

As implied by the above definition, any ELSIF clauses (if used) must come before the ELSE clause (if used).

Odyssey Script Tutorial

More Advanced Decision Statements

The previous section introduced decision making using the IF statement. However, the script language also provides an alternative decision making construct called a **CASE STATEMENT**. In Odyssey script language there is no efficiency reason to prefer one or the other, so the use of Case statements becomes a matter of personal choice at a particular time. Using the refined notation described in the previous section, here is the definition of the syntax of a Case statement:-

```
  CASE <expression> OF
      <label> : <statement sequence>
{    |  <label> : <statement sequence>      }
[  ELSE
    <statement sequence>       ]
  END;
```

The idea is that you expect an expression to evaluate to one of a limited number of possible results. You put that expression at the top (between CASE and OF), and then list each of the expected results, and the sequence of commands you would like the script processor to execute when the result matches that entry in the list (that label). If the result matched none of the labels then the ELSE clause is invoked. This syntax is in many instances clearer than the equivalent, a series of IF-THEN-ELSIF clauses.

For example, the following script extract converts from a "day of the week" expressed as a number from 1 to 7, returning the string equivalent, i.e. Monday, Tuesday etc. Monday is assumed to be day one:-

```
VAR DayNumber : Number;
    DayString : String[10];

BEGIN
    CASE DayNumber OF

           1: DayString := "Monday";
      |     2: DayString := "Tuesday";
      |     3: DayString := "Wednesday";
      |     4: DayString := "Thursday";
      |     5: DayString := "Friday";
      |     6: DayString := "Saturday";
      |     7: DayString := "Sunday";
       ELSE
         Write("Not a valid day number! |");
         Exit();
    END;

    Write("That day is ", DayString, "|");
END;
```

In Odyssey script language (and unlike most other programming languages), you can also go the other way, in other words you can use a string variable as the expression, and in the labels, for example:-

```
VAR DayNumber : Number;
    DayString : String[10];

BEGIN
    CASE DayString OF

           "Monday": DayNumber := 1;
```

```
      |   "Tuesday": DayNumber := 2;
      | "Wednesday": DayNumber := 3;
      |  "Thursday": DayNumber := 4;
      |    "Friday": DayNumber := 5;
      |  "Saturday": DayNumber := 6;
      |    "Sunday": DayNumber := 7;
     ELSE
       Write("Not a valid day string! |");
       Exit();
    END;
    Write("That is day number ", DayNumber, "|");
END;
```

You can attach more than one label to the same statement sequence by listing each label separated by a comma. To demonstrate this, the following script extract takes a month number (from 1 to 12) and calculates how many days there are in that month. It assumes that a logical variable "leap_year" exists, which helps it decide whether February should have 28 or 29 days:-

```
CASE month_number OF

               2: IF leap_year THEN
                     days_in_month := 29;
                   ELSE
                     days_in_month := 28;
                 END;
  |        4,6,9,11: days_in_month := 30;
  | 1,3,5,7,8,10,12: days_in_month := 31;
 ELSE
   Write("Not a valid month number! |");
   Exit();
END;
Write("There are ", days_in_month,
    " days in month ", month_number, "|");
```

Note that the sequence of commands following the colon (after the label) is terminated by a "|" character, which is why the bars are only used AFTER the first sequence of commands, and not before the first label (this character is a punctuation symbol quite distinct from the newline character used in strings).

Odyssey Script Tutorial

Looping Statements

The previous two sections have described how to make decisions in scripts. However, there is another fundamental operation which a language requires in order to become useful, and this is some way of repeating a sequence of commands. The operation is known as **LOOPING** or **ITERATION**.

The Odyssey Script language provides two types of iteration, the **WHILE** loop and the **REPEAT** loop. The difference between the two is that a WHILE loop tests a condition at the beginning of the loop, and a REPEAT loop tests the condition at the end, guaranteeing that the loop is executed at least once. We will discuss the WHILE statement first.

Here is the definition of a WHILE loop statement:-

```
WHILE <logical expression> DO
   <statement sequence>
END ;
```

When the script meets the start of the loop the condition is evaluated, and if FALSE, then the statement sequence is skipped, without ever having been executed. If the expression turned out to be TRUE then the statement sequence is executed, after which control returns to the WHILE line and the condition is tested again.

A WHILE loop is used in preference to a REPEAT loop when you know that you may not want to enter the <statement sequence> part of the loop even once. An example of this is shown in the following script, which asks for the name of a file, and then types that file to the terminal screen:-

```
Script Type_File;

VAR f:File;
   filename:String;
   line:String[255];  -- allow for long lines

BEGIN
   Write("Enter name of a text file? ");
   Read(filename);

   /* test if file exists,
     or no filename entered */
   IF filename="" THEN EXIT() END; -- no error
   IF NOT IsFile(filename) THEN
      Write("Could not find ", filename, "|");
      Exit();
   END;

   Fopen(f, filename);     -- open the file
   WHILE NOT Feof(f) DO  -- stop if end of file
      Fread(f, line);     -- read a line
      Write(line, "|");  -- display that line
   END;                    -- and loop again
   Fclose(f);     -- finished loop, close file
END;
```

A WHILE loop is preferred above is because it is possible for the file to exist, but be empty. We do not want to attempt to read a line from an empty file. The above example made use of several built in commands which will of course be properly documented later in the chapter.

Now to the REPEAT loop. As mentioned above, the major difference between WHILE and REPEAT loops are that the latter tests its condition at the end. Here is the definition of a REPEAT loop statement:-

```
REPEAT
   <statement sequence>
UNTIL <logical expression> ;
```

A good example of when a REPEAT loop would be preferred is a keyboard checking routine. In the following script example, you want the user of the script to enter a Y or N (yes or no) response to a prompt. To be robust, the script repeatedly reads the keyboard until a Y or N (or y or n) is entered, ignoring other characters. The script then sets a logical variable got_yes depending on the result of the exercise.

```
VAR ch : Number;
   got_yes : Flag;

BEGIN
   Write("Please enter Y or N: ");
   Priority(TRUE);

   REPEAT
      ch := RdKey();
   UNTIL (ch=78) OR (ch=89)    -- N or Y
     OR (ch=110) OR (ch=121); -- n or y

   Priority(FALSE);

   IF (ch=89) OR (ch=121) THEN
      got_yes := TRUE; Write("Y|");
     ELSE
      got_yes := FALSE; Write("N|");
   END;
END;
```

Notice that a WHILE loop could not be used because the loop must execute at least once in order to assign an initial value to the variable "ch", prior to the first test.

Odyssey Script Tutorial

Constant declarations

Odyssey scripts can include a 'constants declaration section', using a syntax very similar to that used in Pascal (except that Ody script allows constant expressions). Here are some examples :-

```
    CONST     a = 1234;                /* assumed to be of type number */
              b = $1234;            /* ditto */
              c = "A string constant";
              d = "Hello," + "World";         /* a string constant-expression */
              e = (a+b)+1;                           /* a number constant-expression */
              f = Long(a) * a;
              g = 12345678;                       /* assumed to be of type long */
```

You may have as many constants declared as you like in one constant declaration section, and you may have as many constant declaration sections as you like. A constant declaration section can appear before or after (or both) a VAR declaration section, and you can make the const section global, or you can declare a local constant section inside a procedure or function.

Notice that the constant declaration can be an expression, but if so that expression must evaluate to a constant - ie. it must be possible for the Odyssey script compiler to completely evaluate it at compile time. You cannot use any function within a constant expression, except for the type conversion functions "Number()" and "Long()".

Odyssey Script Tutorial

Commands (Procedures)

The Odyssey script language allows you to create your own specialised commands. You create a command by defining a **PROCEDURE**, using the syntax which will be described in this section. Once a command has been created it can be used in later parts of your script just as if it was another built in command.

A procedure definition looks very much like a complete script in its own right, and that is more or less exactly what it is, except that you put a procedure in the same text file as the script which uses it. Like a script, you can declare variables, but unlike variables declared in the main part of the script, these can only be used inside that procedure, and not by other procedures or by the main script. The memory used for variables in a procedure is recovered when you leave the procedure. To use ("call") a procedure, you simply invoke its name - rather like a magic word. When you do that the procedure will execute, and when it is finished, the script will continue from the statement following the call.

A procedure provides at least two concrete benefits:-

- The ability to encapsulate a series of commands which you often use, so that they can be invoked when needed, instead of repeating the same sequence of commands at several places in the script.
- It allows you to make efficient use of memory, since variables required only when a particular section of the script is running do not hog memory for the entire lifetime of the script.

A procedure also provides another, albeit less concrete benefit, which nonetheless may be the most important feature. A procedure allows you to break a large script down into smaller functional blocks, each of which can be more easily understood than if they were all part of one monster main block in the script. This is simply attacking a complex problem using the "divide and conquer" rule.

Consider the following script example:-

```
SCRIPT long_winded;

VAR Success:Flag;

BEGIN
   Write("Dialing MICROP BBS|");
   Success := FALSE;
   /* try first number */
   IF Dial("FIRST") THEN
      IF WaitFor("User name? ",10) THEN
         Success := TRUE;
         Transmit("John Smith|");
         WaitFor("Password: ");
         Transmit("smithy");
      END;
   END;
   IF NOT Success THEN
      /* try second number */
      IF Dial("SECOND") THEN
         IF WaitFor("User name? ",10) THEN
            Transmit("John Smith|");
            WaitFor("Password: ");
            Transmit("smithy");
         END;
      END;
   END;
```

```
END;
```

The above script tries to dial a BBS which has two access numbers. If the first number answers then the script proceeds to log on to the BBS. If the first number is engaged, the script dials the second number, and tries to log on there instead.

Notice how the second "IF Dial(" sequence is practically identical to the first, except that it uses a different key for the dial command. Does this not strike you as wasteful, and long winded? Instead of repeating an identical sequence of commands, a far better solution would have been to encapsulate that sequence of commands as a procedure, as in the following example:-

```
SCRIPT myscript;

VAR Success:Flag;
   Key:String[8];

/*...................................................*/

PROC Login();
BEGIN
   Success := FALSE;
   IF Dial(Key) THEN
      IF WaitFor("User name? ",10) THEN
         Success := TRUE;
         Transmit("John Smith|");
         WaitFor("Password: ");
         Transmit("smithy");
      END;
   END;
END;

/*...................................................*/

BEGIN
   Write("Dialing MICROP BBS|");
   /* try first number */
   Key:="FIRST";
   Login();

   IF NOT Success THEN -- first number not available
      /* try second number */
      Key := "SECOND";
      Login();
   END;
END;
```

Notice how the main body of the script is now much reduced, and that the sequence of commands for logging on is not wastefully repeated in two different places. By the way, the /*... lines are just comment lines used to make the procedure definition stand out a little better. It is not a requirement of the language.

Another thing to notice is how the Dial() command in the procedure makes use of the variable "Key" which is assigned a value in the main body of the script. Now you know why you need to put quotes around a string when you pass it to a command - because commands which expect a string can accept either literal strings or variables, and it isn't possible to tell which is which unless you wrap literal strings in quotes.

There is one irritating feature of the modified script, and that is the need to declare an extra variable in the main script called "Key", whose sole purpose is to pass data to the login procedure. This variable is not required after login is successful, and yet this variable will not be discarded - it will occupy memory for the entire lifetime of the script, as well as clutter up the list of variables which will be more important after login. It would be nice if we could tell the procedure what key to use without having to declare a variable in the main area of the script (called a "global variable" for short).

As you have probably guessed by now, there is indeed a way to pass data to a procedure without having to declare a global variable. The solution is to declare the variable as a **PROCEDURE PARAMETER**. Here is the last script again, this time using a procedure parameter instead of a global variable. Notice how this affects both the definition of the procedure, as well as the main body of the script:-

```
SCRIPT myscript;

VAR Success:Flag;

/*..............................................*/

PROC Login( Key:String );
BEGIN
   Success := FALSE;
   IF Dial(Key) THEN
      IF WaitFor("User name? ",10) THEN
         Success := TRUE;
         Transmit("John Smith|");
         WaitFor("Password: ");
         Transmit("smithy");
      END;
   END;
END;

/*..............................................*/

BEGIN
   Write("Dialing MICROP BBS|");
   /* try first number */
   Login("FIRST");
   IF NOT Success THEN -- first number not available
      /* try second number */
      Login("SECOND");
   END;
END;
```

If you look at the main body of the script, you will notice that we no longer assign the key names to a variable called "Key", instead we just put the data we would have assigned between the round brackets when we call the procedure. In fact, as far as the main script is concerned, the variable "Key" no longer exists. Only the procedure knows about this variable, and that variable will only come into existence when the procedure is called, and is discarded when the procedure returns.

A point should also be made about string overrides and procedure parameters. Notice that when "Key" was a global variable, we used a string override to limit the string length to eight characters. However, when we made it a procedure parameter no string override was used. This is because string length overrides are not allowed in procedure parameters. This is not a limitation - procedure parameters have no need of overrides, because they automatically allocate exactly the memory they require to hold the data passed to the procedure, and no more. In other words, you can pass a string of any size to a procedure which takes a string parameter. However, this also means that you should avoid assigning values to value string parameters, because you do not know inside the procedure what the maximum length of that string is. In general, you should consider it bad practice to assign values to any procedure parameter from within the procedure.

If your procedure needs to use variables for internal purposes, i.e. not to receive data from the caller, then you can declare those variables actually inside the procedure, like so:-

```
PROC Login( Key:String );

VAR a,b : Number;      -- variables used internally by the
    s : String;       -- procedure.

BEGIN
    IF Dial(Key) THEN
        ...
```

These internal variables are the opposite of a global variable, and are thus referred to as **LOCAL VARIABLES**. A procedure parameter is also viewed by the procedure as a local variable, even though it is declared in the parameters section. The distinctive feature of a local variable is that it is only known about by the procedure which declared it - the variable cannot be used by the main body of the script, or even by another procedure unless it is passed as a parameter.

Here then is the definition of the syntax of a user defined procedure using what we have learned so far. Notice that both the <parameters> and <local variables> symbols are in square brackets, meaning that both are optional. The round brackets of a procedure header however are not optional, and must be present whether or not the procedure uses parameters:-

```
PROC <name> ( [ <parameters> ] ) ;

[ <local variables> ]

BEGIN
      <statement sequence>
END ;
```

Returning to our example, remember that the purpose of the script was to connect to a BBS using one of two access numbers. We set a global variable called "Success" if we succeeded on either one.

Now suppose that we want to keep a note of which access port we were successful with, how would we modify our procedure to do that? Well, instead of having one global flag for success, we could have two - one for each access number, looking at which was set to TRUE would tell us which attempt was successful. So assuming that we have variables called "Success_1" and "Success_2", how would the procedure know which one to set?

Well, we could do something like this:-

```
PROC Login( Key:String );
BEGIN
    IF Key = "FIRST" THEN
        Success_1 := FALSE;
      ELSIF Key = "SECOND" THEN
        Success_2 := FALSE;
    END;
    IF Dial(Key) THEN
        IF WaitFor("User name? ",10) THEN
            IF Key = "FIRST" THEN
```

```
                Success_1 := TRUE;
              ELSIF Key = "SECOND" THEN
                Success_2 := TRUE;
            END;
            Transmit("John Smith|");
            WaitFor("Password: ");
            Transmit("smithy");
        END;
    END;
END;
```

but that has several problems. First of all it looks messy, because it has to check the contents of "Key" everywhere. Secondly, the procedure now has to "know" not only how to do its own job, but the names of all the keys which are going to be passed to it. This is a pity, because otherwise the procedure was completely general, and would have worked with any key, making the script easier to adapt for future needs.

Another trick we could try would be to use another procedure parameter, (called "Success", say), and have the procedure modify that parameter instead of the global variable. Well, nice try, but that won't work. Why not? Well, remember that we said above that procedure parameters are really local variables - they come into existence when the procedure is activated, and are discarded when the procedure returns. Modifying such a parameter would be like making changes to a photocopy of a document - the original remains unaffected.

It would be nice if we could use a version of the procedure parameter idea which allowed us to send the "original" to the procedure, so that the procedure could modify it for us, and send it back.

A way is provided to do exactly that, using a technique called **VARIABLE PROCEDURE PARAMETERS**. To make a parameter variable, you just put the word "**VAR**" in front of the variable name. Here is the example script, modified to use two separate Success variables as described above, and using a VAR parameter to return the modified success flag:-

```
SCRIPT myscript;

VAR Success_1,Success_2:Flag;

/*.............................................*/

PROC Login( Key:String; VAR Success:Flag );
BEGIN
    Success := FALSE;
    IF Dial(Key) THEN
        IF WaitFor("User name? ",10) THEN
            Success := TRUE;
            Transmit("John Smith|");
            WaitFor("Password: ");
            Transmit("smithy");
        END;
    END;
END;

/*.............................................*/

BEGIN
    Success_2 := FALSE;
    Write("Dialing MICROP BBS|");
    /* try first number */
    Login("FIRST", Success_1);
    IF NOT Success_1 THEN -- first number not available
       /* try second number */
       Login("SECOND", Success_2);
    END;

    IF Success_1 THEN
       Write("Connected on FIRST number.|");
      ELSIF Success_2 THEN
       Write("Connected on SECOND number.|");
      ELSE
       Write("Could not connect on either number.|");
    END;
END;
```

If you look at the two calls to "Login" you will see how the individual success variables are passed to the procedure. Taking the first call, what happens in effect is that the value of the global variable "Success_1" is assigned to the local variable "Success" which the Login procedure knows about. When the procedure returns, the contents of "Success" are copied back into the original variable "Success_1", hence the original is modified as the copy was.

Our previous definition of a procedure syntax used the symbol <parameters> without really defining what that meant. Here then is an exact definition of the syntax of the procedure parameters part. Remember that { } means a repeated section, while [ ] means an optional section:-

[**VAR**] <name> {,<name>} : <type>
{ ; [**VAR**] <name> {,<name>} : <type> }

That looks a little complicated, so take your time when trying to understand it. In the meantime, here are a series of examples showing valid procedure headers (ie., just the procedure name and the parameter declarations):-

```
PROC NoParams();
PROC OneParam( a:Number );
PROC TwoParams( a,b:String );
PROC TwoAgain( a:Number; b:String );
PROC VarFirst( VAR a:Number; b:String );
PROC VarSecond( a:Number; VAR b:String );
PROC NumAndTwoVars( a:Number; VAR b,c:String );
PROC TwoVarsAndNum( VAR a,b:String; c:Number );
PROC ThreeVars( VAR a,b:String; VAR c:Number );
```

☞ The VAR modifier only applies to the variable name or list of names immediately following. If there is a semicolon and another list of variables then these are NOT variable parameters unless there is also another VAR. For example, in the case of "VarFirst" above, variable "a" is a variable parameter, but variable "b" is not. In "TwoVarsAndNum", variables "a" and "b" are VAR parameters, while "c" is not.

Odyssey Script Tutorial

Commands (Functions)

Odyssey supports a special type of procedure, known as a **FUNCTION PROCEDURE**, or simply as a **FUNCTION**. A Function is like a normal procedure, except that it is designed to be used in expressions, although it can also be used like a normal procedure. We have already seen how variables are used in expressions, and a function really works a bit like an intelligent variable, in that the expression uses the function as if it were an ordinary variable with an ordinary value, ignoring the fact that the value is actually calculated on the fly, when the function is named.

You can do all the things with a function that you could with a procedure. The only difference is that a function starts with the word **FUNC** instead of PROC, and that you have to tell the script processor what type of variable the function will look like. Here is a definition of the syntax of a function declaration:-

```
FUNC <name> ( [ <parameters> ] ) : <type> ;

[ local variables ]

BEGIN
   <statement sequence>
END ;
```

When we write a function, and want to return a result of the appropriate type to the calling expression, we use a **RETURN STATEMENT**, as follows:-

```
BEGIN
   ....
   RETURN <expression> ;
   ....
END ;
```

Use of the RETURN statement in a function causes an immediate exit from the function, with control returning to the calling expression. In fact, the return statement can also be used in a normal procedure to force an immediate return, simply using a RETURN statement without an expression.

Returning to the example we used in the section on procedures, we could convert the "Login" routine into a function, allowing us to get rid of the "Success" variable, like so:-

```
SCRIPT myscript;

/*..............................................*/

FUNC Login( Key:String ) : Flag;
BEGIN
   IF Dial(Key) THEN
      IF WaitFor("User name? ",10) THEN
         Transmit("John Smith|");
         WaitFor("Password: ");
         Transmit("smithy");
         RETURN TRUE;
      END;
   END;
   RETURN FALSE;
END;

/*..............................................*/
```

```
BEGIN
   Write("Dialing MICROP BBS|");
   /* try first number */
   IF NOT Login("FIRST") THEN
      /* try second number */
      Login("SECOND");
   END;
END;
```

Notice how the main body of the script is allowed to use the function both in an expression (in the <logical expression> part of an IF statement), and also as a normal procedure call.

Although the example above showed a function defined as returning a Flag (logical) type, functions can also be declared as returning any other type, eg. Number, File or String.

In the case of string functions, you cannot use a string override when you declare the function type. In other words the following is *not* legal syntax:-

```
FUNC string_func () : String[20];
```

As in the case of procedure parameters, this is not a limitation, it is simply not necessary to define a string size for a string function, because a string function can return a string of any size.

Odyssey Script Tutorial

Include files

Scripts can use include files, eg :-

```
#include "mystuff.inc"
```

When the script compiler encounters the above line it acts as if that line was replaced by the contents of the file "mystuff.inc". Include files can be used for anything you like, eg. utility functions that you use in a lot of scripts, common constant declarations needed by several different scripts, etc.

Include files can themselves use include files, however one should avoid letting include files nest too deeply, since the system may run out of free file handles.

One should generally try to ensure that you don't include the same file twice in one compilation, as this will almost certainly cause an "Identifier declared twice" error. This happens most easily for example when a script includes a "common declaration header file", and then includes a utility functions file which itself includes the common declarations header. You can avoid this problem by using the conditional compilation features of the Ody script language, eg. if you wrap up your common declarations like this :-

```
#ifndef COMMON_H
#define COMMON_H

CONST   a = 1234;               /* assumed to be of type number */
            b = $1234;              /* ditto */
            c = "A string constant";
            d = "Hello," + "World";
            e = (a+b)+1;
            f = Long(a) * a;
            g = 12345678;           /* assumed to be of type long */

#endif
```

this will ensure that the compiler only sees the above CONST section once in any compilation that pulls in this header file.

If you don't include a path in the include file directive then Odyssey will look for the file in the "Directory for Scripts" specified in the setup dialogs, or in the Odyssey home directory if that setup option is blank. Use a full path specification to override this behaviour.

Odyssey Script Tutorial

Conditional Compilation

By defining compile-time symbols, and by inserting tests into your script of whether a symbol is defined or not, you can control which sections of your script are seen by the compiler. The syntax used for conditional compilation is very similar to that found in a C preprocessor, except that you cannot assign a value to a conditional compilation symbol, and you can't use C type macros.

The conditional compilation directives supported are described below. Note that these directives must appear in your script in lower case, eg. use #define and #ifdef - not #DEFINE or #IfDef.

#define <symbol>

Makes <symbol> known to the compiler. <symbol> must conform to usual Odyssey script language identifier naming conventions, ie. it must start with a letter, and may then contain further letters, digits, or underscores. Also note that like any other script language identifier, the <symbol> is not case sensitive, though the conditional compilation directives themselves *are* (as mentioned above).

#ifdef and #ifndef ('if defined' and 'if not defined')

The syntax of the #ifdef (if defined) directive is :-

```
 #ifdef <symbol>
    <script language statements>
[ #else
    <script language statements> ]
 #endif
```

In other words, if <symbol> is known to the compiler then the first set of script language statements is compiled, otherwise the second set of script language statements is compiled (though only if an **#else** clause is used).

The syntax for the **#ifndef** directive is identical, except that the first script section is compiled if the directive is *not* defined, and so on.

#undef <symbol> (undefine a symbol)

The **#undef** directive makes the compiler forget a symbol. If the compiler never knew that symbol then this statement is ignored (the latter doesn't cause an error).

Odyssey Script Language

Script Commands

This help chapter presents a complete list of all the built in procedure and function commands provided by the Odyssey script language. The commands are grouped into functional categories as follows:-

Modem and Serial I/O Commands

AutoAnswer	Break	HangUp	ModemInit
MNPAnswer	MNPClass	MNPConnect	OnLine
Paste	PortInit	Receive	SetPort
Sleep	Transmit		

Display and Keyboard Commands

Alarm	BackGnd	ForeGnd	ClrEol
ClrEos	ClrLine	ClrScr	ColorDisplay
DelLine	InsLine	DisableCursor	EnableCursor
GotoXY	KeyPressed	RdKey	KillWindow
UseWindow	Window	LoadKeyDef	Menu
Read	SetHelp	WhereX	WhereY
Write			

File and Directory Commands

Chain	ChDir	CopyFile	CurrentDir
DiskSpace	DOS	Download	Upload
Edit	FAppend	FCreate	FOpen
FClose	FDelete	FEOF	FetchStr
FFirst	FNext	FGetPos	FQualify
FRead	FWrite	FRename	FSeek
FSize	IsFile	LastTransferredFile	
MkDir	PickFile	**Shell**	

Dialing Commands

Dial	TagDirEntries	DialTagged	DialQueued

See also: Introduction, Example

Mode Control Commands

CanEscape	CRinTranslation	CRoutTranslation
Emulate	EventLogging	LocalEcho
LogFile	PrinterOff	PrinterOn
RestoreDefaults	SetASCII	SetAutoWrap
SetBackspace	SetCISB	SetDialDelay
SetDialingDir	SetDialPrefix	SetDialTimeout
SetDownloadDir	SetFlowControl	SetMaxRedials
SetRawLogging	SetSoundEffects	SetStripParity
SetZmodem		

Host Mode Commands

FileSize	GetHostInfo	HostShell	WaitForCall

See also: Introduction

Watching and Waiting Commands

WaitFor	WaitForSilence	WatchFor
Received	WatchAgain	ClrWatch
ClrAllWatches	When	WatchEvent
GrabWhen	ReadScreen	

See also: Introduction

DLL Commands

LoadDLL	SendMessage	UnloadDLL

See also: Introduction

Windows API Access Commands

LoadResourceLibrary	FreeResourceLibrary
LoadDialog	GetDialogMessage
EndDialog	GetDlgItemInt
GetDlgItemText	SetDlgFocus
CheckDlgButton	CheckRadioButton
DlgDirList	DlgDirListComboBox
GetDlgCtrlID	GetDlgItem
IsDlgButtonChecked	SendDlgItemMessage
SetDlgItemInt	SetDlgItemText
MessageBox	LoadMenu
DestroyMenu	AssignMenu
CreateToolBar	AssignToolBar
DestroyToolBar	AssignStatus
SetStatusField	WinHelp
FindWindow	SetWindowText
MessageBeep	GetFocus
GetOdyWindow	GetModuleHandle
WinSendMessage	WinPostMessage

See also:
- Introduction
- Accessing any Function in a DLL

Miscellaneous Commands

Address	AllowYield	ASC	CHR
Date	Time	DEC	INC
Delay	DTESpeed	Exit	GetEnv
GetCallInfo	Halt	HaltE	IntToStr
StrToInt	IsDirKey	Length	LogEvent
OdyVersion	Pos	Priority	SetTimer
TimerExpired	SilentMode	StrEdit	SubStr
ToLower	ToUpper		

Odyssey Script Commands

Modem and Serial I/O Commands

The following script commands form the Modem and Serial I/O Commands category:-

AutoAnswer
Break
HangUp
ModemInit
MNPAnswer
MNPClass
MNPConnect
OnLine
Paste
PortInit
Receive
SetPort
Sleep
Transmit

Odyssey Script Commands

AutoAnswer

PROC AutoAnswer(On:Flag);

This command uses the settings in the Setup|Modem (Init strings) dialog to enable or disable your modem's auto-answer mode. It is intended primarily for use in "Host mode" scripts. Remember that sending modem control strings explicitly from a script makes that script non-portable to different types of modem, so use this command instead.

Examples:

```
     AutoAnswer(TRUE); -- enable modem auto-answer
     AutoAnswer(FALSE); -- disable modem auto-answer.
```

Odyssey Script Commands

Break

PROC Break(tenths:Number);

This command asserts the break signal on the modem interface for "tenths" tenths of seconds. Some systems (especially directly connected mainframe computers), require this signal to tell them that a terminal is newly connected and requires attention.

Examples:

```
     Break(10);  -- assert break for one second
     Break(5);   -- assert break for half a second.
```

Odyssey Script Commands

HangUp

PROC HangUp();

This command tells Odyssey to hang up the line, ie. disconnect or go on-hook.

Example:

```
    HangUp();
```

Odyssey Script Commands

ModemInit

PROC ModemInit();

This command uses the "Init String" setting in the Setup|Modem (Init strings) dialog to re-initialise the modem. This is the same string that Odyssey normally sends to the modem when it starts up, or when you press the **ALT+J** command. Remember that sending modem control strings explicitly from a script makes that script non-portable to different types of modem, so use this command instead.

Example:

```
    ModemInit();
```

Odyssey Script Commands

MNPAnswer

FUNC MNPAnswer():Flag;

This command tells Odyssey to check whether a calling modem wants to negotiate an MNP error corrected link. This function would typically be called immediately after the script detects that it has a carrier. The function returns TRUE if an MNP link was established, or FALSE if not.

Example:

```
IF OnLine() THEN
   Got_MNP := MNPAnswer();
```

Odyssey Script Commands

MNPClass

FUNC MNPClass():Number;

This function returns the operating level (class) of the current software MNP connection. Odyssey supports classes 2, 4 and 5. The function will return 0 if software **MNP** is not in use.

Example:

```
     IF Dial("MICROP") THEN
        IF MNPClass() <> 0 THEN
           Write("Software MNP in use.|"); ...
```

Odyssey Script Commands

MNPConnect

FUNC MNPConnect():Flag;

This command tells Odyssey to negotiate an MNP connection with a remote host, which it has just successfully dialed. This command is a hangover from the first version of Odyssey, which did not have a dialing directory. In the current version, when Odyssey is dialing, it is far simpler for a script to allow the dialer to take care of such details as the line parameters, the terminal emulation, and whether or not MNP is required. The only time this command might be preferred to the dialer is if for some reason the dialer cannot be used, perhaps because the modem is not Hayes compatible, and the Odyssey modem configuration section cannot cope with the differences. In this case the script may have to issue the modem dial command explicitly, and also take care of negotiating the MNP link. This function returns TRUE if an MNP link was successfully negotiated, or FALSE if not.

Example:

```
Transmit("ATDP012-345-6789|");
IF WaitFor("CONNECT",30) THEN
   Got_MNP := MNPConnect();
   ....
```

Odyssey Script Commands

OnLine

FUNC OnLine():Flag;

This function tests the status of the carrier (DCD) modem signal, and returns TRUE if the modem is online to a remote modem, or FALSE if it is not. Note that an incorrectly configured cable or modem may cause this function to return a spurious result.

Example:

```
     IF OnLine() THEN
        WaitFor("User name?");
```

Odyssey Script Commands

Paste

PROC Paste(s:String);

This command transfers the string "s" to the serial port. A newline sequence is appended automatically. This function does not perform any special interpretation of characters in the string, neither does it execute any inter-character delays (it does however execute a line delay according to the setting in the ASCII panel of the Setup|File transfer dialog). This makes it faster to use when uploading text, as well as more efficient than the alternative Transmit command when used on an MNP corrected link. On the other hand, the fact that there are no character delays may mean that this command transmits text too quickly for some hosts, in which case the alternative Transmit command will need to be used.

Example:

```
     Fread(infile, line);
     Paste(line);
```

Odyssey Script Commands

PortInit

**PROC PortInit(Baud,Databits:Number;
Parity;Stopbits:Number);**

This command configures the serial port for a required baud rate, data bits and parity. The "Parity" parameter can be **NONE**, **EVEN** or **ODD**. Baud rates should be divided by 100, ie., use 3 for 300 bps, 192 for 19200 bps. This allows you to specify baud rates up to 115200, which would otherwise overflow the maximum value of a Number parameter (note that being able to specify a high baud rate is no guarantee that your hardware will cope with that speed).

Examples:

```
PortInit(12,7,EVEN,2);
PortInit(192,8,NONE,1);
PortInit(576,8,NONE,1);
```

Odyssey Script Commands

Receive

PROC Receive(VAR s:String; timeout:Number [; NoEcho]);

This command tells the script to receive a string from the serial port, terminated by a carriage return. The string, if read, will be stored in the variable "s". A timeout in seconds must be specified. The script can optionally include the **NoEcho** flag, which tells Odyssey that characters must be echoed with a "*" character. A second NoEcho mode is provided which does not echo any character at all. This command is used to allow a remote user to type a string in answer to a prompt generated by an Odyssey script.

Examples:

```
     Receive(s,30);          - read string, with echo.
     Receive(s,30,NoEcho);   - read string, echo '*'.
     Receive(s,30,NoEcho+1); - read string, no echo.
```

Odyssey Script Commands

SetPort

PROC SetPort(PortNo:Number);

This command selects comm port "PortNo" for future serial I/O operations. The command is ignored if the selected comm port does not exist.

Example:

```
    SetPort(2);
```

Odyssey Script Commands

Sleep

PROC Sleep();

This command suspends script execution until the carrier drops. A typical use for this command is when the script has sent a logoff command to the host BBS, and is waiting for that command to produce the expected line drop. The script would then perform any post-connection cleanup it needed, such as closing a log file, or updating a session file. Another use might be to suspend the script while the user interacts with the host directly, meanwhile the script would wait for disconnection before taking control again.

Example:

```
     WaitFor("Command?");
     Paste("bye");
     Sleep();
     LogFile(CLOSE);
```

Odyssey Script Commands

Transmit

PROC Transmit(s:String);

This command transfers a string to the serial port. The characters in the string will be transmitted with inter character and line delays as per the settings of the ASCII panel of the Setup|File transfer dialog. Some characters in the string are not transmitted literally, but instead cause some special action to be performed, as follows:-

Character(s)	Action
~	Pause for half a second.
^x	Send control character x.
^nnn	Send the character whose ASCII code is nnn. This must be a three digit decimal number,, use leading zeroes if necessary to pad out to three digits.
|	Send a newline sequence.

Examples:

```
     Transmit("Is this correct? Y/N: Y^H");
     Transmit("ATZ|~~ATX2|~");
     Transmit("^H ^008");
```

Odyssey Script Commands

Display and Keyboard Commands

The following commands make up the script language **Display and Keyboard Commands** category.

Odyssey Script Commands

Alarm

PROC Alarm(Seconds:Number);

This command causes Odyssey to make an intermittent beeping noise for the required period of seconds. This command would typically be used to warn the user of a normally unattended script that some event has occurred.

Example:

```
     Alarm(5);
```

Odyssey Script Commands

BackGnd,ForeGnd

PROC BackGnd(Color:Number);
PROC ForeGnd(Color:Number);

These commands are used to set terminal window text background and foreground colors from an Odyssey script. These colors will be used for text displayed on the terminal screen, or in a text window if created by the script. The valid background color values range from 0 to 7, and valid foreground colors range from 0 to 15. The mapping of numbers to colors is as follows:-

Background Colors

0=Black, 1=Blue, 2=Green, 3=Cyan, 4=Red, 5=Magenta, 6=Brown, 7=Light Gray.

Foreground Colors

0=Black, 1=Blue, 2=Green, 3=Cyan, 4=Red, 5=Magenta, 6=Brown, 7=Light Gray, 8=Dark Gray, 9=Light Blue, 10=Light Green, 11=Light Cyan, 12=Light Red, 13=Light Magenta, 14=Yellow, 15=White.

Attempts to set bright background colors are ignored, and the equivalent non bright color is set instead, unless the **Emulate 'Blink' attribute** checkbox is disabled in the Setup|Terminal dialog.

Examples:

```
BackGnd(1);      -- set blue background
ForeGnd(15);     -- set white foreground
```

Odyssey Script Commands

ClrEol,ClrEos,ClrLine,ClrScr

PROC ClrEol();
PROC ClrEos();
PROC ClrLine();
PROC ClrScr();

These commands provide various screen or window erasing functions. **ClrEol** clears from the cursor position to the end of the line, **ClrEos** clears from cursor position to end of screen or window, **ClrLine** erases the entire cursor line and repositions the cursor at the left margin, and **ClrScr** clears the entire terminal window, placing the cursor at the origin (top left corner).

Examples:

```
     ClrEol();
     ClrScr();
```

ColorDisplay

FUNC ColorDisplay():Flag;

Obsolete. This function always returns TRUE in the Windows version of Odyssey.

Odyssey Script Commands

DelLine,InsLine

PROC DelLine();
PROC InsLine();

These commands are used for inserting and deleting lines on the terminal window or in a script window.

When **InsLine()** is called the lines on the display from the cursor line and below scroll down by one line, leaving a blank cursor line. The line scrolled out of the display is lost. The cursor does not move.

When **DelLine()** is called the lines on the display from one below the cursor line to the last line scroll up by one line. A blank line scrolls in at the bottom and the old cursor line is lost. The cursor does not move.

Examples:

```
     DelLine();
     InsLine();
```

Odyssey Script Commands

DisableCursor,EnableCursor

PROC DisableCursor();
PROC EnableCursor();

These commands respectively disable (hide) or enable (show) the text cursor for the terminal window or script window. Each window has its own independent cursor, and a call to either of the above commands affects the cursor for that window only.

Example:

```
     Window(5,5,40,4,"",$70);
     DisableCursor();
     Write("Hello - press a key!");
     RdKey();
     EnableCursor();
     KillWindow();
```

Odyssey Script Commands

GotoXY

PROC GotoXY(x,y:Number);

This command positions the cursor at coordinate x,y on the current window. The coordinate system ranges from 0 to <columns-1> in the x direction, and 0 to <lines-1> in the y direction, where <lines> and <columns> indicate the dimensions of the currently active window. Attempts to position the cursor beyond the right margin or below the bottom line will leave the cursor at the rightmost column or last line.

Example:

```
     x:=WhereX(); y:=WhereY();
     GotoXY(0,0); Write("This is an error message.|");
     GotoXY(x,y);
```

Odyssey Script Commands

KeyPressed,RdKey

FUNC KeyPressed():Flag;
FUNC RdKey():Number;

These commands respectively test whether a key is ready to be read, or read a key from the keyboard. The key codes returned by RdKey are as returned by DOS (not Windows). Normal keyboard keys produce a single ASCII code, while function keys produce two codes (a zero followed by a pseudo scan code), requiring two separate calls to RdKey.

The **KeyPressed** command returns TRUE if a key is waiting to be read, or FALSE if not. In either case, the function does not wait for a key.

The **RdKey** command waits for a key press if necessary.

Note: A script cannot use these commands without taking some precautions, because keys are normally read by the foreground Odyssey terminal process, and will not be seen by the script. Therefore, before using either of these commands the script should execute a call to Priority(TRUE) to prevent the terminal process from stealing keys, and then call Priority(FALSE) when keyboard access is no longer required. See elsewhere for a description of the Priority command.

Example:

```
     Priority(TRUE);
     Write("Press any key to stop looping.||");
     REPEAT
          Write("Hello, World!!|");
     UNTIL KeyPressed();
     ch := RdKey();
     Priority(FALSE);
```

Odyssey Script Commands

KillWindow,UseWindow,Window

PROC KillWindow();
PROC UseWindow(win_handle:Number);
FUNC Window(x,y,width,height:Number;
title:String; color:Number):Number;

These commands are provided to allow a script to create, use, and then discard DOS-Odyssey style text windows.

The **Window()** function creates a window of the "Terminal Window" type, with its top left corner at x,y, of width "width" and height "height" (all dimensions in character widths or heights). The window will have a title "title" (an empty string can be supplied if you do not wish a title). The "color" is the IBM PC standard text mode color attribute to give to the window, and its frame. This color attribute is conveniently expressed as a hex number in which the first and second digits are color codes as given in the description of the ForeGnd and BackGnd commands described elsewhere. The function returns a number, called the window handle, which can be used in future calls to UseWindow to select this window for output. The window is by default already selected for output immediately after its creation. When a window is selected for output, all the normal display commands described in this section apply to the script window, and not to the terminal window.

The **UseWindow()** command selects a window for output. The parameter "win_handle" refers to a number obtained from a previous call to the **Window** function. You can also use a handle number of one, which selects the main terminal screen for output. It is not necessary to call **UseWindow** immediately after creating a new window, since the new window is already selected for output. This command is only required if you want to switch to another window without destroying the current window.

The **KillWindow()** command destroys the currently selected window.

☞ **Be extremely careful when using the above commands. Select windows correctly, using only valid handles returned by Window() calls.**

Example:
```
    Priority(TRUE);
    w1 := Window(2,2,40,5,"Window 1",$70);
    w2 := Window(5,5,50,5,"Window 2",$4F);
    Write("This should appear in window 2.|");
    UseWindow(w1);
    Write("This should appear in window 1.|");
    RdKey();
    KillWindow();
    UseWindow(w2);
    RdKey();
    KillWindow();
```

Odyssey Script Commands

LoadKeyDef

FUNC LoadKeyDef(key_fn:String):Flag;

This command loads a keyboard definition file created by the Odyssey "**Keyboard Remapping**" feature. A newly loaded keyboard definition replaces any previously loaded definition, ie. the new keyboard macros are not merged with existing ones. The "key_fn" string parameter is the name of the keyboard template (x.KEY) file. The function returns TRUE if the keyboard definition was successfully loaded, FALSE if it was not.

Examples:

```
     LoadKeyDef("MYKEYDEF");
     IF NOT LoadKeyDef("MYKEYDEF") THEN
         ....
```

Odyssey Script Commands

Menu

FUNC Menu(x,y,width:Number;
Title:String;
option1 {,optionN}:String;
Menu_Flags:Number):Number;

This command allows the script to create and use a floating popup menu. The top left corner of the menu will be positioned at x,y notional character widths/heights from the origin of the Odyssey MDI desktop area.

The "width" and "title" arguments are ignored in the Windows version of Odyssey.

Following the title argument is a variable number of string parameters, each specifying one menu option, with a maximum of forty characters in each. Up to sixteen menu options are allowed. The menu flags parameter allows you to give some optional "flavours" to the menu, such as the ability to leave the menu without making a selection by pressing the <Esc> key. The "menu_flags" values allowed are as follows:-

Menu_Flags **Value**	**Meaning**
0	User cannot cancel the menu by pressing <Esc>.
1	User may cancel menu by pressing <Esc>

The DOS version of Odyssey allowed further menu_flags values of 2 and 3 (ie. bit 1 on or off), which controlled whether or not a menu item could be selected by typing the first character of the menu item. While scripts which set these values are accepted by the Windows Odyssey script compiler, this setting will actually be ignored. However, the same effect can be achieved by prefixing the "selection character" in the menu item with an ampersand '&' - which causes that character to be underlined, and allows that character to be used to select the item.

The Menu() function returns the number of the menu option selected (zero being the number of the first option), or else the function returns -1, meaning that the user pressed <Esc> - this can only happen if the "menu_flags" value allowed escape.

Example:

```
Select := Menu(5,5,24,"Menu Title",
             " 1 - First menu option  ",
             " 2 - Second menu option ",
             " 3 - Third menu option  ",
             " Q - Quit               ",
             3);
CASE Select OF
      -1: (* user hit escape *)
  |     0: (* first option *)
  |     1: (* second option *)
  |     2: (* third option *)
  |     3: (* quit selected *)
END;
```

Odyssey Script Commands

Read

PROC Read(VAR s:String [; NoEcho]);

This command allows the script to read a string from the keyboard. The resulting string is placed in the variable "s". If the optional "NoEcho" modifier is used then the read string routine will echo "*" for characters typed, instead of the characters themselves. This is useful for secure entry of passwords etc.

Examples:

```
     Write("Enter your user name: ");
     Read(Username);
     Write("Enter your password:  ");
     Read(Password,NoEcho);
```

Odyssey Script Commands

SetHelp

PROC SetHelp(help_msg:String);

This command allows a script to display a single line help message on the status line.

The "help_msg" parameter is a string containing the literal text to be displayed, except that any part of the string enclosed in curly braces { } will be highlighted when displayed.

To remove a single line help message from the display simply call the SetHelp command with an empty string as a parameter.

Calls to SetHelp may be nested, ie. a previous help line is not destroyed when overwritten by a new line. A call to SetHelp with an empty string parameter will remove the latest help line, causing the previous help line to become visible once more. There is a maximum nesting limit of ten help lines, after which further SetHelp calls are ignored (unless they are removing a help line rather than placing one).

Examples:

```
     SetHelp("Press: {F2}-select  {Esc}-cancel");
     SetHelp(""); -- remove help line.
```

Changes:

A new feature in the Windows version of Odyssey allows you to split the help line up into "panels", such as the panels which are used in the Odyssey Terminal Window and Text Editor status lines. To do this, simply prefix a sequence of characters in the help line text with a field width enclosed in square brackets. For example :-

```
     SetHelp("[20]PanelText1 [30]PanelText2 [0]PanelText3");
```

The above example creates a status/help line with three panels, the first panel being 20 average character widths wide, the second being 30 avcharwidths wide. The third panel has a width of zero, which is interpreted by Odyssey as meaning "make this panel occupy the remainder of the status line". Only the last panel may have a width of zero. If the zero width panel did not exist then a fourth panel would have been created to occupy the remainder of the status line.

Odyssey Script Commands

WhereX,WhereY

FUNC WhereX():Number;
FUNC WhereY():Number;

These commands allow a script to respectively determine the current cursor X and Y coordinates, relative to the terminal or script window origin (whatever is currently selected for output). These commands are typically used either to ensure that the cursor is at a particular screen location prior to transmitting a string (useful for hosts with complicated login screens), or else are used to store current cursor coordinates in order to restore cursor position later.

Example:

```
WHILE (WhereX()<>30) OR (WhereY()<>12) DO
   (* wait for host to finish login screen *)
END;
Delay(1);
Transmit("password|");
```

Write

PROC Write(item {,item} : String_or_Number);

This command is used to write data to the terminal. It can accept a variable number of string or number parameters. If the parameter is a number then it may optionally be immediately followed by a colon and a value x, causing the number to be right justified in a field padded to width x. If a string parameter contains the character "|" then this will not be displayed, instead this will cause the cursor to be moved to the next line. Characters are actually written to the active terminal emulation, not directly to the display, so it is possible to send escape sequences which control the terminal using the ^x notation described for the Transmit command.

Examples:

```
     Write("Hello, World.|");
     Write("I am ", age, " years old today.|");
     Write("I am ", age:5, " years old today.|");
```

Odyssey Script Commands

File and Directory Commands

The following list of script commands form the **File and Directory Commands** category.

Chain
ChDir
CopyFile
CurrentDir
DiskSpace
DOS
Download,Upload
Edit
FAppend,FCreate,FOpen
FClose
FDelete
FEOF
FetchStr
FFirst,FNext
FQualify
FRead,FWrite
FRename
IsFile
LastTransferredFile
MkDir
PickFile
Shell

Odyssey Script Commands

Chain

PROC Chain(script_name:String);

This command allows one script to execute another. The name of the child script to be executed is passed in the "script_name" parameter. Control does not return to the original script, which is terminated. You can explicitly return control to the original script by using another **Chain** command in the child to invoke the parent. The script to be executed by the chain command must be present in the script directory.

Example:

```
     Chain("MYSCRIPT");
```

Odyssey Script Commands

ChDir

FUNC ChDir(dir_name:String):Flag;

This command is used to change directories from within a script. You must use this command in preference to changing directories from within a DOS shell, or DOS command, since Windows always restores its default "current" directory on return from the shell. The function returns TRUE if the change directory command succeeded in finding the named directory.

Example:

```
     IF NOT ChDir("C:\ODYSSEY\DOWNLOAD") THEN
        Write("Download directory does not exist.|");
        ....
```

Odyssey Script Commands

CopyFile

FUNC CopyFile(source_file,dest_file:String):Flag;

This is a high level function for copying a single file from one location to another. The function returns TRUE if the copy proceeded without error, or FALSE otherwise. The parameter "source_file" must specify a complete file path, and should not contain wildcards. "dest_file" can either be a complete filename, or a file path ending in "\", in which case the copy will keep the same name as the original, and will be stored in the specified location.

Example:

```
IF NOT CopyFile("\backupd\odyssey.dir","\odyssey\") THEN
   Write("Could not copy backup dialing dir.|");
   ....
```

Odyssey Script Commands

CurrentDir

FUNC CurrentDir():String;

Returns the name of the current DOS working directory.

Example:

```
     cdir := CurrentDir();
     Write('Current Directory is "',cdir,'".|');
```

Odyssey Script Commands

DiskSpace

FUNC DiskSpace():Number;

Returns the space available on the current drive, in kilobytes.

Example:

```
   space := DiskSpace();
   Write("The current drive has ",space,"K free.|");
```

Odyssey Script Commands

DOS

FUNC DOS(s:String;
WAIT or NOWAIT [+NOCLEAR]):Number;

Causes the script to pass the command "s" to the DOS command processor for execution.

The WAIT or NOWAIT flags are ignored in the Windows version of Odyssey. Always use "NOWAIT" in new scripts, as the interpretation of these modifiers might change in future.

The optional NOCLEAR modifier tells Odyssey not to clear away the Odyssey screen before executing the DOS command. In the Windows version of Odyssey this means that the DOS application runs inside a minimized window.

This command is retained for backward compatibility with the DOS version of Odyssey. New scripts should avoid calling this function (yes, I know how useful it can be), because future versions of Windows may not run on top of DOS.

Examples:

```
DOS("dir/w", WAIT);
IF DOS("pkunzip zipfile", NOWAIT) <> 0 THEN
    ....
END;
DOS("myprog", NOWAIT+NOCLEAR);
```

Odyssey Script Commands

Download,Upload

FUNC Download(protocol
[; filename:String [,default-action]]):Flag;
FUNC Upload(protocol; filespec:String):Flag;

The **Download** and **Upload** commands are used for file transfer. The Download command receives a file from the host using protocol "protocol", and the Upload command sends a file to the host, also using the protocol specified. Both functions return TRUE if the file transfer succeeded, FALSE if not.

The "protocol" parameter can be one of: ASCII, XMODEM, YMODEM, BYMODEM, CISB, KERMIT or ZMODEM. An additional protocol, GYMODEM, may only be used in the download command.

The **Upload()** command always requires a filename parameter, which may contain wildcards if a batch protocol is used.

The "filename" parameter of the **Download()** command is required if the protocol used is not a batch protocol, ie. if it is one of **ASCII**, **XMODEM**, or **YMODEM**. Remaining protocols do not require, and will not allow a filename to be supplied.

The third parameter of the **Download()** command is optional, and specifies what action Odyssey should take if it finds that the file about to be downloaded already exists. This parameter can take one of the following values:-

- **PromptUser**. The user of the machine running the script is prompted to choose which action he prefers. This is the default if you do not supply the third parameter.
- **CancelTransfer**. Cancel the transfer if the file already exists.
- **KeepOldFile**. Keep the old file by first renaming it, then continue the download.
- **EraseOldFile**. Discard the old file, replacing it with the file about to be downloaded.
- **ResumeTransfer**. If Zmodem or Compuserve B+ is in use, then this option causes a file transfer to be resumed, any conflict is assumed to be caused by an attempt to complete a transfer.

In the case of batch protocols, the script language does not normally allow the use of a second parameter (the filename) since that is handled automatically by the protocol. However, if you wish to specify a third parameter then for batch protocols only you can simply use an empty field for the second parameter (see the last Zmodem example below).

Examples:

```
Download(XMODEM, "filename.ext");
Download(ZMODEM);
success := Upload(ZMODEM, "filename.*");
Download(YMODEM); (* defaults to PROMPTUSER *)
Download(XMODEM,filename,KeepOldFile);
Download(ZMODEM,,ResumeTransfer); --note empty field.
```

Odyssey Script Commands

Edit

PROC Edit(filename:String);

This procedure opens an Odyssey Text Editor window into which is loaded the file indicated by the "filename" argument.

Example:

```
     Edit("READ.ME");
```

Odyssey Script Commands

FAppend,FCreate,FOpen

FUNC FAppend(VAR f:File; filename:String):Number;
FUNC FCreate(VAR f:File; filename:String):Number;
FUNC FOpen(VAR f:File; filename:String):Number;

These commands exist to allow a script to open a file. Each command finds and opens the file "filename", initialises the file variable "f", and associates the file variable with the newly opened file. The file variable is then used in future references to that file when reading, writing, and so forth. Each function returns a number which corresponds to DOS input/output error code, which should be zero if there was no error.

FCreate() creates a new file, ready for data to be written to it. **FOpen()** opens an already existing file. **FAppend()** also opens an already existing file, but does so in such a way that new data written to the file will be appended after the existing file contents. **FOpen()** and **FAppend()** always open files for both reading and writing.

Example:
```
     FAppend(f, "SESSION.LOG");
     FWrite(f, "Session start: ",Date()," ",Time());
     FClose(f);
```

Odyssey Script Commands

FClose

FUNC FClose(VAR f:File):Number;

This command closes an open file, ie. any buffered data is written to disk, the DOS directory is updated, and the file handle is deallocated. All open files *must* be closed before a script exits. The function returns a DOS error code which should be zero if the command succeeded.

Example:

```
     FClose(f);
```

Odyssey Script Commands

FDelete

FUNC FDelete(filename:String):Number;

This command causes the file "filename" to be deleted. The function returns a DOS error code which should be zero if the operation was successful.

Example:

```
     FDelete("FILE.EXT");
```

Odyssey Script Commands

FEOF

FUNC FEOF(f:File):Flag;

This command is used to test whether the file read/write pointer associated with file "f" currently points to the end of the file. The function returns TRUE if so, or FALSE if not. This command is only useful with files opened using the FOpen command, since **FEOF()** is always TRUE for files opened with FCreate or FAppend.

Example:

```
     FOpen(f, "infile.txt");
     FCreate(f2, "newfile.txt");
     WHILE NOT FEOF(f) DO
        FRead(f, line);
        FWrite(f2, line);
     END;
     FClose(f);
     FClose(f2);
```

Odyssey Script Commands

FetchStr

FUNC FetchStr(Key:String;
VAR s1,s2:String;
[filename:String]):Flag;

This command searches a password file for an entry matching the key word "Key", and if found returns two strings from that entry in the "s1" and "s2" parameters. The function returns TRUE if the key was found, or FALSE if not. This command can be used for any purpose, but is primarily a safety feature intended to allow you to avoid accidentally revealing passwords when sharing scripts. Note that this feature is not intended to provide security against persons who have access to your PC. You are encouraged to take your own precautions if you are worried by this prospect.

The **FetchStr()** command by default searches a file called "**PASSWORD.ODY**", but can be instructed to search any other file by supplying the optional filename parameter. In any case, the file searched must be a plain ASCII file laid out as several lines of the format:-

<key> = "string1" "string2"

For example:-

"BIX"	**=**	**"bixid|"**	**"password|"**
"ANYBBS"	**=**	**"John Smith|"**	**"smithy|"**

There should be at least one space on either side of the '=' symbol, and between the two strings to the right. Extra spaces and blank lines between fields will be ignored. The first column is the key, one of which should match the string passed in the "Key" argument to **FetchStr()**. If the key is found then the two string fields which follow are returned in the two string variables. The example below assumes that the sample password file shown above is used.

Note: Previous versions of Odyssey did not allow the key field of the password file line to be in quotes. The preferred method in the current version is to put the key in quotes, although old format password files are still accepted.

Example:

```
VAR Username,Password:String;

BEGIN
     IF Dial("MICROP") THEN
         IF FetchStr("MICROP",Username,Password) THEN
             WaitFor("User name? ");
             Transmit(Username);
             WaitFor("Password? ");
             Transmit(Password);
         END;
     END;
END;
```

Odyssey Script Commands

FFirst,FNext

FUNC FFirst(wildcard:String;   s_attr:Number;
VAR Filename:String;
VAR f_attr:Number):Flag;
FUNC FNext(VAR Filename:String;
VAR f_attr:Number):Flag;

These commands allow scripts to search DOS directories for files matching a known specification.

The **FFirst()** function calls DOS to search the directory according to the file specification passed in the "wildcard" parameter. If files exist which match that specification then the function will return TRUE, and the name of the first of the matching directory entries will be returned in "Filename", with its associated file attributes returned in "f_attr" (see the description of the "s_attr" parameter).

The "s_attr" parameter allows you to control whether special files such as sub-directories will be included in the search. The s_attr parameter is bit mapped, with bits assigned as follows:-

BIT	Meaning
0	Include read-only files in the search.
1	Include Hidden files in the search.
2	Include System files in the search.
3	Search for the volume label.
4	Include subdirectory files in the search.
5	Return files with the archive bit set.

The normal attribute will be 0, which returns only normal files (not hidden or subdirectory entries). The FileAttr value returned by a successful search uses this same bit mapping.

If the initial call to **FFirst()** succeeds in finding a matching file then remaining files can be found through repeated calls to **FNext()**. This function returns TRUE if another matching file was found, and as before the filename is returned in "Filename", with its attributes in "f_attr". The function returns FALSE if there are no more files matching the original specification.

Example:

```
VAR got_file:Flag;
    Filename:String;
    fileattr:Number;

BEGIN
     got_file := FFirst("*.txt",$03,
                       Filename,fileattr);
     WHILE got_file DO
         Write("Found file: ",Filename,"|");
         got_file := FNext(Filename,fileattr);
     END;
END;
```

Odyssey Script Commands

FGetPos

FUNC FGetPos(f:File):Long;

Given the handle of an opened file, this function returns the current position of the file read/write pointer (seek pointer). This function was added because it was finally made possible by the introduction of the new 'Long' data type.

Example:

```
   fseekptr := FGetPos(f);
   ...
   FSeek(f, fseekptr);
```

Odyssey Script Commands

FQualify

FUNC FQualify(filename:String):String;

This command takes a partially qualified filename and returns its fully qualified (unambiguous) equivalent, eg.:-

fullname := FQualify("WINODY.EXE");

might assign "**C:\ODYSSEY\WINODY.EXE**" to fullname. This is useful if you want to change directories, and don't want DOS to become confused as to where your file is.

Example:

```
     Write("Please enter a filename: ");
     Read(filename);
     filename := FQualify(filename);
```

Odyssey Script Commands

FRead,FWrite

FUNC FRead(f:File; VAR s:String):Number;
FUNC FWrite(f:File; s1 {,sn} :String_or_Number):Number;

These commands allow a script to read or write to text files previously opened using the FOpen(), FCreate() or FAppend() commands.

FRead() reads one text line from the file associated with the file handle "f", and assigns that line to the string parameter. The line is expected to be CRLF terminated in the file, but the newline characters are not preserved in the string read. The function returns a DOS error code which should be zero if there was no error.

FWrite() writes one or more items to the file associated with the file handle "f". Each item may be a string or number expression, and should be separated by commas. The FWrite command always adds a newline code to the end of the list of items written, these therefore form one new line in the output file. The function returns a DOS error code which should be zero if there was no error.

Examples:

```
    FRead(f,s);
    FWrite(f,s);
    FWrite(f,"I am ",age," years old today.");
```

Odyssey Script Commands

FRename

FUNC FRename(oldname,newname:String):Flag;

This command renames the DOS file whose current name is passed in the "oldname" parameter to the substitute name passed in the "newname" parameter. This command can also be used to move a file from one directory to another, provided that both directories are on the same DOS drive. The function returns TRUE if the command succeeded, or FALSE if not.

Examples:

```
     FRename("today.log","yesterda.log");
     FRename("\odyssey\bix.log","\old\bix.log");
```

Odyssey Script Commands

FSeek

PROC FSeek(f:File; pos:Long);

Given the handle of an opened file, and a file address relative to the start of that file, this function sets the position of the read/write pointer (seek pointer) to that address. You should normally only use this function on files you are reading, rather than files you are creating. This function was added because it was finally made possible by the introduction of the new 'Long' data type.

Example:

```
   fseekptr := FGetPos(f);
   ...
   FSeek(f, fseekptr);
```

Odyssey Script Commands

FSize

FUNC FSize(f:File):Long;

Given the handle of an opened file, this function returns the size in bytes of that file. This function was added because it was finally made possible by the introduction of the new 'Long' data type.

Example:
```
   FOpen(f, "SESSION.LOG");
   Write("Size of file = ", FSize(f),"|");
   FClose(f);
```

Odyssey Script Commands

IsFile

FUNC IsFile(filespec:String):Flag;

This command allows a script to determine whether a file or directory exists or not. The function returns TRUE if a file or directory exists whose name matches the "filespec" parameter (which may contain wildcards). This function will not find hidden files.

Example:

```
     IF IsFile("*.ZIP") THEN
        ...
```

Odyssey Script Commands

LastTransferredFile

FUNC LastTransferredFile():String;

This function returns the name of the last file transferred using one of the Odyssey file transfer protocols. The result is not defined if you call this function before there have been any file transfers. A point to note is that only the name of the last file transferred is available, so after a batch file transfer it is not possible to determine names for all the files. This function is most useful if you transfer files one at a time and then check the name.

Example:

```
IF Download(ZMODEM) THEN
   filename := LastTransferredFile();
   Write("File: ",filename," received ok.");
END;
```

Odyssey Script Commands

MkDir

FUNC MkDir(dir_name:string):Flag;

This command attempts to create the directory named in the "dir_name" parameter, and returns TRUE if the attempt succeeded, or FALSE if not.

Example:

```
IF MkDir("\ODYSSEY\DOWNLOAD") THEN
   Write("Download directory created.|");
   ....
```

Odyssey Script Commands

PickFile

FUNC PickFile(filespec:string; VAR filename:String):Flag;

This function gives you access to Odyssey's file selector dialog. You pass the function "filespec" which would normally contain wildcard characters, and Odyssey will pop up the file selector dialog, showing all the files which match that specification, allowing the user to select one of the files using the mouse or keyboard.

The function returns TRUE if the user selected a file, or FALSE if the **Cancel** button was clicked or <Esc> was pressed. If TRUE then the argument "filename" will contain the name of the file selected.

Example:

```
    Write("Unpack which downloaded file? ");
    IF PickFile("*.ZIP", zipfile) THEN
       Write("Unpacking: ",zipfile,"|");
       DOS("pkunzip "+zipfile);
       ....
```

Odyssey Script Commands

Shell

PROC Shell();

Obsolete. This command is not supported in the Windows version of Odyssey.

Example:

```
     Shell();
```

Odyssey Script Commands

Dialing Commands

The commands listed below allow a script to control the Odyssey dial function. Please also see Introduction to Script Dialing.

Dial
TagDirEntries
DialTagged
DialQueued
Example Dial Script

Odyssey Script Commands

Introduction to Script Dial Commands

Almost every script will have at least one call to a dial-related command in it. This makes dialing one of the most important topics for the script programmer to understand and be confident about. When writing the script you should be aware of the interaction of the dialing directory, directory tags, the dial queue, the dialer itself, and Odyssey's configuration. This introduction will try to present that information.

Odyssey dials by stepping through and processing the entries in a *dial queue.* A number gets into that queue in one of four ways.

- The user used the *point and shoot* dial method, ie. he highlighted a single entry in the directory and pressed the Dial button (or just double-clicked on the entry). This produces a dial queue with a single item in it.

- The user used the *Tagged Dialing* method, ie. he marked one or more entries in the directory, and then clicked the Dial Tagged button. This produces a dial queue with multiple entries, depending on the number of items which were tagged.

- A script called the Dial() command. This is the script equivalent of point and shoot dialing.

- A script called the TagDirEntries() command, followed by the DialTagged() command. This is how the script manages tagged dialing.

Once the dial queue is generated, it is independant of the tags. The tags themselves stick until they are manually removed, in other words they are not affected by the success or failure of a dial attempt. What *is* affected by the dial attempt is the contents of the queue - when a connection attempt is successful the appropriate telephone number is removed from the queue, otherwise it remains in the queue for a possible future retry.

The dialing procedure is perhaps best understood by means of a few case studies.

Dial Queue with single entry. This is the simplest case. Here Odyssey would fetch the details of the dialing queue entry, and dial the number. If the dialer fails to establish a connection it will redial, up to the maximum number of redial attempts configured in the setup menu. If the dialer succeeds in establishing a connection then Odyssey is automatically set up as per the dialing directory information and we go online to the service.

Dial Queue with multiple entries. Odyssey will start by selecting the first number in the queue and will dial that service. If it fails to connect Odyssey will pause for the *inter-dial delay* defined in setup, and will then step to the next item in the queue. If there are no more items in the queue then Odyssey returns to the first entry and redials. Notice that Odyssey does *not* persist with the first entry until it hits the maximum redial limit, instead it steps immediately to the next entry and only returns to the first if every other number in the queue also fails to connect. Odyssey will keep cycling around the numbers until either a connection is made, or until *each number* has individually been redialed the maximum number of times.

There is another fact to consider: it is not enough for Odyssey simply to establish a connection with the remote modem. It is not the modem the user wants to connect to, it is the host service or BBS that the user is interested in, and a modem connection is no guarantee that the connection is serviceable. The script must therefore have some way of abandoning the connection if it is bad, then connect via one of the other numbers in the queue. In this case the script must not generate a new dial queue, since that queue will be identical to the one previously generated, and will be very likely to establish a connection to the same faulty modem. The script language therefore provides a way to continue dialing using a queue previously generated, minus the entry which made the faulty connection (the latter is automatic - any entry is removed from the queue on connection).

The remainder of this section will document the available script commands which control dialing.

Odyssey Script Commands

Dial

FUNC Dial(key:String):Flag;

This function asks the Odyssey Dialer to dial a number. The parameter "key" should match the key field of a dialing directory entry, and causes Odyssey to generate a dial queue consisting of that single entry. Odyssey then dials the number in accordance with the settings in that entry. The function returns TRUE if the dial attempt succeeded, or FALSE if it failed.

An exception to the above rule arises when the script using the command was started from the dialing directory, in which case Odyssey is already connected to a host and ignores the Dial() command, returning TRUE to satisfy the script that the command succeeded. You can use this feature to attach the same script to several dialing directory entries.

Examples:

```
     success := Dial("MICROP");
     IF Dial("MICROP") THEN
        ...
```

Odyssey Script Commands

TagDirEntries

FUNC TagDirEntries(Key:String;
ClearOldTags:Flag):Number;

A script can take control of tagging dialing directory entries itself - it need not depend on the user to do so. This command allows a script to "tag" all the entries in a dialing directory which have a particular key. If "ClearOldTags" is TRUE then existing tags will be cleared first. This command may also be used to clear tags without setting any new ones, by calling the command with ClearOldTags set to TRUE, and an empty string "" as the key.

The function returns the total count of tagged entries in the directory after this operation. Calling this function also clears the old dial queue.

Example:

```
IF TagDirEntries("MICROP",TRUE)=0 THEN
   Write("No directory entries have key MICROP|");
END;
```

Odyssey Script Commands

DialTagged

FUNC DialTagged():Number;

After tagging directories entries, this command allows you to invoke the Odyssey queued dialing feature. It does so by generating a dial queue consisting of all tagged entries, and then processing the dial queue. The function returns an error code which will be 0 if a connection was made, otherwise error 1 means that there were no tagged entries, and error 2 means that Odyssey dialed all numbers in the queue several times (up to the maximum allowed), but still failed to establish a connection. If a connection is made (return code 0) then the successful entry is removed from the dial queue, thereby allowing the script, if it wishes, to return to other numbers in the queue after this call.

Example:

```
    IF DialTagged()<>0 THEN (* all attempts failed *)
       Write("No answer on any number|");
    END;
```

See also: TagDirEntries().

Odyssey Script Commands

DialQueued

FUNC DialQueued():Number;

Sometimes a script will want to redial, even after a successful connection has been made via a call to DialTagged() - for example, the connection may have been made to a "dead modem", ie. one that that has lost its connection to the host. This function allows the script to continue with other numbers in the dial queue, whereas calling **DialTagged()** again might well result in the same bad connection.

Like **DialTagged()**, this function returns an error code. In this case 0 indicates success, 1 indicates that the dial queue was empty, and 2 means that Odyssey could not establish a connection with any number in the queue, even after redialing.

Example:

```
    IF DialQueued()<>0 THEN (* try another line *)
       Write("No answer on any number|");
    END;
```

Odyssey Script Commands

Example Dial Script

Queued dialing is a rather complex subject, so there follows a further example which puts the above elements together to form a reasonably robust script dialing routine for a service with multiple access numbers.

```
FuncLogin():Flag;

Var BBSok:Flag;
   Id,Password:String;

Begin
    ClrScr();
    Write("|Calling BBS||");
    IF FetchStr("BBS",Id,Password) THEN
       IF TagDirEntries("BBS",TRUE)=0 THEN
          Write("No directory entries have key BBS|");
          RETURN FALSE;
       END;
       SetMaxRedials(4);
       SetDialDelay(1);
       IF DialTagged()<>0 THEN (* all dial attempts failed *)
          Write("No answer on any BBS number|");
          RETURN FALSE;
       END;
       BBSok:=FALSE;
       REPEAT
            IF WaitFor("login:",10) THEN
               BBSok:=TRUE;
             ELSE
               HangUp(); (* drop the bad connection *)
               IF DialQueued()<>0 THEN (* try another line *)
                  Write("No answer on any BBS number|");
                  RETURN FALSE;
               END;
            END;
       UNTIL BBSok;
       Paste("hostname");
       WaitFor("user) ");    Transmit(Id);
       WaitFor("Password:"); Transmit(Password);
       RETURN TRUE;
      ELSE
       Write("|Failed to find Id and Password.|");
       RETURN FALSE;
    END;
End; /* Login */
```

Odyssey Script Commands

Mode Control Commands

The following script commands allow you to examine and modify the Odyssey configuration from a script.

CanEscape
CRinTranslation,CRoutTranslation
Emulate
EventLogging
LocalEcho
LogFile
PrinterOff,PrinterOn
RestoreDefaults
SetASCII
SetAutoWrap
SetBackspace
SetCISB
SetDialDelay
SetDialingDir
SetDialPrefix
SetDialTimeout
SetDownloadDir
SetFlowControl
SetMaxRedials
SetRawLogging
SetScreenMode
SetSoundEffects
SetStripParity
SetZmodem

Odyssey Script Commands

CanEscape

PROC CanEscape(enable:Flag);

This command allows a script to enable or disable the users ability to cancel a script by pressing the <Esc> key. A parameter of TRUE allows the <Esc> key to cancel a script, while FALSE disables this effect. If the <Esc> key is pressed while cancellation is disabled then the key is treated like any normal key press, ie. transmitted to the remote host.

Script programmers should realise that if the user's ability to cancel a script is disabled, then the user has no way at all to stop a runaway script. You should therefore use the **CanEscape()** command only with scripts which have been thoroughly tested.

Examples:

```
CanEscape(TRUE);
CanEscape(FALSE);
```

Odyssey Script Commands

CRinTranslation,CRoutTranslation

FUNC CRinTranslation(CR or CRLF):Number;
FUNC CRoutTranslation(CR or CRLF):Number;

These functions override the "CR xxx translation" options of the Setup|Terminal dialog. A parameter of CR tells Odyssey to leave carriage returns unchanged, while CRLF tells Odyssey to convert an incoming (or outgoing, in the case of CRoutTranslation) CR into a CRLF pair. Both functions return the previous setting of the option as a numeric value, which can be used as a parameter in a later call to the function, to restore the original setting.

Examples:

```
numvar := CrOutTranslation(CRLF);
CrOutTranslation(numvar);
CRinTranslation(CR);
```

Odyssey Script Commands

Emulate

FUNC Emulate(emulation:String):Flag;

This command is used to select a new terminal emulation in Odyssey; the emulation named must be available in the Odyssey program files directory.

As well as loading the emulation, this command also causes a keyboard definition file to be loaded, if one exists which matches the name of the terminal emulation file. The function returns TRUE if the emulation was loaded, FALSE if it was not.

Example:

```
IF NOT Emulate("ANSI") THEN
   Write("ANSI-BBS emulation not available.|");
   Emulate("TTY");
END;
```

Odyssey Script Commands

EventLogging

FUNC EventLogging(enable:Flag):Flag;

This command is used to enable or disable the event logging feature. The "enable" parameter should be "TRUE" to enable event logging, or "FALSE" to disable it. The function result is the previous setting of the event logging option.

Example:

```
     WasEnabled := EventLogging(TRUE);
```

Odyssey Script Commands

LocalEcho

FUNC LocalEcho(enable:Flag):Flag;

This function can be used to disable or enable local echo mode from a script. A parameter of TRUE enables local echo, while a parameter of FALSE disables it. The function returns the original setting of the Local Echo flag.

Examples:

```
     old_echo := LocalEcho(TRUE);
     LocalEcho(old_echo);
```

Odyssey Script Commands

LogFile

FUNC LogFile(Log_command; [filename:string]):Flag;

This command is used to control text logging from within an Odyssey script. The parameter "Log_command" can be one of the following:-

- **OPEN**. Enable text logging to the file "filename", or to "ODYSSEY.LOG" if the filename parameter is not supplied.
- **CLOSE**. Close the log file.
- **SUSPEND**. Cease logging temporarily, but leave the current log file open.
- **RESUME**. Resume logging after a suspension.

The function returns TRUE if the logging command was successful.

Examples:

```
LogFile(OPEN, "MYFILE.LOG");
LogFile(SUSPEND);
LogFile(RESUME);
LogFile(CLOSE);
```

Odyssey Script Commands

PrinterOff,PrinterOn

PROC PrinterOff();
PROC PrinterOn();

These commands are used to turn printer logging off and on. Note that in the Windows version of Odyssey the logged text is not physically printed (submitted to the Windows print manager) until printer logging is turned off.

Examples:

```
     PrinterOn();
     ....
     PrinterOff();
```

Odyssey Script Commands

RestoreDefaults

PROC RestoreDefaults();

A script which has used any of the **Set**xxxx mode control commands described in this manual section to alter Odyssey setup options may wish to restore Odyssey to "factory defaults" before ending the script, so that each script may begin from a known initial setup. This command is provided for that purpose. The **RestoreDefaults()** command causes Odyssey to restore settings by reloading ODYSSEY.CFG, or by using built-in Odyssey defaults if the config file does not exist. **RestoreDefaults()** does *not* however reprogram the serial port.

Example:

```
    RestoreDefaults();
```

Odyssey Script Commands

SetASCII

PROC SetASCII(CharDelay,LineDelay:Number;
BlankExpansion:Flag);

This command allows a script to override the settings in the ASCII panel of the Setup|File transfer dialog, the three arguments to this command corresponding to the three options in that panel. **CharDelay**is the inter-character delay in milliseconds (0-9999), **LineDelay** is the inter-line delay in milliseconds (0-9999), and BlankExpansion tells the ASCII upload routine whether or not blank lines should be replaced with a line containing a single space, to be compatible with host editors which regard an empty line as being an instruction to leave the editor. The latter argument should be TRUE or FALSE.

Example:

```
    SetASCII(20,40,TRUE);
```

Odyssey Script Commands

SetAutoWrap

PROC SetAutoWrap(on:Flag);

Auto-wrap is the feature whereby Odyssey will automatically insert a carriage return into the input stream when the cursor reaches the terminal right hand margin. This feature can be enabled or disabled in the Setup|Terminal dialog, or may be overridden using this command.

Example:

```
    SetAutoWrap(TRUE);
```

Odyssey Script Commands

SetBackspace

PROC SetBackspace(Key:Number; Destructive:Flag);

This command controls whether pressing the Backspace key produces either ASCII BS or DEL in VTxxx terminal emulations. It affects the setting of the equivalent option in the Setup|Terminal dialog. The Key parameter should be either 8 or 127 - other values are treated as meaning the same as 127. The "Destructive" parameter is for future expansion, and is not currently used - you should pass FALSE as a dummy parameter.

Example:

```
    SetBackspace(8,FALSE);
    SetBackspace(127,FALSE);
```

Odyssey Script Commands

SetCISB

PROC SetCISB(AutoInvoke,IntResponse,
SAok,EscapeCtl:Flag);

This command allows a script to control the options which affect file transfer using the Compuserve B+ protocol. The parameters are equivalent to the options presented in the Compuserve B+ panel of the Setup|File transfer dialog.

The following example tells Odyssey that Compuserve B+ auto-invoke is to be enabled, that it should respond to the "interrogate sequence" issued by a Compuserve host, that it is ok to send-ahead (window) during a CIS B+ session, and that control characters other than the default set should *not* be escaped (quoted), unless demanded by the host.

Example:

```
    SetCISB(TRUE,TRUE,TRUE,FALSE);
```

Odyssey Script Commands

SetDialDelay

PROC SetDialDelay(Secs:Number);

The "dial delay" is the time Odyssey pauses for between one dial attempt and the next, the intention of which is to allow time for the modem to recover. The time required varies from modem to modem, with the best modems requiring no delay at all. For safety however, Odyssey sets this delay to one second. A script can override the configured value, using this command, to the number of seconds passed in the parameter.

Example:

```
    SetDialDelay(2);
```

Odyssey Script Commands

SetDialingDir

FUNC SetDialingDir(Filename:String):Flag;

This command allows a script to select an alternative dialing directory file, and returns TRUE or FALSE depending on whether that file was found or not. The alternative dialing directory remains selected until the Odyssey session ends, or until another dialing directory is selected. Passing an empty string "" to this command is equivalent to selecting the default dialing directory file "ODYSSEY.DIR". Odyssey makes no assumptions about where the alternate directory file may be found, so a full file path should be supplied.

Example:

```
    IF NOT SetDialingDir("\ODYSSEY\ALTERNATE.DIR") THEN
       Write("Couldn't find alternate dial directory.|");
    END;
```

Odyssey Script Commands

SetDialPrefix

PROC SetDialPrefix(PrefixS:String);

The "dial prefix" is that part of the modem dial command which precedes the telephone number. Odyssey knows about two dial prefixes, one each for tone and pulse dialing, plus Odyssey has an option in the Setup|Modem (Dial Commands) dialog which tells it whether *Tone* or *Pulse* dialing is currently selected. If tone dialing is selected then the **SetDialPrefix()** parameter will override the tone dialing prefix string. If pulse dialing is currently selected then a call to this command will override the pulse dial prefix string.

Example:

```
    SetDialPrefix("~|~ATDT");
```

Odyssey Script Commands

SetDialTimeout

PROC SetDialTimeout(Secs:Number);

The "dial timeout" is the time Odyssey allows, after sending a dial command, for the modem to establish a connection. If this time period elapses without a successful connection, or a recognised failure string having been received, then Odyssey will abandon the call itself. The **SetDialTimeout()** command overrides the timeout value in the Setup|Modem dialog, making it the number of seconds passed in the parameter.

Example:

```
    SetDialTimeout(60);
```

Odyssey Script Commands

SetDownloadDir

PROC SetDownloadDir(dirname:String);

Sets the directory for downloads to that named in the "dirname" parameter. This command overrides the equivalent option in the Setup|General dialog.

The directory for downloads is the directory in which Odyssey will place all files received from a remote host using a file transfer protocol. Log files are also placed there.

Note that this command does *not* create the named directory. If you name a directory using this command (or the menu option) then Odyssey assumes that the directory must already exist. If it doesn't then you should create it first.

Example:

```
    SetDownloadDir("C:\ODYSSEY\DOWNLOAD");
```

Odyssey Script Commands

SetFlowControl

PROC SetFlowControl(type);

This command allows a script to override the flow control setting configured in Odysseys Setup|Comms... dialog.

The 'type' argument specifies the type of flow control required. Possible values are :-

NONE	- Odyssey should not use flow control.
XONXOFF	- Odyssey should use software (XON/XOFF) flow control.
RTSCTS	- Odyssey should use RTS/CTS hardware flow control.

Examples:

```
      SetFlowControl(NONE);
      SetFlowControl(XONXOFF);
      SetFlowControl(RTSCTS);
```

Odyssey Script Commands

SetMaxRedials

PROC SetMaxRedials(max:Number);

The "max redials" is the number of attempts Odyssey should make, after the first attempt, to dial a number. The total number of dial attempts is therefore "max redials" plus one. The SetMaxRedials() command sets the number of redials allowed, overriding the figure given in the Setup|Modem dialog.

Example:

```
    SetMaxRedials(5);
```

Odyssey Script Commands

SetRawLogging

PROC SetRawLogging(on:Flag);

Odyssey has two text logging modes, "*Raw*" and "*ASCII*". ASCII logging mode means that Odyssey will attempt to produce a log file which can be loaded into an ASCII-only text editor, ie. with control characters and terminal control sequences removed. When raw logging however, these characters and sequences are left intact. The **SetRawLogging()** command enables or disables Raw Logging mode, overriding the option in the Setup|General dialog.

Example:

```
    SetRawLogging(TRUE);
```

SetScreenMode

PROC SetScreenMode(MenuLine,StatusLine,
BigScreen:Flag);

Obsolete. This command is ignored in the Windows version of Odyssey.

Odyssey Script Commands

SetSoundEffects

PROC SetSoundEffects(reserved, Bells, FTransfer:Flag);

This command provides control over sound effects settings in the Setup|General dialog. The first parameter is ignored in the Windows version of Odyssey. The second parameter tells Odyssey whether to beep when the ASCII BEL character is received from the serial port, and the last parameter controls whether Odyssey sounds an alarm when a file transfer completes.

Example:

```
    SetSoundEffects(FALSE,TRUE,TRUE);
```

Odyssey Script Commands

SetStripParity

PROC SetStripParity(on:Flag);

Sometimes a user may wish to connect to a service which expects an eight bit, no parity link, using a low cost network which expects a seven bit even parity link. This can cause problems. For example, if you enable even parity to please the network you find that you cannot transfer files with the host. If you set no parity, then you get strange garbage characters when you talk to the network. The secret is to use eight bits no parity, but enable parity bit stripping. This command may be used to enable or disable the parity bit stripping option, and overrides the option given in the Setup|Terminal dialog. You can also enable parity bit stripping for a service using the dialing directory.

Example:

```
    SetStripParity(TRUE);
```

Odyssey Script Commands

SetZmodem

**PROC SetZmodem(AutoDownload,FullStreaming,
EscCtrls:Flag);**

This command allows a script to control the various Odyssey options which affect a Zmodem protocol file transfer. The arguments to the command correspond to the options in Zmodem panel of the Setup|File transfer dialog. **AutoDownload**, if true, enables the Odyssey feature whereby an incoming Zmodem file header causes the file transfer to begin automatically. **FullStreaming** means that Odyssey does not request the sender to insert pauses, which might be necessary if you are downloading to a very slow floppy drive. **EscCtrls** may be necessary if you are using a network which is not completely transparent to all control characters - the Zmodem protocol already escapes the most troublesome control characters (eg. XON and XOFF), but the ultra cautious may wish to escape them all, which is what will happen if EscCtrls is true.

Example:

```
    SetZmodem(FALSE,TRUE,FALSE);
```

Odyssey Script Commands

Host Mode Commands

The script commands listed below are used in a script which implements a Host Mode for in with Odyssey. See also Introduction

FileSize
GetHostInfo
HostShell
WaitForCall

Odyssey Script Commands

Introduction to Host Mode Scripts

Odyssey's host mode is implemented as a perfectly ordinary script, with a somewhat unusual name - **ODYHOST.HSC**. When you select **Answer (host) mode** from the Odyssey Window menu all that Odyssey really does is run this script. You could replace this script with one of your own if you liked, and Odyssey wouldn't mind in the least. Odyssey doesn't care what the script does, or whether or not that script is precompiled. All Odyssey looks for is a valid script file with the correct name, in the correct place.

A host mode script requires access to certain features which would be rather difficult to implement as script procedures, and so Odyssey provides a few built in commands specifically designed to support host mode. However, there is no requirement that a user written host mode script actually use these calls - they are there if needed.

Printing the contents of ODYHOST.HSC should provide you with a good demonstration of the requirements of a working host mode script.

Odyssey Script Commands

FileSize

PROC FileSize(f:File; VAR Bytes,Xblocks:String);

This command takes an opened file handle as the first parameter, and returns the size of the associated file in Bytes and Xmodem blocks. Note that these are returned as strings, since the sizes are quite likely to be larger than a Number variable can hold (this command predates the introduction of the **Long** type in Odyssey). This command is intended to be used by a Host mode script, to inform a caller about the size of a file he is about to download. In this situation it is not important that the script cannot express the file size as a number.

Example:

```
    FOpen(f,Filename);
    FileSize(f,bytes,blocks);
    FClose(f);
    SendString("|File: "+Filename+"|  "
       +bytes+" bytes, ("
       +blocks+" Xmodem blocks).|");
```

Odyssey Script Commands

GetHostInfo

PROC GetHostInfo(VAR NormPass, PrivPass,
Welcome,HostDir:String;
VAR MNPwanted:Flag);

This command allows a script to "ask" Odyssey for the settings in the Setup|Host mode dialog, and is intended for use by Host mode scripts. A given script may choose to use or ignore these settings, however where appropriate a host mode script should naturally try to do what the user expects.

Example:

```
    GetHostInfo(NormPass,PrivPass,
              Welcome,HostDir,MNPwanted);
```

Odyssey Script Commands

HostShell

PROC HostShell();

Obsolete. This command is no longer available in the Windows version of Odyssey (the compiler accepts the command, but the command no longer works).

Odyssey Script Commands

WaitForCall

FUNC WaitForCall():Number;

The core of any host mode script will be the point at which the script waits for a call to arrive. This command provides a simple way for that function to be carried out. The function returns 0 if a call has been connected, 1 if escape was pressed on the host keyboard (ie. the owner of the host machine wants to leave host mode), any other value is an error code indicating a failure to connect a call (currently there should be no other result codes). If the **Baud rate detection** field is enabled in the Setup|General dialog then the terminal speed will be automatically set to the actual connect speed before control returns to the script.

Example:

```
    CallResult := WaitForCall();
```

Odyssey Script Commands

Watching and Waiting Commands

These commands provide "watch for string" and "wait for string" facilities in the Odyssey script language. See also Introduction.

Odyssey Script Commands

Watching and Waiting - Introduction

Almost all scripts you write for Odyssey will centre around the need to recognise specific strings transmitted by the host. These are usually prompts, and your script needs to respond to those prompts in a particular way. Odyssey provides several different ways of looking for, and responding to such prompts.

The basic method of recognising a string is to use the WaitFor command. Using this command causes the script to stop until the correct string arrives from the modem. However, using this method only allows you to wait for one specific string at a time - this is not always appropriate, because certain situations are bound to arise where you need to check for the arrival of one of several strings, without knowing in which order, if at all, these strings are going to arrive. Odyssey script copes with these more complex requirements through the use of a feature called "Watch Processes". This section will describe the basic WaitFor command first, and will then devote the remainder to describing watch process commands.

Odyssey Script Commands

WaitFor

FUNC WaitFor(target:String [; t_secs:Number]):Flag;

This is the basic "wait for a string" command. The script will stop executing, and will not restart until either the required string arrives, or until the timeout period expires. The timeout is optional, and if not specified the WaitFor command will wait indefinitely. If a timeout is required then "t_secs" should pass the required timeout period in seconds. The function returns TRUE if the string was received, or FALSE if it was not. The FALSE result can only happen if the optional timeout was provided.

The script processor ignores case differences when deciding whether or not the string has been received. For example, both "USER NAME" and "user name" would be accepted by the WaitFor commands given as examples below.

Examples:

```
      WaitFor("User name? ");    -- wait indefinitely.
      WaitFor("User name? ",10); -- time out in 10 secs.
```

Odyssey Script Commands

WaitForSilence

FUNC WaitForSilence(secs,max_wait_secs:Number):Flag;

This command is used when you want to give a host time to update the terminal display, before you issue the next command ("silence" means a period when nothing is written to the terminal). The parameters are "secs" - the number of seconds of silence to wait for, and "max_wait_secs", which is the maximum number of seconds to wait for the required silent period. Use the latter to ensure that a script doesn't hang forever if the modem starts generating a constant stream of noise. A "max_wait_secs" parameter of 0 tells the script processor not to time out this command. The function returns TRUE if the required period of silence was seen, false if the timeout expired first.

If there has already been more than "secs" seconds of silence prior to the WaitForSilence() call, then a TRUE result will be returned immediately.

Examples:

```
      WaitForSilence(5,0); -- expect 5 seconds of silence.
      IF WaitForSilence(5,30) THEN...
```

Odyssey Script Commands

WatchFor

FUNC WatchFor(target:String):Number;

This command tells Odyssey to set up a process in the background which will watch whether the string "target" ever arrives. The function returns a number (called a handle) which is used to identify this process in calls to related functions. For example, in order to find out whether the string has been received, you test function Received(handle).

It is important to realise the difference between calls to **WatchFor()** and WaitFor(). **WaitFor()** waits for a single string, and the script is suspended until that string arrives. **WatchFor()** is a function which tells Odyssey that you want to know when a particular string arrives, but having noted this fact the script processor immediately proceeds to the next script statement.

You can have more than one string being watched for at any one time - up to thirty are allowed. If that limit is exceeded then **WatchFor()** will return -1 as the identifying number of the next string.

Example:

```
VAR mailp:Number;

BEGIN
    mailp := WatchFor("Mail:");
```

Odyssey Script Commands

Received

FUNC Received(handle:Number):Flag;

This function is used to test whether a particular string associated with a watch process has arrived yet. The handle is as returned in a previous call to WatchFor(). This function returns either TRUE (yes, the string has been received), or FALSE (no, it has not).

As usual, differences of case (upper or lower) are ignored in deciding whether a particular string has been received.

Examples:

```
got_mail := Received(mailp);
IF Received(mailp) THEN
    ....
```

Odyssey Script Commands

WatchAgain

PROC WatchAgain(handle:Number);

Normally, once a WatchFor() string has been received, calls to the Received() function will return TRUE evermore. This procedure resets the received attribute of a particular string (identified by the handle returned by **WatchFor**). This means that if a watch process has been set up and a string received, then you can instruct Odyssey to look for another occurrence of the same string.

Example:

```
     WatchAgain(mailp);
```

Odyssey Script Commands

ClrWatch

PROC ClrWatch(handle:Number);

This command cancels the watch process associated with this handle. The handle is freed up and may be reallocated by future calls to WatchFor().

Example:

```
     ClrWatch(mailp);
```

Odyssey Script Commands
ClrAllWatches
PROC ClrAllWatches();

This command cancels *all* currently executing watch processes. All handles are made available for reuse.

Example:
```
     ClrAllWatches();
```

Odyssey Script Commands

When

FUNC When(s1:String; s2:String):Number;

This command is a specialised variant on a watch process. The function accepts arguments s1 (a string to watch for), and s2 (a string to send to the remote host when s1 is received). The transmission of s2 is automatically handled by Odyssey. A **When()** call counts as a call to WatchFor() - one handle from the maximum of 30 allowed is allocated to this task (the function returns the handle number allocated). Calls to ClrWatch() may be used to cancel individual **When()** processes, and a call to ClrAllWatches() will cancel all watch processes, including the **When()** variants.

Example:

```
     When("more?","y|");
```

Odyssey Script Commands

WatchEvent

FUNC WatchEvent(handle1, {,handleN} : Number;
[timeout:Number]):Number;

This function takes one or more (ie. a variable number) of arguments, each of which is a handle returned by a previous call to the WatchFor() function. A timeout may also be supplied, and this is described later. This function suspends the script until one of the strings associated with the listed handles is received, and the function then returns the handle of the received string. For example:-

```
apple  := WatchFor("Apple");
orange := WatchFor("Orange");
lemon  := WatchFor("Lemon");

CASE WatchEvent(apple,orange,lemon) OF

      apple:Write(" -- Keeps the Doctor away|");
  |   orange:Write(" -- It's Juicy, Ma!|");
  |    lemon:Write(" -- Suck it and see...|");
END;
```

Another technique is to assign the **WatchEvent()** result to a variable, so that the script can use its knowledge of which handle was satisfied, i.e. :-

```
Handle := WatchEvent(apple,orange,lemon);
WatchAgain(Handle); (* prime watch process again *)
CASE Handle OF
 (* ... etc                  *)
```

If a timeout is specified it must be as a number constant (ie. not a variable), and it must be as the last argument to the function. The number is a timeout in seconds. The following is an example of the function in use:-

```
mainw := WatchFor("Main:");
readw := WatchFor("Read:");
mailw := WatchFor("Mail:");
LostCarrier := FALSE;
REPEAT
     CASE WatchEvent(mainw,readw,mailw,30) OF

            mainw:Write("[Got MAIN prompt]");
       |     readw:Write("[Got READ prompt]");
       |     mailw:Write("[Got MAIL prompt]");
       |   TIMEOUT:Write("[30 seconds are up!]");
       | NOCARRIER:Write("[Script lost carrier]");
                 LostCarrier:=TRUE;
     END;
UNTIL LostCarrier;
```

Note the constants **TIMEOUT** and **NOCARRIER** which can be used as labels to test the result of the **WatchEvent()** call. These labels can also be used in other expressions, such as:-

```
IF WatchEvent(mainw,readw,30)=TIMEOUT THEN...
```

or again:-

```
    num := WatchEvent(mainw,readw,30);
    IF num = TIMEOUT THEN...
```

The **NOCARRIER** test is *not* optional, in other words a carrier loss will always cause **WatchEvent()** to terminate, and good practice should dictate that you test for this possibility as in the above code.

Odyssey Script Commands

GrabWhen

FUNC GrabWhen(Target:String; VAR line:String):Number;

Sometimes it is useful to be able to capture data from the incoming stream of text, and to do that you need to start recording characters at the right time. This function provides you with the means to do this easily. It watches for a string which you specify, and when it sees that string it copies the remainder of the incoming line which contained that string into a string variable.

The first argument to this function is the string to watch for, and the second argument is the string variable into which the resulting line is to be stored. The latter *must* be a global string variable - local string variables will not be accepted because of the possibility that the procedure scope which activated **GrabWhen()** may be no longer active when the process is satisfied, a mistake which would cause the script processor to mangle its stack.

This routine is another variant on a "watch" process, and the function result is a handle for the watch process, as per WatchFor() and When(). The usual rules for watch variants apply, ie. a call to **GrabWhen()** counts as a use of **WatchFor()**, and one of the maximum 30 watch processes is allocated. You can use the Received() function to test whether the process has been satisfied. For this variation on **WatchFor()** the "received" attribute is set only after the string has been seen, *and* the line has been copied.

You may use the handle returned by **GrabWhen()** in calls to WatchEvent(), WatchAgain(), ClrWatch() and **Received()**, and a call to ClrAllWatches() will also cancel any **GrabWhen()** processes still running.

Example:

```
     mailp := GrabWhen("You have mail from ", userid);
```

Odyssey Script Commands

ReadScreen

PROC ReadScreen(x,y,len: Number):String;

Not, strictly speaking, a watching command, but it *is* normally used in that context, so we have documented it in this section. This command reads a string directly off the terminal display, where X and Y are the screen coordinates (0-79,0-<height-1>), and LEN is the number of characters to copy. If you use an illegal x,y or a zero length then an empty string will be returned. The copied string is returned in the function result.

This command is intended to be used in situations where the normal watching and waiting commands are not suitable. For example, when using ANSI emulation it is quite possible that a string you can see on the screen will not be seen by a WaitFor(), because of invisible control characters or terminal control sequences embedded in the string (if the string uses more than one color then this *must* have happened). On the other hand, **ReadScreen()** "sees" exactly what you do - so it is possible to read the string right off the screen, compare it with what you expect, and thereafter proceed as you would have done with **WaitFor()**.

Example:

```
     MyStringVar := ReadScreen(2,2,30);
```

Odyssey Script Commands

DLL Commands

These commands are provided to allow a script to communicate with Odyssey DLLs.

LoadDLL
SendMessage
UnloadDLL

See also: Introduction

Odyssey Script Commands

Introduction to DLL Commands

DLLstands for ***Dynamically Linked Library***. Odyssey terminal emulation, file transfer protocol, editor and FAX modules are all implemented as DLLs, that is, as separate semi-independant programs which are loaded and unloaded as required. DLLs are quite similar to the more old fashioned overlay schemes, except that DLLs are more efficient - they have more to do with memory organisation than with memory economisation.

Skyro Software hopes to add many optional modules in the form of add-on DLLs to Odyssey in the future. Rather than being forced to augment the script language to add specific support for the features of each module every time one appears, we have decided to add a generic, message based support, though at the moment the FAX module is the only one which recognises messages sent from a script, as of this writing (see Fax Server Script Interface).

Odyssey Script Commands

LoadDLL

FUNC LoadDLL(DLLname:String):Number;

Function LoadDLL loads a named DLL module into memory, and returns a handle which will be used to reference that module in future calls. This is analagous to a file open call. The "DLLname" parameter should be the name of the DLL, including the extension, but not including the path (the DLL is assumed to reside in the Odyssey home directory).

For example, to open the FAX server DLL, the command would be:-

```
          fax_dll := LoadDLL("FAXSERV.DLL");
```

Where "fax_dll" is the returned DLL handle. This handle variable can in fact be called anything you like, provided you have declared it as a number variable earlier in your script. You should check the returned handle value - if it is less than 16 then an error has occurred, otherwise it is a valid DLL handle.

At the moment, error codes are defined as follows, however you should treat any return value less than 16 as an error. :-

0 = DLL file not found.
1 = File is not a DLL
2 = Insufficient memory to load DLL.
3 = Too many active DLLs
4 = DLL failed to initialise correctly.
5 = same as 4.

Odyssey Script Commands

SendMessage

```
FUNC SendMessage(dll_handle: Number;
                             Command:String;
                             VAR str_arg:String;
                             num_arg:Number
                             ):Number;
```

This is how you send messages to a DLL. The "dll_handle" argument identifies the DLL which is to receive the message. The "Command" argument is a string which names the command to be performed. The "str_arg" parameter is a string argument for the DLL command (note that it must be a variable), and num_arg is a numeric argument for the DLL command. The interpretation of the string and numeric command arguments (or even if they are used) will depend on the command issued, and will be documented along with the command. Unless otherwise noted in the command description, the SendMessage() result code should generally be interpreted as 0=success, while <>0 indicates an error.

Examples:

```
    SendMessage(fax, "PRINT", "myfax.fax 1-99", 0);
    SendMessage(fax, "SET-DETAIL", "", 1);
```

Odyssey Script Commands

UnloadDLL

FUNC UnloadDLL(dll_handle: Number):Number;

This function is the inverse of LoadDLL() - it causes a DLL to be discarded from memory, after which you should not send it any further messages. The return value is 0 if the call was successful, 1 if the handle was invalid (you should however treat any non zero result as an error code).

For example, to unload the FAX server DLL, the command should be:-

```
            fax_dll := UnloadDLL(fax_dll);
```

This form of the call is recommended, since it has the effect of invalidating the DLL handle variable in your script, as well as just unloading the DLL. This means that the Odyssey internal code will safely intercept any further attempts to use the handle.

Odyssey Script Commands

Windows API Access Commands

The topics below describe the Windows API access commands provided by the Odyssey script language.

Odyssey Script Commands

Introduction to Windows API Support

This section of the Odyssey Script Language Help system outlines how an experienced Windows programmer may access Windows™ API features from the Odyssey script language. This document does *not* attempt the huge task of documenting the Windows API. Interested novices should look in any good bookshop for the Windows SDK programming manuals, published by Microsoft Press.

The Windows Odyssey script language has been extended (relative to the DOS version of Odyssey) in order to allow script authors to create and use Windows style dialog boxes, menus, toolbars and status lines. For experienced programmers, Odyssey provides a mechanism whereby an Ody script can call any function exported by a Windows DLL.

Odyssey script is not an application level programming language, so it is not possible for a script to animate a dialog in *precisely* the same manner one would use in a standard Windows application written in C or Pascal, we do however come pretty close. Most of the differencies lie in how a dialog is loaded and initialized. The main differences are:-

- A true Windows application generally binds resources (definitions of dialog box templates, menu layouts etc), into the executable file of the application. Odyssey scripts are not executable files as far as Windows is concerned, so this step is not possible. In order to use such resources a programmer must create the various dialog and menu templates, and then bind them into a resource-only DLL. The Odyssey script must load that DLL using a call to the new script function LoadResourceLibrary() before it can animate any resource in that library. If a dialog box uses any custom controls then the custom control library will also have to be loaded by the script. A script should unload any resource libraries it has used by calling FreeResourceLibrary() before the script terminates. A demonstration resource-only DLL called DEMODLG.DLL is provided with Odyssey, along with a demonstration script called TESTDLG.SRC which shows how to access and use those resources.

- You cannot write any kind of callback function using the Odyssey script language (particularly: you can't write a dialog callback function). Instead, in the case of dialogs Odyssey provides the required callback function internally, which provides a buffer for API messages between Windows and the script. Scripts retrieve the next buffered API message by calling the new GetDialogMessage() function.

  Note that since Windows API messages are buffered (ie. not received by the script at the time they were sent), that a script cannot handle any message which receives data passed as a pointer in one of the message arguments, eg. you can't use data objects pointed to the lParam argument. A script can however *send* messages which pass pointers to data.

An Odyssey script which wants to have the Windows "look and feel" should create a dialog which will form the core of the user interaction with the script. This then allows the script to make use of script language features which make it possible to associate menus, toolbars and status lines with that dialog (ie. associated objects which are displayed whenever the dialog is the active window), allowing the script to almost completely supplant the normal Odyssey user interface.

Note that Odyssey does not currently provide utilities for actually creating the resources mentioned - these must be created using third party tools, eg. any standard Windows programming language from one of the usual vendors (Microsoft, Borland etc).

Borlands ***Resource Workshop*** utility is particularly useful for this, since it is capable of reading, modifying and writing resources directly to and from a DLL - a script author could thus use the supplied DEMODLG.DLL as a skeleton to be modified to his requirements, using Resource Workshop. Without Resource Workshop (or another resource editor with similar features) the script programmer would have to mess with resource compilers, DLL stub code sections, linkers and so forth.

A typical step-by-step process for a script which uses Windows resources would be :-

1. Load the resource library(s) needed.
2. Load the dialog
3. Load any menus/toolbars/status lines required, and assign them to the dialog.
4. Animate the dialog until the user clicks a close button.
5. Call EndDialog() and wait until the dialog is closed.
6. Discard the menus/toolbars etc which were loaded.
7. Discard the resource library(s) which were used.

The demo script TESTDLG.SRC shows how to carry out all of the above steps.

The script using the new WinAPI features should include the file "windows.inc" so that it has access to the message constants used in the examples below, ie. the script should have the line :-

#include "windows.inc"

near the beginning of the script source. Note that "windows.inc" is not a complete rendition of the "windows.h" shipped with most C compilers. Instead, we have included what we think are the most useful constant definitions of those which a script is likely to need. If however you discover a need for a constant which is missing from this file, then by all means feel free to add that constant definition yourself. Or perhaps even better - create an extention include file with your extra constants, to avoid our "windows.inc" becoming as unmanageably large as the standard version.

Odyssey Script Commands

Accessing any function in a DLL

This section describes a new script language mechanism which allows an experienced programmer to call any function exported by a DLL. This includes standard Windows DLLs, or any function in a DLL available to the script programmer. This gives the script programmer access to the complete Windows API, and it also opens the possibility of sharing the work of a script between the script and a user-written DLL, where the DLL handles the complex and/or time consuming part of the work.

All you need to do to access any DLL function is declare a "prototype" for that function in the script. This feature puts tremendous power in the hands of the script programmer, but also tremendous responsibility. Calling the wrong function, or the right function with the wrong arguments can quite easily cause a global protection fault, or hang the system.

All you need to know in order to declare the prototype is the module name or file name of the DLL that exports the function, the name or ordinal number of the function, and the arguments that the function expects.

The Function Prototype

Here is a sample function prototype; this one provides access to the Windows API function GetSystemDirectory() :-

```
VAR hKernel:Number;

FUNC GetSystemDirectory(VAR sysdir:String; cbMax:Number):Number
  = CALLDLL[hKernel];
```

In other words, the FUNC line is a standard function header which you might see prior to the body of any user written script function. However, instead of a function body we have the "= CALLDLL[module_handle]" directive, which tells the Odyssey script processor that this is really a DLL function, and also says where to find it (in the DLL whose module handle is stored in the 'hKernel' integer variable). Once this prototype is declared, the function may be called just like any other script function, provided that the hKernel variable is initialized before the first call. A number of such function prototypes could very usefully be packed together in an include file (however: let us please *not* go down the road of everything-plus-the-kitchen-sink being declared in a single include file!).

The Module Handle

The module handle (eg. hKernel) must be a global number variable, and it must be initialized before you can call any DLL function which references that module. The initialization can take two forms, eg :-

```
hKernel := GetModuleHandle("KERNEL");
```

or :-

```
hKernel := LoadResourceLibrary(
          "C:\WINDOWS\SYSTEM\KRNL386.EXE");
```

You should use the first initialization method when you know that the DLL is already resident in memory (eg. because it is one of the standard Windows system DLLs such as "KERNEL", "GDI" or "USER").

You use the second initialization method when you think the DLL *may* not be in memory already (don't worry about the DLL being loaded twice: Windows itself prevents that happening, and merely increments a usage count on the second or later load request). If you load a module into memory using LoadResourceLibrary() then you should remember to call FreeResourceLibrary() on that module handle before the script terminates (this simply decrements the usage count if the module was already loaded).

Note: You only need to initialize a module handle once, regardless of the number of functions which reference it. Also, note that the module handle is the same as the 'hRes' handle we have previously been using to access resources in a DLL.

A Sample Script

The following is a more complete version of the above example, showing both the module-handle initialization, and the function call.

```
SCRIPT DLLCallDemo;

VAR hKernel:Number;

(*..........................................................*)

FUNC GetSystemDirectory(VAR sysdir:String; cbMax:Number):Number
  = CALLDLL[hKernel];

(*..........................................................*)

VAR SysDir:String;

BEGIN
     hKernel := GetModuleHandle("KERNEL"); (* init module handle *)

     GetSystemDirectory(SysDir, 80);          (* call function! *)
     Write('Windows system directory is "',SysDir,'".|');
END;
```

CALLDLL Directive with an 'Alias'

The preceding examples used the simplest form of the CALLDLL directive, which will also be the most common form. However, there are certain variants of the directive which you may need to use in unusual circumstances.

If you want the name of the function, as known to the script processor, to be different from the name of the function as known to Windows, then you can use the following version of the CALLDLL directive :-

```
FUNC GetSysDir(VAR sysdir:String; cbMax:Number):Number
  = CALLDLL[hKernel,"GETSYSTEMDIRECTORY"];
```

as you can see, there is an optional second argument to the CALLDLL directive which supplies the name of the function as it is known to Windows. Your script then uses calls to "GetSysDir" throughout, but the script processor translates these into calls to GETSYSTEMDIRECTORY. Essentially, 'GetSysDir' is an alias for 'GETSYSTEMDIRECTORY'. You might use this form of the directive if, for example, the name of the DLL function clashes with the name of an existing Odyssey script command. Note that if you *don't* include this alias directive then Odyssey assumes that the DLL function has the same name that you have declared in the FUNC line (except that Ody forces the name to all upper case before passing it to Windows, which is compliant with the 'pascal' calling convention typically used by an exported DLL function).

CALLDLL Directive with Function Ordinal Number

One important fact not mentioned so far is that if you use the standard or 'alias' forms of the CALLDLL directive, then Odyssey assumes that the function involved has been exported by name from the DLL - with the name being either the name of the Odyssey function, or the alias name. However, if the function is exported by ordinal number, or you only know the ordinal number and not the name, then you can use the following variant of the CALLDLL directive :-

```
FUNC GetSystemDirectory(VAR sysdir:String; cbMax:Number):Number
  = CALLDLL[hKernel,135];
```

In other words, the syntax is just like the 'alias' variant, but uses an integer constant (the ordinal) instead of the function name. Using the ordinal value may also be more efficient than retrieving the function by name on every call. The SDK tool EXEHDR can be used to obtain the ordinal numbers of exported functions in a DLL.

Calling DLL Functions using the C Calling Convention

Most exported DLL functions (including all but one Windows API function), use the 'Pascal calling convention'. This means that the calling program pushes function arguments on the stack from left to right, and the *called* function has responsibility for popping those arguments off the stack.

However, some DLL functions (such as the Windows API function "wsprintf"), use the C calling convention, in which the caller pushes the arguments in order from right to left, and is responsible for cleaning up the stack afterwords.

If you want to call a "C calling convention" function from an Odyssey script, just declare the prototype using the CALLDLL_C directive instead of the CALLDLL directive. Apart from the name change, the CALLDLL_C directive has identical syntax (including options). For example :-

```
FUNC wsprintf2(VAR s:String; szFormat:String; a,b:Number)
  = CALLDLL_C[hUser,"_WSPRINTF"];
```

Note that the 'alias' directive must be used if the function name needs a preceding underscore. Note also that although you can call a "c calling convention" function from an Odyssey script, you cannot get a function prototype to accept a variable number of arguments, or to accept arguments of unknown type.

Limitations of the CALLDLL Mechanism

Although you can access any DLL function, you naturally cannot use any language features which the script language does not support. In particular, if the DLL function requires you to install a callback function then you are stuck: the script language does not and cannot create a callback function (you could however write your own service DLL and declare the callback there...).

Another problem is DLL functions in which one or more of the arguments are pointers to a array or structure. The Ody script language does not currently provide array or structured types. However, you can often get around this limitation by declaring a sequence of variables in which the sequence has the same format as the structure you wish to emulate, and then you pass the address of the first variable in the sequence (or simply declare the argument as VAR), in leu of the structure address. For example, if for some reason you wanted to call the "User" function **GetClientRect()** - which expects an LPRECT argument - you could simulate it as in the following example :-

```
SCRIPT DLLCallDemo;

VAR hUser:Number;

(*..........................................................*)

PROC GetClientRect(hWnd:Number; VAR dummy:Number)
        = CALLDLL[hUser];

(*..........................................................*)

VAR left,top,right,bottom:Number; (* RECT structure *)

BEGIN
    hUser := GetModuleHandle("USER"); (* init module handle *)

    GetClientRect(GetOdyWindow(), left);
    Write("Ody client window width   = ",right,".|");
    Write("Ody client window height = ",bottom,".|");
END;
```

A similar trick can be used to simulate array arguments, if the array doesn't need to be too large.

Odyssey Script Commands

LoadResourceLibrary, FreeResourceLibrary

FUNC LoadResourceLibrary(module_name:String):Number;
PROC FreeResourceLibrary(hRes:Number);

These functions allow a script to load and unload resource DLLs. A handle to a resource DLL is required before a script can load dialog, menu and toolbar templates.

The **LoadResourceLibrary()** function attempts to find the "resource only DLL" whose name is passed in <module_name>. If the DLL is found and successfully loaded then the function returns a handle (similar to a file handle), which is used to identify the resource library when you need to extract an individual resource (eg. when you call the LoadDialog() function). If the return value from **LoadResourceLibrary()** is less than 32 then an error has occurred, and the return value is the error code (this is the same error code returned by the LoadLibrary() standard Windows API function).

The **FreeResourceLibrary()** function discards a resource DLL identified by *hRes*, which is the handle returned by a previous call to **LoadResourceLibrary()**. It is good practice for a script to explicitly unload any DLLs it loads, although Odyssey does in fact unload such DLLs automatically when the script terminates.

Examples:
```
   hRes := LoadResourceLibrary("c:\winody\demodlg.dll");
   .....
   FreeResourceLibrary(hRes)
```

Odyssey Script Commands

LoadDialog

**FUNC LoadDialog(hRes:Number; szDlgName:String;
x,y:Number):Number;**

This function is used to load a dialog template, and is described in detail below. Note that menus, toolbars etc are associated with a particular dialog handle, hence you must create a dialog before you can load a menu, toolbar or status line. Also, you must have obtained a resource library handle before you can read a dialog from that library (see LoadResourceLibrary()).

Note that script dialogs (except for MessageBox dialogs) are always modeless, ie. top level menus and toolbar buttons are still active when the dialog is active. This means that you must ensure that WM_COMMAND identifiers and accelerator keystrokes used by menus and dialog controls do not clash (script dialogs must be modeless since Odyssey must be able to run other tasks within Odyssey while a script is running, which is not possible with a modal dialog).

The **LoadDialog()** function attempts to read a dialog template with the name <szDlgName> from the resource DLL identified by <hRes>, which is the handle returned by a previous call to LoadResourceLibrary. If successful, the function creates and displays the dialog at coordinates (x,y) pixels relative to the Odyssey MDI desktop origin, and returns a dialog (window) handle. The script should immediately enter its GetDialogMessage() loop to retrieve the **WM_INITDIALOG** message which will already have been placed in the dialog message queue.

The function returns 0 if the resource handle was invalid, or if the dialog name did not match any dialog template stored in the resource DLL.

The script dialog manager checks the x,y coordinates passed. If x or y are negative, then the dialog is displayed at a default x or y position on the screen. If x or y are so large that the dialog would be completely or partially off the screen, then the dialog manager adjusts the x or y coord so that the right or bottom edge of the dialog is aligned with the right or bottom of the Odyssey desktop area.

Example:
```
   hDlg := LoadDialog(hRes,"TESTDLG",-1,-1);
```

Odyssey Script Commands

GetDialogMessage

FUNC GetDialogMessage(
hDlg:Number;
VAR iMsg,wParam:Number;
VAR lParam:Long):Flag;

This function is used to wait for messages being sent to the dialog identified by the <hDlg> handle. **GetDialogMessage()** is intended to be called from within a loop, as in the example shown below. If no message is available for the given dialog then the script goes to sleep until one becomes available. The function returns FALSE when the dialog is closed, ie. after EndDialog() is called.

Example:

```
WHILE GetDialogMessage(hDlg,iMsg,wParam,lParam) DO
    CASE iMsg OF
    | WM_INITDIALOG:
        SetDlgItemText(hDlg,IDD_TEXT0,myText);
        (* plus any other initialization required. *)
    | WM_COMMAND:
        CASE wParam OF
        | IDD_MYBUTTON:
            DoNestedDialog();
        | IDOK:
            EndDialog(hDlg, 1);
        | IDCANCEL:
            EndDialog(hDlg, 0);
        END;
    END;
END;
RETURN wParam;
```

Odyssey Script Commands

EndDialog

PROC EndDialog(hDlg, retValue:Number);

This procedure is very similar to the Windows API function of the same name. It does not cause the dialog <hDlg> to be destroyed immediately, instead it simply sets a flag which Odyssey sees the next time GetDialogMessage() is called, at which point the dialog is destroyed, and the **GetDialogMessage()** function returns FALSE, ending the dialog loop.

The value retValue is copied to the *wParam* argument of the final **GetDialogMessage()** call. This allows the dialog loop to pass a result back up to the outer function (note the RETURN line in the example given).

See GetDialogMessage() for an example of this command in use.

Odyssey Script Commands

GetDlgItemInt

FUNC GetDlgItemInt(hDlg,idControl:Number;
VAR x:Number):Flag;

This function is much like the Windows API function of the same name, but the arguments are organised in what is hopefully a slightly more convenient arrangement (particularly: no need to declare an 'lpTranslated' boolean variable in order to receive the success/failure indication).

Given a dialog window handle <hDlg> returned by a previous call to LoadDialog(), and the ID number of a control in that dialog <idControl>, this function reads the text in an edit control, translates it into an integer, and returns the integer in the <x> variable. The function returns TRUE on success, or FALSE if the edit control string contained illegal characters.

Example:

```
IF GetDlgItemInt(hDlg, IDD_NUMCTRL, x) THEN
  Write("x = ",x,"|");
```

Odyssey Script Commands

GetDlgItemText

FUNC GetDlgItemText(hDlg,idControl:Number;
VAR s:String):Number;

This function is much like the Windows API function of the same name, but the arguments are organised in what is hopefully a slightly more convenient manner (in particular: the standard API function has an extra parameter giving the length of the destination string, which we don't need because the script compiler automatically handles string overflow checks).

Given a dialog window handle <hDlg> returned by a previous call to LoadDialog(), and the ID number of a control in that dialog <idControl>, this function reads the text in an edit control, and returns it in the <s> variable. The function returns 0 on failure, else it returns the length of the string copied into <s>.

Example:

```
    len := GetDlgItemText(hDlg, IDD_MYTEXTCTRL, s);
```

Odyssey Script Commands

SetDlgFocus

PROC SetDlgFocus(hDlg, idControl:Number);

Given a dialog window handle <hDlg> returned by a previous call to LoadDialog(), and the ID number of a control in that dialog <idControl>, this function sets the current focus to that control. The control with the current focus is the one which receives keyboard input.

Example:

```
        SetDlgFocus(hDlg, IDD_FIRSTCTRL);
```

Odyssey Script Commands

CheckDlgButton

PROC CheckDlgButton(hDlg, idButton, uCheck:Number);

This procedure is used to set the "checked" state of a check button control. <hDlg> is returned by a previous call to LoadDialog(), and identifies the dialog containing the control, <idButton> is the number of that control (assigned when the dialog template was created), and <uCheck> is the intended check state (0=unchecked, 1=checked, 2 is used only for three-state check buttons, and sets the alternate checked state).

Example:

```
        CheckDlgButton(hDlg, IDD_CHECKBOX, TRUE);
```

Odyssey Script Commands

CheckRadioButton

PROC CheckRadioButton(
hDlg, idFirstButton,
idLastButton,
idCheckButton:Number);

This procedure is used to select (check) a single button in a group of radio buttons, while at the same time deselecting all the other buttons in the same group. The button group must have a contiguous range of identifiers. <hDlg> is returned by a previous call to LoadDialog(), and identifies the dialog containing the radio button group. <idFirstButton> is the number identifying the first button in the group, while <idLastButton> identifies the last button in the group. <idCheckButton> identifies the button which is to be selected - that button is checked, and all other buttons in the group are unchecked.

Example:

```
CheckRadioButton(hDlg,IDD_FIRST,IDD_LAST,idButton);
```

Odyssey Script Commands

DlgDirList

```
FUNC DlgDirList(
        hDlg:Number;
        VAR Path:String;
        idListBox, idStaticPath, uFileType:Number
    ):Number;
```

This function is used to fill a listbox with a list of file or directory names. Note that the Windows API assumes that a dialog requiring this function has at least these three elements :-

- The listbox control itself, to display the list of files.
- An "Edit" control, where the user enters the file name he wants to select.
- A "static text" control, where Windows displays the current path (ie. the directory from which the files currently in the listbox are extracted).

The parameters to the **DlgDirList()** function tell the Windows API which files you want to list, and also gives it the IDs of the listbox and static text controls mentioned. The programmer calls **DlgDirList()** to fill the listbox, and then fills in the edit control with a string passed back in the Path variable.

<hDlg> is the return value from a previous call to LoadDialog(), and identifies the dialog box containing the listbox control. <Path> is a string containing a wildcard - these are the files you want listed. <idListBox> is the ID of the listbox control, idStaticPath is the ID of the static text control which will receive the path string, and uFileType is a bit mask which specifies the type of files/directories which are to be listed. The uFileType argument can be a combination (bitwise OR) of the following values declared in "windows.inc" :-

DDL_READWRITE-	List read-write files with no special attributes.
DDL_READONLY-	List read-only files.
DDL_HIDDEN-	List hidden files.
DDL_SYSTEM-	List system files.
DDL_DIRECTORY-	List directories.
DLL_ARCHIVE-	List files which have their "archive" bit set.
DDL_DRIVES-	List drives.
DDL_EXCLUSIVE-	If this option is included then *only* files with the specified attributes are listed, otherwise files with the specified attributes are listed in addition to ordinary read/write files.

It is ***VERY IMPORTANT*** to note that <Path> is a variable! When this string is passed to Windows, it should contain the complete directory path and file specification, eg. "C:\WINODY\DOWNLOAD\*.ZIP". Windows will remove the path part of this specification and copy it to the static text control, returning the tail part of the specification (ie. "*.ZIP") back in the Path variable, which the programmer then assigns to the edit control.

The return value from **DlgDirList()** is non-zero if the function was successful, otherwise it is zero.

Odyssey Script Commands

DlgDirListComboBox

FUNC DlgDirListComboBox(
hDlg:Number;
VAR Path:String;
idComboBox, idStaticPath, uFileType:Number
):Number;

This function is used to fill a combo-listbox with a list of file or directory names, otherwise this function is identical to DlgDirList(). Note that the Windows API assumes that a dialog requiring this function has at least these three elements :-

- The combobox control itself, to display the list of files.
- An "Edit" control, where the user enters the file name he wants to select.
- A "static text" control, where Windows displays the current path (ie. the directory from which the files currently in the combobox are extracted).

The parameters to the **DlgDirListComboBox()** function tell the Windows API which files you want to list, and also gives it the IDs of the combobox and static text controls mentioned. The programmer calls **DlgDirListComboBox()** to fill the listbox of the combobox control, and then fills in the edit control with a string passed back in the Path variable.

<hDlg> is the return value from a previous call to LoadDialog(), and identifies the dialog box containing the combobox control. <Path> is a string containing a wildcard - these are the files you want listed. <idComboBox> is the ID of the combobox control, idStaticPath is the ID of the static text control which will receive the path string, and uFileType is a bit mask which specifies the type of files/directories which are to be listed. The uFileType argument can be a combination (bitwise OR) of the following values declared in "windows.inc" :-

DDL_READWRITE-	List read-write files with no special attributes.
DDL_READONLY-	List read-only files.
DDL_HIDDEN-	List hidden files.
DDL_SYSTEM-	List system files.
DDL_DIRECTORY-	List directories.
DLL_ARCHIVE-	List files which have their "archive" bit set.
DDL_DRIVES-	List drives.
DDL_EXCLUSIVE-	If this option is included then *only* files with the specified attributes are listed, otherwise files with the specified attributes are listed in addition to ordinary read/write files.

It is ***VERY IMPORTANT*** to note that <Path> is a variable! When this string is passed to Windows, it should contain the complete directory path and file specification, eg. "C:\WINODY\DOWNLOAD\*.ZIP". Windows will remove the path part of this specification and copy it to the static text control, returning the tail part of the specification (ie. "*.ZIP") back in the Path variable, which the programmer then assigns to the edit control.

The return value from **DlgDirListComboBox()** is non-zero if the function was successful, otherwise it is zero.

Odyssey Script Commands

GetDlgCtrlID

FUNC GetDlgCtrlID(hWnd:Number):Number;

Given the handle of a child window (eg. a control in a dialog box), returns the ID number of that control. The function returns 0 if the window handle was invalid.

Example:

```
    idCtrl := GetDlgCtrlID(hWndCtrl);
```

Odyssey Script Commands

GetDlgItem

FUNC GetDlgItem(hDlg,idControl:Number):Number;

Given a dialog window handle <hDlg> returned by a previous call to LoadDialog(), and the ID number of a control in that dialog <idControl>, returns the window handle of that control. The function returns 0 if the dialog handle or control ID was invalid.

Although intended for use with dialogs, this function actually works with any window which owns child windows - provided you have the handle of the parent window, and the ID of a child window, you can obtain the window handle of the child (the TESTDLG.SRC demo script uses this trick to get the handle of Odyssey's MDI client window).

Example:

```
     hWndCtrl := GetDlgItem(hDlg, IDD_MYBUTTON);
```

Odyssey Script Commands

IsDlgButtonChecked

FUNC IsDlgButtonChecked(hDlg,idControl:Number):Flag;

This function returns TRUE if the button control with ID <idControl> in the dialog <hDlg> is selected (checked). Otherwise it returns FALSE.

Example:
```
  IF IsDlgButtonChecked(hDlg, IDD_BTN) THEN
     ....
```

Odyssey Script Commands

SendDlgItemMessage

```
FUNC SendDlgItemMessage (
        hDlg, idControl, iMsg, wParam:Number;
        lParam:Long
   ):Long;
```

This function is used to send Windows API messages to a child control in a dialog. The function is actually a shortcut for the following :-

```
    WinSendMessage(GetDlgItem(hDlg,idControl),
                                      iMsg,wParam,lParam);
```

In other words, it is precisely the same as WinSendMessage(), except that it is specialised towards sending messages to children of a known parent window, ie. controls in a dialog.

SetDlgItemInt

```
PROC SetDlgItemInt (
        hDlg, idControl, uValue:Number;
        fSigned:Flag
    );
```

Given a dialog window handle <hDlg>, and the ID number of a control in that dialog <idControl>, this function sets the text of that control to the string representation of the value <uValue>. The <fSigned> flag indicates whether <uValue> is to be treated as signed or unsigned.

Odyssey Script Commands

SetDlgItemText

```
PROC SetDlgItemText(
        hDlg, idControl:Number;
        s:String
    );
```

Given a dialog window handle <hDlg>, and the ID number of a control in that dialog <idControl>, this function sets the text of that control to the contents of the string <s>.

Odyssey Script Commands

MessageBox

FUNC MessageBox (
hWndParent:Number;
text, title:String;
style:Number
):Number;

This function allows access to the Windows API function **MessageBox()**. <hWndParent> is the handle of the window which should inherit the focus when the message box dialog closes, <text> is the message to display in the message box, and <title> is the caption to give to the message box.

The <style> value controls what icon and buttons will appear in the message box. Various constants are declared in "windows.inc" which you should 'OR' together to give the message box the appearance you want it to have. The constants are described below :-

- Include one of the these constants to select the icon which will appear in the message box - **MB_ICONEXCLAMATION**, **MB_ICONINFORMATION**, **MB_ICONQUESTION**, **MB_ICONSTOP**.
- Include one of these constants to select the range of buttons you want to have in the message box - **MB_ABORTRETRYIGNORE**, **MB_OK**, **MB_OKCANCEL**, **MB_RETRYCANCEL**, **MB_YESNO**, **MB_YESNOCANCEL**
- Include one of these buttons to override which button is initially highlighted - **MB_DEFBUTTON1**, **MB_DEFBUTTON2**, **MB_DEFBUTTON3**. The default is **MB_DEFBUTTON1**.

The function return value indicates which button the user clicked in order to close the dialog. The value is either **IDABORT**, **IDCANCEL**, **IDIGNORE**, **IDNO**, **IDOK**, **IDRETRY** or **IDYES**, depending on which button types were enabled, and which one was clicked.

All of the constants mentioned are declared in "**windows.inc**", which you must include in any script that uses them.

Example:

```
IF MessageBox(GetFocus(),
     "Try that again?", "Demo",
     MB_ICONQUESTION+MB_YESNO)=IDYES THEN
  /* do whatever it was again */
END;
```

Odyssey Script Commands

LoadMenu

```
FUNC LoadMenu (
        hRes:Number;
        szMenuName:String
    ):Number;
```

This function is used to load a Windows menu template from the resource only DLL identified by the <hRes> argument, which should be the return value from a previous call to LoadResourceLibrary(). <szMenuName> is the name of the menu resource in the resource library. Note that the menu is only loaded, not activated.

The function result is a handle which should be used to identify the menu in calls to DestroyMenu() and AssignMenu(), or 0 if the resource handle or menu name was invalid.

Please note that items in menus loaded by a script *MUST* use menu command identifiers in the range 1000-1199 so that Odyssey can distinguish messages coming from a script menu from those coming from its own menus. Menu **WM_COMMAND** messages with ID values in the stated range are forwarded to the script - other values are not.

Also, since Odyssey is an MDI application, all top level menus, including menus loaded by a script, *MUST* contain a *Window* submenu. The Window submenu typically contains commands for tiling and cascading MDI children, and for arranging iconic children. The demo script TESTDLG.SRC shows how to implement these Window menu functions. Odyssey also adds MDI child titles to the Window menu so that users can select document windows from the menu, which is why the Window submenu must exist.

A script which uses this menu function must not also use the old script language Menu() command, as the two features are not compatible.

Odyssey Script Commands

DestroyMenu

PROC DestroyMenu(hMenu:Number);

Destroys the menu identified by the menu handle, which was returned by a previous call to LoadMenu(). If **DestroyMenu()** destroys the currently active menu, then Odyssey will switch to the standard terminal window menu before destroying the script menu.

Odyssey Script Commands

AssignMenu

```
PROC AssignMenu (
        hDlg, hMenu, iWindowPopup:Number
    );
```

AssignMenu() creates an association between the dialog window <hDlg> and the menu handle <hMenu>. In other words, that menu will be displayed whenever the associated dialog window is the active window. <iWindowPopup> is the index of the Window menu in the scripts menu template, where 0 is the index of the first submenu (typically the File submenu), 1 is the index of the second submenu (typically the Edit submenu) and so on.

<hMenu> may be 0, in which case Odyssey reverts to displaying the menu which is normally displayed when the terminal window is active. The value of <iWindowPopup> is ignored in this case.

Note: A menu object may be assigned to more than one dialog window, and therefore is not automatically destroyed when the dialog is destroyed. Menu objects loaded by a script *are* destroyed automatically when the script which loaded it terminates, however you should consider it good practice to destroy menus explicitly, within the script.

Odyssey Script Commands

CreateToolBar

```
FUNC CreateToolBar (
        hRes:Number;
        szBitmapName:String;
        nButtons:Number;
        ButtonIDs:String
    ):Number;
```

This function creates a toolbar object which is derived from a bitmap resource read from the resource-only DLL identified by <hRes>, which should be the return value from a previous call to LoadResourceLibrary(). <szBitmapName> is the name of the bitmap resource.

Odyssey creates a toolbar containing <nButtons> buttons, and paints the surface of each button with a different "mini-icon" taken from a portion of the bitmap. The bitmap should consist of a grid of these mini-icons, each of which should be 19 pixels wide by 17 deep, plus a one pixel border to the right and bottom of each icon. The border should normally be white, however if you make the right-border of an icon light red (RGB(255,0,0) - palette index 9 on the standard system palette), then Odyssey will insert a gap between this button and the next - allowing the script to visually separate groups of related icons.

There should be a maximum of 5 icons in one bitmap row (ie. the bitmap is 100 pixels wide), though you may have any number of icon rows. Icons are extracted from the bitmap in order from left to right, top to bottom and each is then assigned in turn to a toolbar button. The bitmap should be 16-color, and should use only the standard system colors.

Note: The demonstration resource DLL "DEMODLG.DLL" contains a sample toolbar source bitmap which might give you a clearer idea of how the bitmap should be arranged.

A script dialog handler function is notified of a toolbar button press via a **WM_COMMAND** message, with a button ID in the wParam argument. The wParam ID of each button is set using the <ButtonIDs> string argument shown above. The string should look something like this :-

"1001[help text1],1002[help text 2],1003[help text 3]"

ie. the string should contain <nButtons> button IDs, each separated by a comma. Note that the button ID's should be distinct, ie. there should be no controls in the dialog or in a menu with the same ID as a toolbar button, unless the toolbar button is intended to have the same meaning as the dialog button or menu item. The text inside the square brackets (if present) is used to implement the "tool tips" feature, whereby if you leave the mouse cursor over a toolbar icon, a little yellow window appears containing a brief description of the purpose of the icon (a couple of words at most please). The latter is not intended to provide exhaustive help information, it exists solely to provide a text alternative to the icon, if the user doesn't immediately understand the purpose of the icon.

Note that CreateToolBar() only loads the toolbar into memory. The toolbar is not activated until it is assigned to a dialog window with AssignToolBar().

The function result is a handle which should be used to identify the toolbar in calls to DestroyToolBar() and **AssignToolBar()**. The function result is zero if the <hRes> argument was invalid, or if the named bitmap was not found.

Odyssey Script Commands

AssignToolBar

```
PROC AssignToolBar (
        hDlg, hToolBar:Number
    );
```

AssignToolBar() creates an association between the dialog window <hDlg> and the toolbar handle <hToolBar>. In other words, that toolbar will be displayed whenever the associated dialog window is the active window.

<hToolBar> may be 0, in which case Odyssey reverts to displaying the toolbar which is displayed when the terminal window is active.

Note: A toolbar object may be assigned to more than one dialog window, and therefore is not automatically destroyed when the dialog is destroyed. Toolbars created by a script *are* destroyed automatically when the script that created it terminates, however you should consider it good practice to destroy toolbars explicitly, within the script.

Odyssey Script Commands

DestroyToolBar

PROC DestroyToolBar(hToolbar:Number);

Frees the memory associated with a toolbar object created by CreateToolBar(). The toolbar handle is invalid after this call.

Note: A toolbar object may be assigned to more than one dialog window, and therefore is not automatically destroyed when the dialog is destroyed. Toolbars created by a script *are* destroyed automatically when the script that created it terminates, however you should consider it good practice to destroy toolbars explicitly, within the script.

Odyssey Script Commands

AssignStatus

```
FUNC AssignStatus (
        hDlg:Number;
        StDefStr:String
     ):Number;
```

Function **AssignStatus()** is very similar to the old script language command SetHelp() (which is still supported). The difference is that whereas the old status lines were a single 'stack' shared by all windows in Odyssey, this new feature associates an independant stack of status lines with a particular window. Whenever that window is active, the status line most recently added to its stack is displayed. This function also allows a script to make use of the 'Windows Style' status lines which are split into fields which can be independantly updated.

AssignStatus() creates a status line and assigns it to dialog <hDlg>, meaning that this status line is displayed whenever <hDlg> is the active window, unless deleted or superceded by a later status line. Script dialogs which don't call **AssignStatus()**, or which delete all the status lines they've assigned, share the status line that the Ody terminal window uses.

<StDefStr> is a 'picture string' which defines how the status line will look. It should contain one or more fields of the form '**[nn]ttttt**' where 'nn' is the width of a status line cell in units of average char widths, and 'ttttt' is the text which will initially appear in that cell. The text may be omitted if the cell should initially be empty. The final cell on the status line may have a width of zero, which means that the cell occupies the remainder of the width of the status line. If the status line contains no '[width]' fields, then StDefStr is assumed to contain the text of a status line consisting of a single, full width cell. The function returns a handle for the new status line, which should be used in calls to SetStatusField().

Like **SetHelp()**, if StDefStr is an empty string then the status line most recently assigned to the dialog is destroyed, in which case the function returns the handle of the status line which was previously on top of the stack, or 0 if all status lines assigned to the dialog have been destroyed. Again like **SetHelp()**, groups of characters in the text of a status cell may be surrounded by curly brackets, {like this}, in which case the enclosed characters are highlighted, ie. painted in a different color.

Example:

```
   AssignStatus(hDlg, "[20]One[0]Two");
```

The above example sets a status line with two cells, the first of which is 20 AvCharWidths wide, with the second cell occupying the remainder of the width of the status line. The text in the two cells is 'One' and 'Two'.

Odyssey Script Commands

SetStatusField

```
PROC SetStatusField (
        hStatus:Number;
        nField:Number;
        szText:String
    );
```

This command is used to update a single cell in a status line. This procedure can be called at any time, whether or not the dialog which owns the status line is the active window, and regardless of whether hStatus is the current status line displayed for that dialog.

<hStatus> identifies the status line to be updated, and should be the return value from a previous call to AssignStatus(). <nField> is the number of the status field (cell) to update, 0 being the number of the first cell. <szText> is the text to display in that cell.

☞ Odyssey does very little parameter validation on the hStatus handle you pass to this command, and might crash if you call this command with an invalid hStatus, eg. when hStatus is not a status line handle, or when hStatus has already been deleted.

A status line is automatically deleted when the window which owns it is destroyed.

Odyssey Script Commands

WinHelp

```
FUNC WinHelp (
        HelpFn:String;
        fuCommand:Number;
        dwData:Long
    ):Flag;
```

This function is identical to the Windows API function WinHelp(), except that the first argument (the main application Window handle) is missing from the Ody script version. The latter is supplied by Odyssey when it forwards this call to the Windows API. The function returns TRUE on success.

An Odyssey script language programmer must use third party tools to create the Windows .HLP file which **WinHelp()** attempts to read.

Odyssey Script Commands

FindWindow

```
FUNC FindWindow (
        szClassName, szCaption:String
    ):Number;
```

This function is identical to the Windows API function of the same name, except that you cannot pass NULL as an argument (however, passing an empty string "" provides equivalent functionality).

FindWindow() searches for a top level Window which has the given class name or caption, returning a window handle, or 0 if the window was not found. One of the arguments may be an empty string, meaning that any window matching only the other argument will be accepted. This function is intended to be used to find the main Window of other applications, so that a script can post messages to it. You could also use this function to find the handle of the Odyssey window, but that kind of nasty trickery is not encouraged! This function does not search through child windows.

Odyssey Script Commands

SetWindowText

PROC SetWindowText(hWnd:Number; s:String);

Sets the caption of the window identified by the handle <hWnd> to the text contained in string variable <s>. If hWnd identifies a child window such as a static text or edit control then SetWindowText sets the text in the control.

Odyssey Script Commands

MessageBeep

PROC MessageBeep(uAlert:Number);

Call this procedure to sound a beep note on the PC speaker. The <uAlert> argument can be 0, or one of the **MB_ICON...** constants defined for the MessageBox() function - different alert numbers supposedly sound different notes, but Windows doesn't implement that feature unless you have a fancy sound board installed, or a non-standard speaker driver.

Odyssey Script Commands

GetFocus

FUNC GetFocus():Number;

Returns the handle of the window which currently owns the input focus.

Odyssey Script Commands

GetOdyWindow

FUNC GetOdyWindow():Number;

Returns the handle of the Odyssey application window. This may be used by scripts which need to post messages to Odyssey.

Odyssey Script Commands

GetModuleHandle

```
FUNC GetModuleHandle (
        modname:String
    ):Number;
```

Returns the module handle of a module which Windows already knows about (eg. "GDI"). Note that <modname> is a runtime module name and *not* a file name. The function result is the HMODULE of the module, or zero if Windows could not find the module in memory.

Odyssey Script Commands

WinSendMessage

```
FUNC WinSendMessage (
        hWnd, iMsg, wParam:Number;
        lParam:Long
    ):Long;
```

This function is identical to the Windows API function **SendMessage()**, but has a different name because the Odyssey script language already has a built in command called SendMessage().

Odyssey Script Commands

WinPostMessage

```
PROC WinPostMessage (
        hWnd, iMsg, wParam:Number;
        lParam:Long
    );
```

This function is identical to the Windows API function **PostMessage()**, but has a different name for the sake of symmetry with WinSendMessage().

Odyssey Script Commands

Miscellaneous Commands

The script commands listed below do not fit well into any specific category.

Odyssey Script Commands

Address

FUNC Address(VAR obj:AnyType):Long;

Many Windows API messages expect you to pass a far pointer to a variable (usually a string) in the lParam argument of SendMessage() or PostMessage(). This would previously have been impossible to do using Odyssey script language, which provided no method of determining the address of a variable. The **Address()** function fixes that problem. Note that the return value is a long, so may be used directly in the Ody script language equivalent of the message passing function - WinSendMessage() or WinPostMessage(). Please note that messing with the address value could cause a Protection Fault in Windows.

Example:

```
s := "A new string for a listbox";
hWndList := GetDlgItem(hDlg, IDD_LISTBOX);
WinSendMessage(hWndList, LB_ADDSTRING, 0, Address(s));
```

Odyssey Script Commands

AllowYield

PROC AllowYield(yieldOK:Flag);

AllowYield() allows the programmer to prevent the script from yielding to other tasks (including other internal Ody tasks and other Windows apps), while a time critical section of script is running.

Background:

Applications running under Windows 3.x are cooperatively multitasked, that is, they must explicity yield CPU time in order to allow other tasks to run. Odyssey normally runs a script as a low priority background task, allowing it to run for a few p-code instructions, and then the script is suspended while other Odyssey windows are updated, and also so that Ody can do its duty to Microsoft, by yielding CPU time to other applications.

Odyssey uses two forms of yield, referred to here as a "short yield" and a "long yield". A short yield is when Odyssey gives some CPU time to Windows (calls PeekMessage a few times), but doesn't explicitly suspend the script or give any time to other internal tasks within Odyssey. A long yield is when Odyssey suspends the script, updates other Odyssey windows, checks online clocks etc, *and* gives CPU time to Windows.

Odyssey usually usually yields in these circumstances :-

a) If the script calls a command which involves waiting for some external event, then the script is suspended until that event occurs. Commands in this category are :

Delay-	Script suspended until timer expires.
Edit-	Script suspended until edit window closes.
GetDialogMessage-	Script suspended until dialog msg arrives.
RdKey-	Script suspended until key is pressed while terminal window has the focus.
Sleep-	Script suspended until carrier drops.
Transmit-	Script suspended until transmission is completed (note that this doesn't happen with Paste).
WaitFor-	Script suspended until string arrives, or timeout expires.
WaitForSilence-	Script suspended until required silent period is seen, or timeout expires.
WatchEvent-	Script suspended until one of the listed events is triggered.

b) Odyssey normally executes a "short yield" whenever the script calls one of the built-in script commands (the DOS version of Ody didn't yield at this point - this change was added to give more CPU time to Windows).

c) If Priority() is TRUE, Odyssey does a long yield whenever the script calls function KeyPressed(). If **Priority()** is false, then the script also yields when it calls Received().

Certain situations may arise in which a script programmer wants to prevent Odyssey from yielding for a while, for example :-

i) The script might be adding a lot of entries to a listbox control. If Odyssey yields to Windows every time you call SendDlgItemMessage() then it may take a *long* time to fill that listbox.

ii) If the script is posting messages to another application, it may want to ensure that several messages are posted before it yields control to allow the other application to read the first of those messages from its input queue - especially if each message involves some screen repainting.

All of the above was a preamble to describing the new **AllowYield(flag)** command. If you call **AllowYield(FALSE)** then Odyssey will not do the "short yields" mentioned when you call a built in procedure. This allows you to (for example) update that listbox more quickly in case (i) above. You should (as quickly as possible) perform the high priority task you need to perform, and then call **AllowYield(TRUE)**, to let other tasks get some CPU time. Note that calls to **AllowYield()** may be nested, ie. if you call AllowYield(FALSE) five times, then you must call **AllowYield(TRUE)** five times to enable CPU yields again.

Note that regardless of the state of **AllowYield()**, Odyssey will still suspend the script in the cases listed in (a) above.

: **AllowYield()** gives a script programmer more than enough rope to hang himself - this power must *not* be abused! It is unacceptable for a script to call **AllowYield()** as a matter of course, simply to make the script run faster. Doing so would mean that other applications, and other tasks inside Odyssey, get no CPU time at all; their message queues would undoubtedly fill up, and Windows will certainly crash. You should use **AllowYield(FALSE)** *only* to bracket small sections of time critical code, or where it is important not to yield (as in (ii) above), and remember to call **AllowYield(TRUE)** as soon as possible. Be especially careful not to do an early return from a function which calls **AllowYield(FALSE)**, unless it calls **AllowYield(TRUE)** before the return.

Odyssey Script Commands

ASC

FUNC ASC(s:String; i:Number):Number;

This function returns the ASCII code of the character at position i in the string s. The first character in any string has a position of 0. For example, ASC("ABCDEF",1) should return 66, the ASCII code of the letter 'B'.

Example:

```
    Write("ASCII code of 'C' is ",ASC("C",0));
```

Odyssey Script Commands

CHR

FUNC CHR(n:Number):String;

This function returns a one-character string, the one character being that which has the ASCII code n. For example, CHR(66) should return the string "B". The character need not be printable, but must not be ASCII code zero, as CHR(0) is reserved for use as the string terminator. This function is probably most useful in string expressions, as in the example given below.

Example:

```
    Write(CHR(27)+"[2J"); -- clear VT100 screen.
```

Odyssey Script Commands

Date,Time

FUNC Date():String;
FUNC Time():String;

These commands can be used to obtain the current date and time from Windows. The Date() function returns the current system date as a string formatted as dd-mmm-yyyy, for example "09-Sep-1993". The Time() function returns the current system time as a string, formatted as hh:mm:ss, for example "21:51:09".

Ideas: You can use these commands to time stamp log files, or you could use them to write a script which executes at the same time every day using some code such as:-

```
WHILE Time()<>"18:30:00" DO
   (* wait for right time *)
END;
```

Another hint is that you can convert the time string to a number using StrToInt(), but only if you truncate the string to the first five characters (ie. four digits), otherwise the number will overflow, eg. :-

```
the_time := Time();
the_time := SubStr(the_time,0,5);
time_num := IntToStr(the_time);
```

this is written for clarity, not efficiency. You could actually write it as a one-liner.

Example:

```
 time_num := IntToStr(SubStr(Time(),0,5));
```

Odyssey Script Commands

DEC,INC

PROC DEC(VAR n:Number [; amount:Number]);
PROC INC(VAR n:Number [; amount:Number]);

These commands are provided for incrementing and decrementing numbers. These commands are more efficient and concise than the equivalent assignment statement such as:-

```
    a := a+1;
```

An optional second parameter may be supplied which is the amount the number should be incremented or decremented by. If this parameter is not provided then a default of one is assumed.

Examples:
```
     INC(a);
     DEC(a);
     INC(a,4);
     DEC(a,2);
```

Odyssey Script Commands

Delay

PROC Delay(secs:Number);

This command causes a script to pause for the number of seconds passed in the "secs" parameter. The script resumes when this time has expired.

Example:

```
    Delay(3);
```

Odyssey Script Commands

DTESpeed

FUNC DTESpeed():Number;

This function returns the current DTE speed, ie. the speed at which the terminal is currently communicating with the modem.

Example:

```
    Write("Current baud rate is ",DTESpeed());
```

Odyssey Script Commands

Exit

PROC Exit();

This command allows a script to cancel itself, i.e. the script is terminated and Odyssey returns to terminal mode. This command would typically be used when a fatal error is detected.

Example:

```
IF NOT Dial("ANYBBS") THEN
   Write("Service not available.|");
   EXIT();
END;
```

Odyssey Script Commands

GetEnv

PROC GetEnv(varname:String; VAR value:String):Flag;

This command searches for and returns (if it exists) a DOS environment variable named in the varname parameter. The function returns TRUE if the variable exists, or FALSE if it does not. The value of the variable is returned in the "Value" string. For example:-

```
IF NOT GetEnv("COMSPEC",CmdName) THEN
   Write("Command Processor name not defined!!|");
END;
```

COMSPEC is a standard DOS environment variable found on most DOS machines. It tells DOS (and other programs) where the command interpreter is located. After the call shown above the CmdName variable will most likely contain "C:\COMMAND.COM", although this depends on your system configuration.

GetCallInfo

FUNC GetCallInfo(VAR ServiceName:String;
VAR Connect_Speed:Number;
VAR Key:String):Flag;

A multi-purpose script which makes use of queued dialing will probably want to know which service it succeeded in connecting to. This function allows the script to get that information. The function returns FALSE if no connection was established, otherwise "ServiceName" is the name of the service as it appears in the dialing directory, "Connect_Speed" is the connect speed the modem reported (not necessarily the same as the current DTE speed), and "key" is the directory entry key.

Example:

```
IF Dial("ANYBBS") THEN
   GetCallInfo(Service,Connect,Key);
   Write("Service is:       ",Service,"|");
   Write("Connect speed is: ",Connect,"|");
   Write("Directory key is: ",Key,"|");
END;
```

Odyssey Script Commands

Halt,HaltE

PROC Halt();
PROC HaltE(return_code:Number);

These commands are like EXIT, in that they allow a script to cancel itself, with the exception that the HALT commands also close the Odyssey application, with control returning to Windows.

The HALT() command simply halts Odyssey as described above. The HaltE() command is similar, but allows you to set a return code for a calling application to pick up. The value of the return code should be between 0 and 255.

Examples:

```
     Halt();
     HaltE(2); -- return error code = 2.
```

Odyssey Script Commands

IntToStr,StrToInt

PROC IntToStr(n:Number; VAR s:String);
PROC StrToInt(s:String; VAR n:Number);

These commands provide number to string conversion and vice versa. IntToStr converts the number "n" to ASCII form and stores it in the string variable "s". StrToInt takes a string containing a number in ASCII form and returns that number in the variable "n".

Examples:

```
     IntToStr(45, numstr);
     StrToInt(numstr, x);
```

Odyssey Script Commands

IsDirKey

FUNC IsDirKey(Key:String):Flag;

This command allows a script to determine whether an entry with a given key is present in the dialing directory or not. A robust script can therefore interrogate the directory for the correct entries before it enters its login routines. The function returns TRUE if "key" identifies a dialing directory entry, or FALSE if it does not.

Example:

```
IF NOT IsDirKey("ANYBBS") THEN
   Write("To run this script you need to have|");
   Write("an entry in your Dialing Directory|");
   Write("with a key of ANYBBS.|");
   EXIT();
END;
```

Odyssey Script Commands

Length

FUNC Length(s:String):Number;

This function returns the length in characters of the string passed in the "s" parameter.

Example:

```
Write("Please enter a filename: ");
Read(Filename);
IF Length(Filename)>8 THEN
   Write("The filename cannot be longer than|");
   Write("eight characters.");
   RETURN FALSE;
END;
```

Odyssey Script Commands

LogEvent

PROC LogEvent(s:String);

This command causes the string 's' to be recorded in the event log file (ODYSSEY.REC), provided that event logging is enabled. Odyssey prepends a date and time, and appends CRLF, otherwise the string is recorded exactly as you pass it.

Example:

```
     LogEvent("Description of Event");
```

Odyssey Script Commands

OdyVersion

FUNC OdyVersion():String;

This function returns a string containing the current Odyssey version number, eg. "2.00". You can use this function to ensure that the Odyssey version being used is compatible with any assumptions made in your script.

Example:

```
IF OdyVersion()<>"2.00" THEN
   Write("Warning: This script requires...");
   ....
```

Odyssey Script Commands

Pos

FUNC Pos(substring,s:String):Number;

This command searches the string "s" for the substring specified in the first parameter. If found then the function returns the character position of the substring in the second string. For example (where x is a number variable) :-

```
x := Pos("World", "Hello, World!!");
Write(x,"|");
```

would display "7" (the first position is zero). If the substring is not found then the Pos function returns minus one.

This command has suffered from a minor bug since it was first introduced. The bug is that, contrary to most other script commands involving string comparison, the Pos() command is case sensitive. Unfortunately, several users have discovered this and have written scripts which depend on this undocumented operation; so rather than fix the bug, we have been forced to turn it into a "feature", by documenting it here.

Example:

```
got_unread := Pos("Unread", line) <> -1;
```

Odyssey Script Commands

Priority

PROC Priority(enable:Flag);

Odyssey provides two commands KeyPressed() and RdKey() which respectively test the occurrence of a key press, and read a key from the keyboard. However there is a problem with this in that scripts normally run in the background, and any keys pressed are seen first by the terminal process and either transmitted to the remote host or used to invoke a function (menus etc.).

You can prevent this by calling **Priority(TRUE)**, which temporarily makes execution of the script the major priority, and so the script gets first look at the keyboard, allowing **KeyPressed()** and **RdKey()** to function correctly. However, do remember to call **Priority(FALSE)** when you no longer need access to the keyboard.

Odyssey will revert to the terminal process if the script execution is suspended for any reason - this happens while WaitFor(), or Delay() etc. calls are executed, and when **RdKey()** is called. The script still has priority however, so when these calls complete the script will take over again, and will continue to do so until you call **Priority(FALSE)**.

Examples:

```
     Priority(TRUE);
     Priority(FALSE);
```

Odyssey Script Commands

SetTimer,TimerExpired

PROC SetTimer(seconds:Number);
FUNC TimerExpired():Flag;

Scripts have access to a software timer, and these commands allow you to make use of this feature. **SetTimer()** allows a script to start the timer, which will expire in "seconds" seconds. You can test elsewhere in the script whether the time period has expired using a call to **TimerExpired()**.

TimerExpired() is a function, and returns TRUE if the timer has expired, or FALSE if it has not.

Example:

```
     SetTimer(10); -- expire in ten seconds
     (* do something here *)
     IF TimerExpired() THEN
        ....
```

Odyssey Script Commands

SilentMode

PROC SilentMode(enable:Flag);

In the Windows version of Odyssey, calling SilentMode(TRUE) causes Odyssey to minimize the Terminal Window - this might be used by scripts which wish to hide away the messy details of the interaction with the host, ie. to avoid confusing novice users. SilentMode(FALSE) restores the terminal window.

Examples:

```
SilentMode(TRUE);
....
SilentMode(FALSE);
```

Odyssey Script Commands

StrEdit

FUNC StrEdit(VAR s:String [, NoEcho]):Flag;

This command gives a script access to the string edit routine used internally by Odyssey for text input. The string "s" must be assigned an initial value before the function is called. The second parameter is optional, and controls whether Odyssey will echo characters or not. The function returns FALSE if the user cancelled the text dialog, TRUE if not. If TRUE, then "s" will contain the modified string.

Example:

```
IF StrEdit(s) THEN...
StrEdit(s,NoEcho);
```

Odyssey Script Commands

SubStr

FUNC SubStr(s:String; start,length:Number):String;

This function allows a script to extract substrings from original strings. The function returns that part of the string starting at character position "start" for "length" characters. If length is more than the number of characters in the string following the starting position then SubStr will copy all available characters from the start position onwards. Remember that character positions in strings start from zero, not one.

Example:

```
     s := SubStr(line,0,5);
```

Odyssey Script Commands

ToLower,ToUpper

FUNC ToLower(s:String):String;
FUNC ToUpper(s:String):String;

These functions convert strings between all lower case, and all caps. The converted string is returned as the function result.

Example:

```
     lower := ToLower("UPPER");
```