Cloning ICOM Receivers

Using the ICOM IC-R2 Handheld Scanner as an example

Data structures and computer control information

Version 1.0 of 14 Sept 2001

by BlakkeKatte@yahoo.co.uk
http://uk.geocities.com/blakkekatte

Cloning ICOM Receivers (using the ICOM IC-R2 Handheld Scanner as an example.)

Table of Contents

Acknowledgements

I developing this document, I have made extensive use of resources from the internet. I wish to acknowledge:

- Bruce A Pope for his document “Everything you always wanted to know about the IC-R10 that is not in the manual.” (http://people.we.mediaone.net/baptpdc/r10/r10.htm). In that document, Bruce describes his decoding of the Icom IC-R10 scanner.

- Ekki Plicht (http://www.plicht.de/ekki) who maintains a site with a lot of Icom programming material. Ekki has also written the CIVTest program (http://www.plicht.de/ekki/software/civt.htm) that allows commands to be sent to an ICOM radio and displays both the commands and the responses from the radio.

- Dave (AA6YQ) (http://www.ambersoft.com/Amateur_Radio/index.htm), author of the CI-V Explorer program (http://www.qsl.net/civ_commander) and other Icom computer control resources.

- Goran Vlaski (http://www.digital-laboratory.de/) for his freeware ICR2 programming utility and his Icom Clone Utility that can back up and restore the IC-R2 by storing clone files on disk.

- Gommert Buysen (http://www.butel.com.nl) for his Advanced Radio Control (ARC) program that can clone the IC-R2, and the iclone2 program which can back up and restore the IC-R2 by storing clone files on disk.

- Contributors to the IC-R2 discussion forum at http://groups.yahoo.com/group/icomr2.

- Icom for the DOS based software for programming the IC-R2, and their 1991 guide entitled “CI-V : Icom Communications Interface - V Reference Manual”

- Raihan Kibria (http://www.tu-darmstadt.de/~rkibria) for his frhed (Free Hex Editor) program. This easy to use and free utility makes an easy task of exploring, comparing and changing data files in either ASCII or Hex.

BlakkeKatte
September 2001

Data Representations

In this document, data is represented in a variety of ways:

- Characters are represented using the ASCII and ANSI character sets.
- Numbers are represented in hexadecimal, ASCII represented hexadecimal, or binary.

The ASCII Character Set

The ASCII table describes the American Standard Code for Information Interchange, the basis of character sets used in most present-day computers. US-ASCII uses only the lower seven bits (characters 0 to 127) to convey some control codes, space, numbers, most basic punctuation, and unaccented letters a-z and A-Z. Appendix B shows the ASCII Character set.

The ANSI Character Set

The ANSI character set (developed by the American National Standards Institute - ANSI), is a standard extension of the ASCII character set. The table at Appendix C shows characters 128-255 of the ANSI character set.

Hexadecimal Numbers.

Hexadecimal (Or "hex") numbers count in base 16 (decimal numbers count in base 10). Hexadecimal numbers are represented using the digits 0-9, with their usual meaning, plus the letters A-F (or a-f) to represent hexadecimal digits with values of (decimal) 10 to 15. The right-most digit counts ones, the next counts multiples of 16, then 16^2 = 256, etc.

There are many conventions for distinguishing hexadecimal numbers from decimal or other bases in programs. In the C programming language for example, the prefix "0x" is used, e.g. 0x694A11. In Visual Basic the prefix "&H" is used, e.g. &H3F6. In this document the prefix "$" is used, e.g. $3AF.

One of the advantages of using hexadecimal numbers is that two characters represent one byte of data (an 8 bit number). For example 159 (decimal) is represented as $9F.

In this document, numbers are usually represented in a hexadecimal format:

Hexadecimal number	$0	$1	$2	$3	$4	$5	$6	$7	$8	$9	$A	$B	$C	$D	$E	$F	$10
Decimal equivalent	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Binary Numbers

A number representation consisting of zeros and ones used by practically all computers because of its ease of implementation using digital electronics and Boolean algebra.

A **bit** is a binary digit. A single one or zero in a binary number.

A **byte** is eight bits. A byte typically holds one character. Two hexadecimal characters can represent the contents of a byte.

A **nibble** is half a byte. Since a byte is eight bits, a nibble is four bits. One hexadecimal character can represent the contents of one nibble.

ASCII Represented Hexadecimal

A data scheme where two ASCII characters are used to represent one byte of data. For example, one data byte containing the hexadecimal value $7E (126 Decimal, or 1111110 binary) would be represented by the two ASCII characters '7E'. The value 3 is represented by the two ASCII characters '03'.

Background to Cloning an ICOM Radio.

Background

Icom equipment has provision for computer control and management.

The full implementation can be used for two purposes:

- to 'clone' the configuration of a radio. That is, to set up the operational parameters of a radio (frequencies, modes, scanning rates etc) either by copying them from another radio of the same type (cloning) or by using software to send them to the radio. This process is called 'cloning' the radio. It generally involves turning the radio off after the cloning process is complete, and turning it back on to reinitialise.
- to control the radio while it is operating e.g. to tell the radio to change to another frequency or mode, or to obtain details from the radio such as received signal strength.

The Icom IC-R2 receiver only allows for 'cloning' by computer. It does not allow for operational control.

The CI-V System

The current ICOM standard system for communication between computer and radio is called the ICOM Computer Interface Version 5. This is commonly called the CI-V interface standard.

The CI-V system allows for a computer to communicate, through it's serial communications port, to an Icom radio.

In its full implementation, the CI-V system allows up to four radios to be controlled from the same cable and computer interface. Icom has developed the CT-17 interface to allow up to four radios to be connected to one computer.

Cloning uses the same protocol, but it is assumed that there will only be one radio communicating with one computer (or two radios communicating with each other) and the interface hardware can be simpler. Often the hardware is installed inside the plug for connection to the computer, with the hardware drawing its power from the computer's serial port. Icom has developed the OPC-478 interface for cloning from a computer.

Understanding the Hardware Interface.

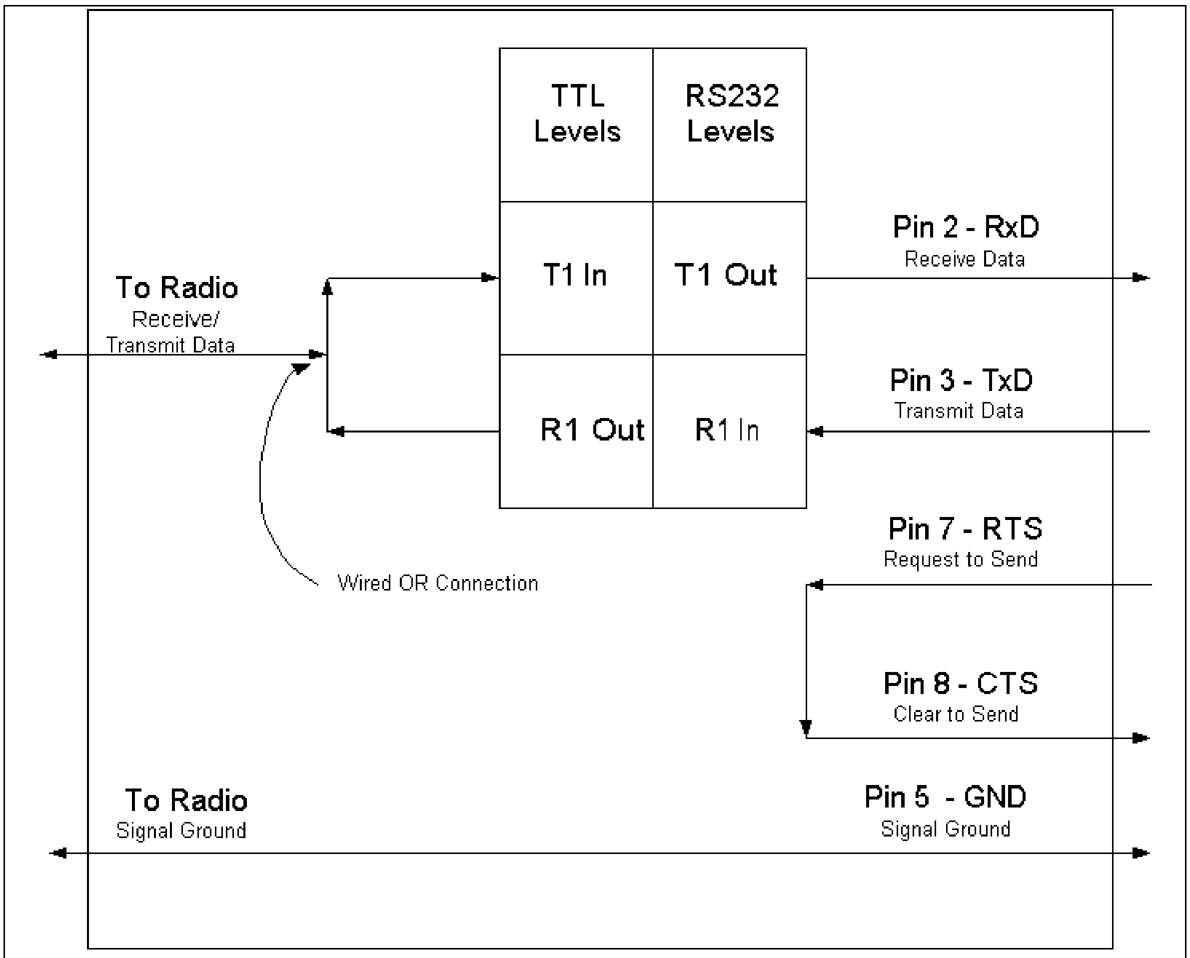
An interface is needed to convert between the low voltage TTL transistor logic of the IC-R2 to the higher voltage levels of the RS-232 computer serial communications interface.

Hardware required for communication between radio and PC is described below. Circuits are provided for the two ICOM Interface units:

- the CT-17 interface used for remote control;
- the OPC-478 interface used for cloning

These devices perform similar functions, but the plugs for connection to the radio are wired differently.

The Icom CT-17 interface is based on the Maxim Max232 RS-232 Line Driver/Receiver (http://pdfserv.maxim-ic.com/arpdf/1798.pdf). The following block diagram of the CT-17 interface shows how the RS-232 and TTL voltages are matched together, and shows the flow of data through the device.



ICOM CI-V to RS-232 converter - logical diagram
(RS-232 pinouts are for a nine pine connector)

This diagram clearly shows how data transmitted from the computer is reflected back to the computer for monitoring of the transmission. The diagram also shows that radio data is also reflected back to the radio so that collisions can be detected.

Because the transmit-data line and the receive-data line are connected together, the CI-V interface is connected in a wire-OR configuration. When the computer transmits a command, the command is automatically echoed back as received data, followed by the radio's response to the command, if any. For example, if an eleven-byte command is transmitted to a device on the cable, and a six-byte response is sent, the computer will receive a total of seventeen bytes (11+6=17).

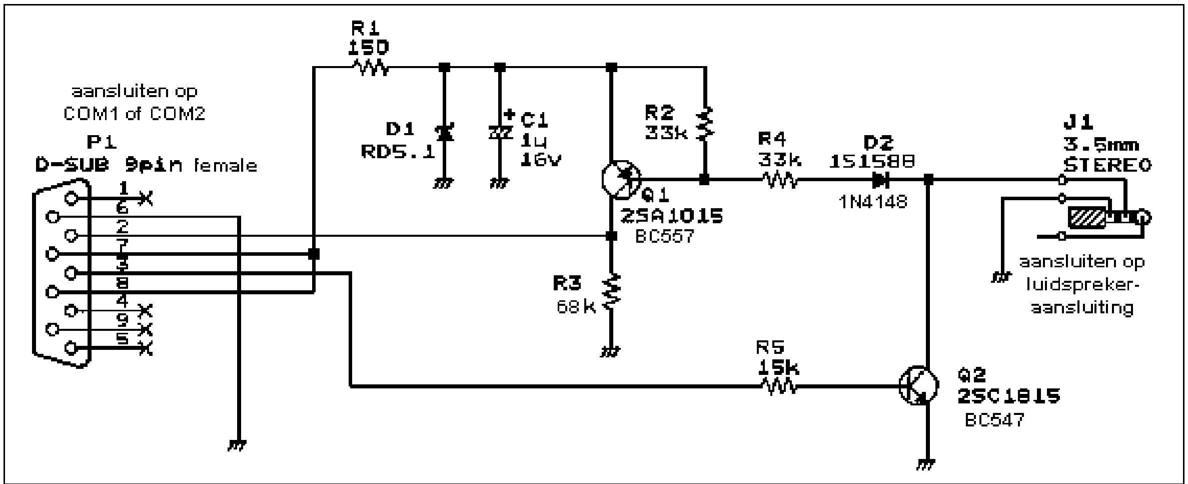
This configuration allows devices on the cable to monitor their own transmissions in order to detect interface collisions.

A collision occurs when two or more devices transmit simultaneously. If a collision occurs, the command must be re-transmitted. The radio or computer that is transmitting reads its own transmissions back from the communications cable. If it detects that another device has transmitted at the same time (i.e. there has been a collision) the radio or computer stops transmitting, listens to make sure that there are no other data transmissions on the cable, transmits a jamming code (5 characters of value $FC), and retransmits the original command.

Question: does the cloning system actually follow this part of the protocol? The cloning system seems to assume that no more than two devices will be on the cable at any one time and the protocol seems to minimise the possibility of collisions occurring during cloning

Icom OPC-478 Interface.

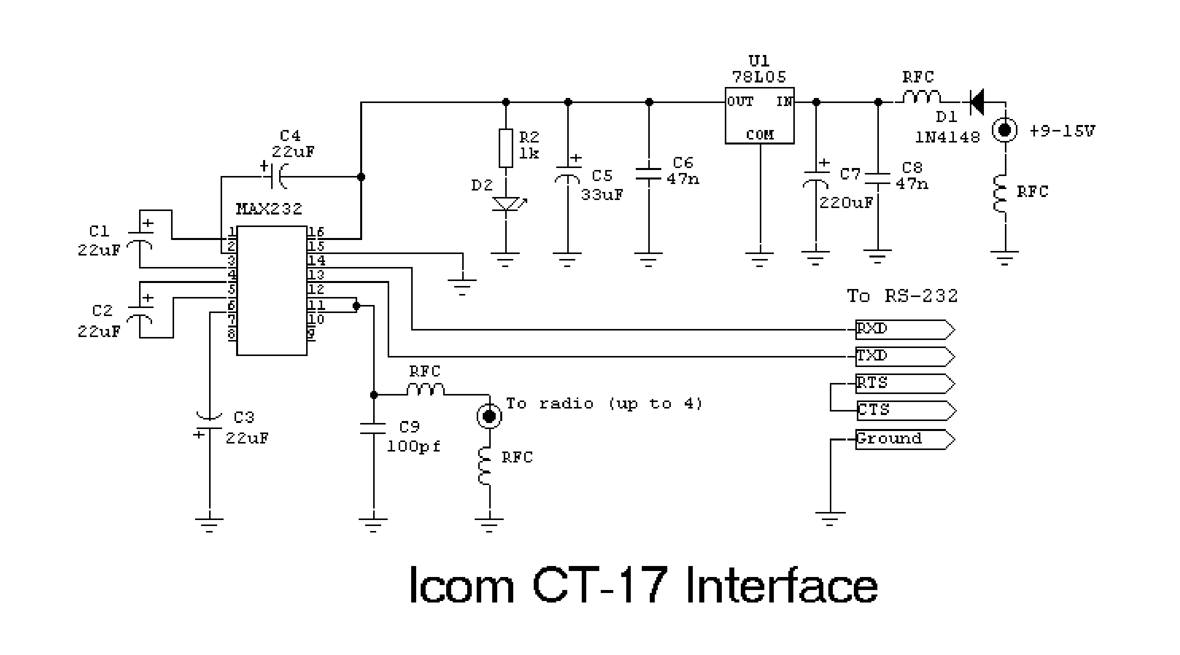
Following is the Icom design for the OPC-478 interface used for cloning. Note that this interface derives its power from the RS-232 serial port of the connected computer.



Source : http://www.lumanet.org/pages/rogerwilco/r_cci.gif.
This version adds common part numbers to the original ICOM design.

Icom CT-17 Interface.

Following is the Icom design for the CT-17 interface used for computer control of Icom equipment. This interface allows up to four devices to be controlled by one computer.



This example is taken from the Icom CI-V spec 1986 (source http://www.plicht.de)

Differences between the OPC-478 and CT-17 Interfaces.

These two items are electrically similar, except that they terminate in plugs that are wired differently. The OPC-478 cloning cable generally plugs into the earphone socket of the radio. The CT-17 serial cable generally plugs into a separate data socket. The difference in the plug wiring is:

Plug	OPC 478 Cloning Cable	CT-17 Serial Cable
Tip	unused	CI-V data
Centre	CI-V data	unused
Ring	Ground	Ground

A Tip. An OPC-478 cloning cable with a stereo plug can be converted for use as a serial cable by using a mono plug to stereo socket converter. This allows stereo earphones to be used in a mono socket. It effectively connects both the tip and the centre of the plug together, so that a signal applied to the tip of the plug, also appears on the centre of the stereo plug. If more than one device is to be connected, external cabling such as common 3.5mm Y-adapters can be used to connect multiple devices.

Other designs.

There are a number of sites on the internet that provide designs and construction details of RS232 to IC-R2 interfaces. See for example the collection of links at http://www.plicht.de/ekki/civ/index.html.

An interface similar to the CT-17 is published each year in the ARRL Handbook (Chapter 22 in the 2001 Handbook). The original design was published in 1993 in the ARRL QST magazine (http://www.arrl.org/tis/info/pdf/9302037.pdf) (copy at http://groups.yahoo.com/group/Icom_R-10/files/9302037.pdf) but is repeated each year in the Handbook. The article also gives a good summary of the RS-232 protocol and associated communications issues. The printed circuit board layout for the design is at http://www.arrl.org/notes/hbk-templates/iface.pdf.

Communication Parameters.

Following are the serial communication parameters for the Icom CI-V system:

Parameter	**Value**
Start bits	1
Data bits	8
Stop bits	1
Parity	None
Data rate	9600 (configurable). Early equipment had the baud rate fixed at 2400 baud. *Q is the IC-R2 configurable?*
Handshaking	None (NB: RTS and CTS are tied together at the computer end)

Icom Command Language.

Icom uses a simple command language to communicate between computer and radio. The basic structure of a command is

$FE	$FE	to-addr	fm-addr	Cmd	SubCmd	---Data---	$FD

$FE	1 byte	the preamble (sent twice)
to-addr	1 byte	the address of the device to which this command is being sent
fm-addr	1 byte	the address of the sender of the command
Cmd	1 byte	the code for the command to be followed
SubCmd	1 byte	Optional. A sub-command for some commands
Data	variable	the data to be processed
$FD	1 byte	the postamble, sent once.

In the case of the IC-R2, the following addresses are used:

Computer	$EF
IC-R2	$EE

Question: are these generic cloning addresses? If so, there are issues with cloning two radios on the same cable. Do the ICr3 use the same address for cloning? Does the IC-R10 have different addresses for cloning and control? Also make sure order of addresses is correct, and that examples are consistent.

Commands Used when Cloning.

The subset of the CI-V command set used for cloning is:

$E0	Interrogate radio for version/model/user comment	$FE $FE $EE $EF **$E0 <model data>** $FD For the IC-R2, the Icom provided software will send **$21 $27 $00 $00** if it is unsure of the model number. The software subsequently uses the value returned from the radio (see command $E1). For my version of the IC-R2 the model data is **$21 $27 $00 $01.** (A model code of $00 $00 $00 $00 can be used in this command. If it is used, any radio will respond

		with its actual model number)
$E1	Returned data (from radio)	$FE $FE $EF $EE **$E1 <model data><user data><some other data>**$FD The model data returned from the radio should be used in all subsequent commands. The user data returned from the radio is the user comment field loaded into memory when cloning (see details under "Memory Structure" later in this document.) Other data in my model returns ($0A $80 $01). Goran Vlaski in his software shows this as representing internal switch settings and the state of mods and fixes)
$E2	Set Radio into “Clone Out” mode. (Memory Read)	$FE $FE $EE $EF **$E2 <model data>** $FD The values for the model data should always be those returned by the radio. The radio responds by returning all of its memory data - see command $E4 for the format.
$E3	Set radio into “Clone In” Mode. (Memory Write)	$FE $FE $EE $EF **$E3 <model data>** $FD The values for the model data should always be those returned by the radio. The radio displays "clone in" on its LCD. This command must be followed by valid memory data for uploading - see command $E4 for the format.
$E4	Payload data. (Can be from computer to radio to be written to memory, or from radio to computer.)	$FE $FE $EE $EF **$E4 <line of data>** $FD (from computer to radio) $FE $FE $EF $EE **$E4 <line of data>** $FD (from radio to computer) The line of data contains the memory address, the payload length, the payload data, a checksum. The format of a line of data transmitted with this command is discussed in detail later in this paper.

$E5	Termination code. Send to radio at end of cloning memory write operation	$FE $FE $EE $EF **$E5 <some data>** $FD The data in this command decodes to 'Icom Inc.' This command must be sent or the cloning operation will terminate with an error.
$E6	Termination Result. Sent by radio at end of cloning memory write operation	$FE $FE $EF $EE $**E6 $00** $FD This is an acknowledgement from the radio giving the termination status of the cloning operation. $00 : Completed with no errors $01 : Completed with errors If the operation was unsuccessful, the radio will display CL Err on its LCD, and will have to be powered off to recover. On restart it reinitialises in a minimal default configuration.

Transferring Data.

The cloning process transfers data directly to the memory of the radio. The data transfer command provides details of the memory locations and data to be transferred.

The command that transfers data to or from a radio has a simple structure. The data is transferred using a series of commands, each with the same basic layout.

<table>
<tr><td rowspan="3">Command sequence to load data from the computer into radio memory</td><td>command preamble</td><td>$FE $FE</td></tr>
<tr><td>code to communicate from PC to radio</td><td>$EE $EF</td></tr>
<tr><td>command code for data transfer</td><td>$E4</td></tr>
<tr><td rowspan="3">Data</td><td>Starting Memory Address</td><td>2 ASCII represented hexadecimal numbers (4 characters)</td></tr>
<tr><td>Number of bytes of data to be loaded</td><td>1 ASCII represented hexadecimal number (2 characters</td></tr>
<tr><td>Data to be loaded</td><td>ASCII represented hexadecimal numbers. (2 characters per number)</td></tr>
</table>

	Checksum	1 ASCII represented hexadecimal number. (2 characters)
Terminating command byte showing data transfer is complete.	One hexadecimal number with the value $FD.	

Representation of the data to be transferred

Data being transferred is encoded in ASCII represented hexadecimal. Two ASCII characters are used to represent one byte of data.

Representing a number		
Decimal number	249	
Changed to hexadecimal	$F9	
converted to characters	'F' '9'	F9 would be seen in a word processor

Representing Characters		
Character string	Icom	
Changed to hexadecimal string ('I' is the 49th letter in the ASCII table, 'c' is the 63rd, o is the 6F th, m is the 6D th)	$49 $63 $6F $6D	
Converted to characters	4 9 6 3 6 F 6 D	as seen in word processor

Example line of data

An Icom command frame transferring data from computer to radio looks like this:

Line of data	þþîïä00301012640000000080061266000000008006C6ý

The above line contains this information:

Command sequence to load data from the computer into radio memory	command preamble	þþ	$FE $FE
	code to communicate from PC to radio	îï	$EE $EF
	command code for data transfer	ä	$E4

Starting memory address (e.g. 30 Hex)	`0030`	
Length of data (e.g. 10 Hex or 16 decimal)	`10`	
Data (32 characters representing 16 bytes of data)	`12640000000080061266000000008006`	
Checksum	C6	
Terminating command byte showing data is complete.	ý	$FD

Calculating a Checksum.

A checksum is used to trap errors in the data transmission. The transmitter calculates the checksum on the data it sends. The receiver recalculates the checksum on the data it receives. If part of the data is corrupted in transmission, the checksums will be different, and the receiver knows there has been an error. (If an error is detected, the radio stops the cloning operation and displays ‘CL Err’ (or Clone Error) on its LCD. The radio has to be reset to continue.

The CI-V system uses a very basic ‘twos-complement’ checksum. It will trap basic errors, but some more complex errors may not be detected.

(Twos complement is a number system used in some computers to represent negative numbers in binary. Each bit of the number is inverted (zeros are replaced with ones and vice versa, and then one (000...0001) is added (ignoring overflow)

Example Calculation:

Using logical operations on binary numbers, the checksum calculation looks like this.

Address, payload length, and data divided into 2 digit hex numbers	`00 30 10 12 64 00 00 00 00 80 06 12 66 00 00 00 00 80 06`	
Total of all the above 2 digit hex numbers.	$023A	0000 0010 0011 1010
Get the ones complement inverse	$023A XOR $FFFF = $FDC5	1111 1101 1100 0101
Add one to get the twos complement	$FDC5 + 1 = $FDC6	1111 1101 1100 0110

The checksum is two bytes long. We only need one byte (eight bits). Mask out the high end byte	$FDC6 AND $00FF = $C6	0000 0000 1100 0110
Checksum	$C6	1100 0110

IC-R2 Memory Structure

Methodology

To find out the memory structure of the IC-R2, the following methods were used:

- The official ICOM CS-R2 software was used. A copy was kept of a basic .ICF file produced by that software. Using the program, one change was made to the configuration of the radio, and the changed configuration was saved in a second .ICF file. The two files were compared and changes identified. This was repeated for each function provided by the software.
- A similar procedure was followed using software produced by others. In this case the changes were uploaded to the radio to confirm that the software was in fact producing valid results.
- An analysis of .ICF files also identified areas of data which appeared to have a useful purpose, but which were not being altered by any of the software packages. By guessing the format, making changes to the data, and uploading the altered data into the radio, it is sometimes possible to find the purpose of the data fields by finding the changed behaviour of the radio.

It can be seen that this is a time consuming, laborious exercise that requires a lot of attention to detail.

Accuracy and Completeness

Because of the process used, the memory table detailed in this document is not guaranteed to be error free, and anyone relying on the data should confirm its accuracy.

There are also a number of areas in memory that appear to be used by the radio, but which have a purpose that has not been identified. The table should not be regarded as complete.

Overall Structure

Memory Locations		Length Bytes	Data	Comments
Hex	Decimal			
0000-0C7F	0-3199	3200	Memory Channel Data	400 memory channels, of 8 bytes each
0C80-0DFF	3200 - 3583	384	Scan Edges	24 pairs Scan Edges of 16 bytes for each pair
0E00-0E0F	3584 – 3599	16		*Unused?*
0E10-0E5F	3600 – 3679	80	VFO data	10 VFO settings.
0E60-0E6F	3680 – 3695	16	Common parameters	
0E70-0E77	3696 – 3703	16	Values at startup	
0E78-0E7F	3704 – 3711	8		*Unknown?*
0E80-0E87	3712 – 3719	1	Common Parameters	
0E88-0E8F	3720 - 3727	8	skip values for channel mode	
0E90-0F9F	3728 – 3999	272		*Unused?*
0FA0-0FAF	4000 – 4015	16	Comments	User comment
0FB0-0FBF	4016 – 4031	16	Icom Version details	

0000 - 0C7F : Memory Channel Data.

The IC-R2 has 8 memory banks with 50 channels in each bank. This gives a total of 400 memory channels to be stored by the IC-R2. Details of the memory channels are the first thing stored in the R2. Each memory channel takes up 8 bytes of memory for storage. Storage is in 4 bit units or ‘nibbles’.

The memory does not divide channels into banks - only the 400 channels are stored. Only the programs used by the radio distinguish the banks.

The same structure is used for each edge of a Scan Edge pair and for VFO data.

The structure is as follows.

Bytes	Nibbles	Total	Purpose	Values
0-2	0-5	24 bits	Frequency	See separate note on frequency format
3	6	4 bits	Duplex	0 = Simplex 1 = Simplex 2 = -Duplex 3 = +Duplex
3-5-	7-11	20 bits	Offset	See separate notes on frequency format
6	12	Leftmost 2 bits of nibble 12 (a total of 2 bits)	Mode	0=FM (i.e. 00xx) 1=WFM i.e. (01xx) 2=AM (i.e. 10xx)
6	12-13	Rightmost 2 bits of nibble 12 and all of nibble 13 (a total of 6 bits)	CTCSS Tone	There are 50 CTCSS Tones. Range of values is $0 to $31 (0-49 decimal) $0 = xx00 0000 $31 = xx11 0001 See separate table for list of preset tones and their data values)
7	14	leftmost 2 bits	Program Skip	0=Scan (00xx) 1=Program Skip (01xx) 2=Scan Skip (10xx)
7	14	Rightmost 2 bits	Tone Squelch	0=off (xx00) 1=on (xx01)
7	15	Rightmost 4 bits	Tuning Step	There are 10 tuning steps 0=5 kHz 1=6.25 kHz 2=10 kHz 3=12.5 kHz 4=15 kHz 5=20 kHz 6=25 kHz 7=30 kHz 8=50 kHz 9=100 kHz F=9kHz in AM band ? *Auto????*

Frequency Formats

Frequencies and offsets are stored in kilohertz. *Resolution and rounding? The Icom CSR2 software shows 5 places after the decimal point)*

As an example, 123456 represents 123456 kHz or 123.456 MHz.

Frequency formats are in ASCII represented hexadecimal format where the leading two decimal digits will sometimes be represented by one hexadecimal character.

Frequencies are stored in 6 nibbles(3 bytes). Offsets are stored in 5 nibbles.

For frequecies:

- 1200.000 MHz is stored as C00000
- 200.000 MHz is stored as 200000
- 1,234.567 MHz is stored as C34567

For Offsets:

- 200.000 MHz is stored as C0000
- 20.000 MHz is stored as 20000
- 123456 MHz is stored as C3456.

Special Format – Broadcast band with 9kHz separation.

Some models accommodate the fact that, in some countries, stations in the broadcast band have a separation of 9kHz. Other countries provide for broadcast band separation of 10 kHz. The ability to handle 9 kHz separation is programmed into the IC-R2 in an unusual way.

The frequencies in the broadcast band range of 0.495 MHz to 1.620 MHz have a different format:

Frequency	Format	Calculation
0.495	00FFFF	0.495+0*0.009
0.504	01FFFF	0.495+1*0.009
0.513	02FFFF	0.495+2*0.009
….	….	….
0.576	09FFFF	0.495+9*0.009
0.585.	0AFFFF	0.495+10*0.009
0.594	0BFFFF	0.495+11*0.009
….	….	….
1.602	7BFFFF	0.495+123*0.009

1.611	7CFFFF	0.495+124*0.009
1.620	7DFFFF	0.495+125*0.009

This frequency format occupies 6 digits. The mode must be fixed as AM. Duplex, Offset, Tone Squelch, CTCSS Tone are not available. Only Scan Skip can be programmed.

Following is an example (spaces added for clarity).

```
13FFFF000000880F  27FFFF000000880F
```

This shows two stations 0.666 (13 hex [i.e. 19 decimal] * 0.009 + 0.495) and 0.846 (27 hex [i.e. 39 decimal] * 0.009 + 0.495). It shows Duplex set to 0, Offset frequency set to 0, An arbitrary CTCSS tone (no 8), tuning step set to F, mode set to AM, and the scan option set to ‘scan’.

CTCSS Tones Table

Data Value	CTCSS Tone
0	67.0
1	69.3
2	71.9
3	74.4
4	77.0
5	79.7
6	82.5
7	85.4
8	88.5
9	91.5
10	94.8
11	97.4
12	100.0
13	103.5
14	107.2
15	110.9
16	114.8
17	118.8
18	123.0
19	127.3
20	131.8
21	136.5
22	141.3
23	146.2
24	151.4

Data Value	CTCSS Tone
25	156.7
26	159.8
27	162.2
28	165.5
29	167.9
30	171.3
31	173.8
32	177.3
33	179.9
34	183.5
35	186.2
36	189.9
37	192.8
38	196.6
39	199.5
40	203.5
41	206.5
42	210.7
43	218.1
44	225.7
45	229.1
46	233.6
47	241.8
48	250.3
49	254.1

0C80-0DFF : Scan Edges.

There are 24 pairs of scan edges. Each pair has an upper frequency limit or edge (with all details), and a lower frequency limit (or edge). Each edge has the same structure as a memory frequency channel.

0E10-0E5F : VFO (Band) Data.

Each VFO element has the same structure as a memory frequency channel.

VFO (in CSR2 software)	VFO (in IC-R2 manual)	VFO Goran's program	Frequency Range of VFO	Default Frequency on Radio Initialisation
1	0.495	1.6	0.495 - 1.620 MHz	
5	5	5	1.625 - 29.995 MHz	
50	51	51	30 - 107.995 MHz	
		76		
Air	118	118	108 - 135.995 MHz	
VHF	145	145	136 - 255.095 MHz	
300	370	370	255.1 - 382.095 MHz	
UHF	430	430	382.1 - 769.795 MHz	
800	850	850	769.8 - 960.095 MHz	
1200	1295	1295	960.1 - 1309.995 MHz	

NOTE: To be resolved. The documentation gives 9 VFO slots. The data file has 10 slots for VFO data. Goran Vlaski's program shows 10 VFO slots.

0E60-0E6F : Common Parameters

Byte	Total Bytes	Purpose	Values
$0E60	1	Dial Select	0=100 kHz 1=1 MHz 2=10 MHz
$0E61	1	Priority	0=off 1=on 2=bell

			Question: where are the details of the priority frequency held?
$0E62	1	Scan Resume	0 = 0 sec 1=1 sec ….. 5=5 sec 6=Hold
$0E63	1	Scan Pause	0=2 sec 1=4 sec … 9=20 sec A=hold
$0E64	1	Program skip	0=off 1=on
$0E65	1	Bank scan	0=bank 1=all
$0E66	1	Expand Mode	0=off 1=on
$0E67	1	Channel mode	This also sets values at OE 80 byte 2 0=off 1=on (in both locations)
$0E68	1	Operation Beep	0=off 1= on
$0E69	1	Display backlight	0=off 1=on 2=auto
$0E6A	1	Auto Power Off	0=off 1=30 min 2=60 min 3=90 min 4=120 min
$0E6B	1	Power Save	0=off 1=auto
$0E6C	1	Monitor Mode	0=Push 1=Hold
$0E6D	1	Dial Speed	0=off 1=on
$0E6E	1	Scan Edge	0=P0 1=P1 ….. 18=P24 19=Band 1A=All

$0E6F	1	Lock Mode	0=Normal 1=No SQL 2=No Vol 3=All

0E70-0E77 : Values on Start-up

Byte	Total Bytes	Purpose	Values
$0E70		current menu option (The Menu option which will appear when menu is next called up)	0=skip 1=Dsel 2=TSQL 3=Tone 4=Dup 5=Offset 6=Resume 7=Pause 8=prio 9=Beep A=Light B=AP off C=Psave D=Moni E=Speed F=Lock 10=CH 11=Expand
$0E71		VFO to use (This relates to the ‘band’ to use’)	Bottom edge of the ‘band’ 0=0.495 MHz 1=1.620MHz 2=5 3=51 4=118 5=145 6=370 7=430 8=850 9=1295.0 MHz A=use Channel Mode
$0E72		Channel number	1=1 …… 3D = 62
$0E73		Squelch Level	1=Auto 2=Level 1

			…… A=Level 9
$0E74		Memory Channel	some error here 16=22, 18=24, 18=399(18F)
$0E75			
$0E76			
$0E77			

0E80-0E87 : Values on Start-up

Byte	Total Bytes	Purpose	Values
$0E80	top	attenuator	2=off, 6=on (0=off, 4=on) What is correct figure here?
	bottom	Operation on Power Up	*Normal=0* *Scan Tone=1* *Normal Scan=2*
$0E81	top	Memory or VFO	8=mem, 0=VFO (set to 0 if in channel mode
	bottom		Ch Mode 0=off, 1=on
$0E82			
$0E83			
$0E84			
$0E85			
$0E86			
$0E87			

0E88-0E8F : Channel Mode Values

Byte	Total Bits	Purpose	Values
$0E98 - $0E8F	64	TV Channel Skip	1 Bit per channel 0=off, 1=on Is this correct???????? Goran's program shows only 62 channels

0FA0-0FAF : User Comments

These locations contain 16 characters of user comments in text. For example:

55	73	65	72	20	43	6F	6D	6D	65	6E	74	20	31	32	33
U	s	e	r		C	o	m	m	e	n	t		1	2	3

0FB0-0FBF : Icom Version Details

These memory locations contain the following fixed data that is not amended.

49	63	6F	6D	43	6C	6F	6E	65	46	6F	72	6D	61	74	33
I	c	o	m	C	l	o	n	e	F	o	r	m	a	t	3

--00--

Appendix A : ICOM's ICF Disk File Format

Icom's .ICF format

Icom provides a DOS software program (CSR2) to load data into, and download data from the IC-R2. This program stores data files in a .ICF Format. This format is based on the format sent to the IC-R2, with some differences:

- The file is a text file. That is, it contains lines of ASCII characters, with each line being terminated with a Carriage Return and Line Feed combination.
- There are two extra lines at the beginning. The first contains some data which identifies the equipment model, and is used in the set up of the data transfer process. The second line contains user comments that are read by the Icom CSR2 program.
- The leading command bytes are not included
- The checksum is not included
- The terminating command byte is not included
- The data is encoded into a character based format. This encoding has to be removed before the data can be used

The First Two Lines.

The first line in the file contains the model number returned by the radio. For example:

First Line of ICF File	21270001

The second line contains a # sign followed by the user comments contained in the radio. The ICOM programs use this to provide a description when opening the file. An example second line is:

Second Line of ICF File	#User Comment

Both these lines are in normal, unencoded, text.

ICF File Coding.

The remainder of the ICF file contains the memory data that would be sent to (or received from) the radio. Only the starting memory location, length of data, and the data itself is recorded. The checksum is not recorded in an ICF file. The data in an ICF File is encoded using a simple coding system. Lines in an .ICF file look like this:

Encoded .ICF data	`ggjghghimkggggggggoggmhimmggggggggoggm`
Hex data after encoding removed (and with the Carriage Return and Line Feed removed)	`003010126400000000800612660000000008006`

Removing the encoding is simple, as the following example shows.

The letter ‘g’ is the 103rd character in the ASCII character set. Deducting 55 from that leaves 48. The 48th character in the ASCII character set is the number ‘0’. Thus, ‘g’ represents ‘0’. ‘h’ represents ‘1’ and so on.

code letter	g
position in ASCII table	103
subtract 55	103 - 55 = 48
Character at 48th position in ASCII table	'0'

code letter	}
position in ASCII table	125
subtract 55	125 - 55 = 70
Character at 70th position in ASCII table	'F'

As hexadecimal digits represent the data transferred to the IC-R2, only the 16 hexadecimal digits need to be decoded. The following table lists the code letters from the file, and the hexadecimal digits they represent.

code	**hex digit**
g	0
h	1
i	2
j	3
k	4
l	5
m	6
n	7

<table>
<tr><th>code</th><th>hex digit</th></tr>
<tr><td>o</td><td>8</td></tr>
<tr><td>p</td><td>9</td></tr>
<tr><td>x</td><td>A</td></tr>
<tr><td>y</td><td>B</td></tr>
<tr><td>z</td><td>C</td></tr>
<tr><td>{</td><td>D</td></tr>
<tr><td>|</td><td>E</td></tr>
<tr><td>}</td><td>F</td></tr>
</table>

--00--

Appendix B - ASCII Table

Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex
(nul)	0	0x00	(sp)	32	0x20	@	64	0x40	`	96	0x60
(soh)	1	0x01	!	33	0x21	A	65	0x41	a	97	0x61
(stx)	2	0x02	"	34	0x22	B	66	0x42	b	98	0x62
(etx)	3	0x03	#	35	0x23	C	67	0x43	c	99	0x63
(eot)	4	0x04	$	36	0x24	D	68	0x44	d	100	0x64
(enq)	5	0x05	%	37	0x25	E	69	0x45	e	101	0x65
(ack)	6	0x06	&	38	0x26	F	70	0x46	f	102	0x66
(bel)	7	0x07	'	39	0x27	G	71	0x47	g	103	0x67
(bs)	8	0x08	(	40	0x28	H	72	0x48	h	104	0x68
(ht)	9	0x09	)	41	0x29	I	73	0x49	i	105	0x69
(nl)	10	0x0a	*	42	0x2a	J	74	0x4a	j	106	0x6a
(vt)	11	0x0b	+	43	0x2b	K	75	0x4b	k	107	0x6b
(np)	12	0x0c	,	44	0x2c	L	76	0x4c	l	108	0x6c
(cr)	13	0x0d	-	45	0x2d	M	77	0x4d	m	109	0x6d
(so)	14	0x0e	.	46	0x2e	N	78	0x4e	n	110	0x6e
(si)	15	0x0f	/	47	0x2f	O	79	0x4f	o	111	0x6f
(dle)	16	0x10	0	48	0x30	P	80	0x50	p	112	0x70
(dc1)	17	0x11	1	49	0x31	Q	81	0x51	q	113	0x71
(dc2)	18	0x12	2	50	0x32	R	82	0x52	r	114	0x72
(dc3)	19	0x13	3	51	0x33	S	83	0x53	s	115	0x73
(dc4)	20	0x14	4	52	0x34	T	84	0x54	t	116	0x74
(nak)	21	0x15	5	53	0x35	U	85	0x55	u	117	0x75
(syn)	22	0x16	6	54	0x36	V	86	0x56	v	118	0x76
(etb)	23	0x17	7	55	0x37	W	87	0x57	w	119	0x77
(can)	24	0x18	8	56	0x38	X	88	0x58	x	120	0x78
(em)	25	0x19	9	57	0x39	Y	89	0x59	y	121	0x79
(sub)	26	0x1a	:	58	0x3a	Z	90	0x5a	z	122	0x7a
(esc)	27	0x1b	;	59	0x3b	[	91	0x5b	{	123	0x7b
(fs)	28	0x1c	<	60	0x3c	\	92	0x5c	|	124	0x7c
(gs)	29	0x1d	=	61	0x3d	]	93	0x5d	}	125	0x7d
(rs)	30	0x1e	>	62	0x3e	^	94	0x5e	~	126	0x7e
(us)	31	0x1f	?	63	0x3f	_	95	0x5f	(del)	127	0x7f

Appendix C - ANSI Table

Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex
€	128	80	Ÿ	160	A0	À	192	CO	à	224	E0
	129	81		161	A1	Á	193	C1	á	225	E1
‚	130	82	¡	162	A2	Â	194	C2	â	226	E2
ƒ	131	83	¢	163	A3	Ã	195	C3	ã	227	E3
„	132	84	£	164	A4	Ä	196	C4	ä	228	E4
…	133	85	¤	165	A5	Å	197	C5	å	229	E5
†	134	86	¥	166	A6	Æ	198	C6	æ	230	E6
‡	135	87	¦	167	A7	Ç	199	C7	ç	231	E7
ˆ	136	88	§	168	A8	È	200	C8	è	232	E8
‰	137	89	¨	169	A9	É	201	C9	é	233	E9
Š	138	8A	©	170	AA	Ê	202	CA	ê	234	EA

‹	139	8B
Œ	140	8C
	141	8D
Ž	142	8E
	143	8F
	144	90
‘	145	91
’	146	92
“	147	93
”	148	94
•	149	95
–	150	96
—	151	97
˜	152	98
™	153	99
š	154	9A
›	155	9B
œ	156	9C
	157	9D
ž	158	9E
Ÿ	159	9F

ª	171	AB
«	172	AC
¬	173	AD
	174	AE
®	175	AF
¯	176	BO
°	177	B1
±	178	B2
²	179	B3
³	180	B4
´	181	B5
µ	182	B6
¶	183	B7
·	184	B8
¸	185	B9
¹	186	BA
º	187	BB
»	188	BC
¼	189	BD
½	190	BE
¾	191	BF

Ë	203	CB
Ì	204	CC
Í	205	CD
Î	206	CE
Ï	207	CF
Ð	208	D0
Ñ	209	D1
Ò	210	D2
Ó	211	D3
Ô	212	D4
Õ	213	D5
Ö	214	D6
×	215	D7
Ø	216	D8
Ù	217	D9
Ú	218	DA
Û	219	DB
Ü	220	DC
Ý	221	DD
Þ	222	DE
ß	223	DF

ë	235	EB
ì	236	EC
í	237	ED
î	238	EE
ï	239	EF
ð	240	F0
ñ	241	F1
ò	242	F2
ó	243	F3
ô	244	F4
õ	245	F5
ö	246	F6
÷	247	F7
ø	248	F8
ù	249	F9
ú	250	FA
û	251	FB
ü	252	FC
ý	253	FD
þ	254	FE
ÿ	255	FF