XM Protocol Specification

Protocol Description for the XM PCR/Direct/Commander products

Last Updated: **Wednesday, January 05, 2005**

This document created by:

Ben S. Stahlhood
Hybrid Mobile Technologies
bstahlhood@hybrid-mobile.com
bitoholic@gmail.com
http://weblogs.asp.net/bstahlhood/

Credit to those that make this possible:

Perl XM PCR project
Nick Sayer
XM Fan xmfan.com

Table of Contents

1 Introduction 3
1.1 Objectives 3
2 The rise and fall of the XM PCR 3
2.1 The rise of XM PCR 3
2.2 Third party software 4
2.3 The whipping boy and the fall of XM PCR 4
2.4 The aftermath 5
3 The XM PCR protocol command set 6
3.1 The PCR byte code layouts 6
3.2 The “PCR” mode byte codes 7
3.3 The PCR command set 7
Command: Power Up 8
Command: Power Down 8
Command: Get Radio ID 8
Command: Set Mute 9
Command: Change Channel 9
Command: Get Channel Info 9
Command: Get Extended Channel Info 9
Command: Monitor Channel Data 10
Command: Get Signal Data 11
3.4 Combination of commands 11
3.5 Conclusion 12

1 Introduction

When I originally started to make something for the XM hobbyists crowd, I wrote a library in .NET/C# that would allow you to talk to the XM devices. Later, I had so many requests from others that used Java, C/C++, Visual Basic 6, etc. I figured the best way is to just document everything here in once nice document and let you guys write the libraries and SDK's. Plus I have to admit that the one I wrote was such a rush job, it was not fit for coding consumption ;)

So I will get on to the objectives and break down how to communicate with the XM devices outlined in this document. I will explain the protocol at the byte code string level, explaining what each byte or series of bytes mean. If you have any questions or comments you can reach me at the contact info specified on the cover page.

1.1 Objectives

The objectives of this document are:

a) Give a little history about the XM hardware. (Skip right to Section 3 if you do not care to hear about this)
b) Explain the difference between the XM PCR and the XM Direct/Commander hardware.
c) Explain the byte code string layout
d) Explain what commands can be sent and what each one does.
e) Explain what to expect back from the hardware as responses to sent commands.

2 The rise and fall of the XM PCR

2.1 The rise of XM PCR

XM Satellite Radio has always been available for automobile owners. Devices such as the Delphi XM SKYFi allowed users to listen to XM Satellite Radio in there car and then bring it inside there home and drop it in a cradle to listen on your home stereo. They also have a kit that is a portable audio system.

Enter the XM PCR. This device would connect to your computer via USB. It allowed you to listen to XM Radio on your home computer while you worked. It came with a packaged software product from XM that had all the channel listings, favorites, signal strength, and other relevant info a user could find helpful. It did not take long before computer programmers sniffed the communications and reverse engineered the protocol for the XM PCR device. XM was just doing serial over USB. It used common drivers for the FTDI chipset. It would create a virtual COM port for the device to communicate with.

2.2 Third party software

The initial project that I am aware of was the Open XM Perl project. The programmer of this project was where a lot of the initial info on the XM PCR protocol originated. Then Nick Sayer wanted to develop an application for the Mac crowd so he could learn Java/Cocoa programming for Mac OS X. So he did an amazing job at porting and adding a lot of functionality. He has kept up on all the XM news to this day as far as I know.

A ton of applications started appearing on the web: MacXM, XtremePCR, PCRCommander, TouchXM, Mobile Media Center (our product), and FrodoPlayer. These all had similar functionality but tried to give a different user experience, or XM was just a value added feature to an already established software package.

2.3 The whipping boy and the fall of XM PCR

A gentleman by the name of Scott Maclean out of Canada owned his own company called Nerosoft. He started of creating a Visual Basic Active X component that would allow programmers to easily communicate and control the XM PCR. He then decided to write an application with Tivo like abilities. He was tired of missing his favorite shows on XM. He created a piece of software called TimeTrax that allowed time shifted recording. You could set all kinds of options to allow you to record content off the XM Satellite Radio network and save it to MP3 audio files. It was a great product idea and it took off like crazy.

You can imagine it did not take long for the men in black suites from the RIAA to come knocking. XM denies it, but most feel that the RIAA must have put pressure on XM for allowing this to happen. Within a matter of weeks the XM PCR was pulled off the shelves. This made a lot of users upset as you can imagine. The XM PCR became a precious commodity all of a sudden. People were selling them on eBay for an average price of 400.00.

2.4 The aftermath

Users were not the only ones affected. All the developers out there of third party XM Software were upset as well. All that time and effort and now the device there software relied on was gone. Software like MacXM, XtremePCR, PCRCommander, TouchXM, Mobile Media Center (our product), and FrodoPlayer were all now deprecated so to speak. Only those users that could track down an XM PCR or already had one would benefit from these applications. The rest would be left out in the cold. No more XM PCR.

XM had released a product sometime ago called the XM Commander. It has a remote interface and is made to be hooked up in the car. It would work with any existing setup since it had its own remote interface that you mounted in your car. This product goes for $169.99. This was a huge price difference when compared to the XM PCR which was priced at $49.99. The other major issue was it had its own specialized interface and not USB. So this could not be used by home users.

Many people all around the same time started reverse engineering the communications of the XM Commander. Hybrid Mobile Technologies as well as others knew that XM had announced another product called XM Direct, which looked almost like the XM Commander, with a few medications. It would use special after-market adaptors to communicate with head units. We realized that the remote interface for the XM Commander acted as head unit and we knew if sniffed the communications it would provide the byte codes we would need, plus when the XM Direct arrived we would have another XM PCR like device to use. This brings us to now and the source of information that this document covers.

3 The XM PCR protocol command set

3.1 The PCR byte code layouts

The XM protocol is based on bi-direction communication. There are requests sent from the "controller" device to the XM hardware. The "controller" being your PC, head units, and the remove control from XM. There are responses sent from the XM hardware back to the "controller", basically this is the feedback based on the requests you sent. You can look at this as your calling a function and you want a return type to check against. The responses will let you know if the XM hardware is getting the right commands or not, or in some cases return the data you are requesting (Channel Info, Song Tile, Artists, etc).

All requests take the form:

5A A5 <length of data> <command data> ED ED

5A A5 = header bytes
<length of data> = two bytes in the big-endian (MSB) byte order.
<command data> = the number of bytes that <length of data> specifies
ED ED = tail bytes

All responses take the form:

5A A5 <length of data> <command data> <checksum>

5A A5 = header bytes
<length of data> = two bytes in the big-endian (MSB) byte order.
<command data> = the number of bytes that <length of data> specifies.
<checksum> = the number of bytes received from the requests.

The trick to the XM Direct and Commander is to send a set of bytes codes in a particular order to turn the XM Direct or Commander into what I call "PCR" mode. Then all of the following commands work on all three hardware devices. There are checksum bytes in the XM Direct and Commander, but they are ignored at this point in time, so just use the standard PCR EDED for all commands.

3.2 The "PCR" mode byte codes

You send the following codes. You will acknowledge a response from the first two requests, but not the last one.

5AA50003740001EDED	Put Hardware into listening/command mode
5AA5000474020101EDED	Turn on the power
5AA50003740B00EDED	Turn the DAC mute off

3.3 The PCR command set

I will now list all the commands, what they do, and the response data to expect. I will do it using the following format:

<command name>
<command description>

<command requests bytes>
<command requests bytes arguments or info

<command response bytes> for the bytes you should ignore I will substitute XX. The first byte after the header bytes and data length bytes is always your response byte code for the command you sent. It should always be the command byte with the MSB set. Example, if you send the command byte 13 which is mute you will get a response byte of 93.

<command response byte data, if applicable>

Command: Power Up

Description: Power up the XM device

Byte codes: **00AABBCCDD**
AA – Channel label size (can be 08, 10, 16)
BB – Category label size (can be 08, 10, 16)
CC – Artists and tile size (can be 08, 10, 16)
DD – Radio type (01 means "loss of power is expected")

Response bytes:
80ZZXXXXAABBCCDDEEXXXXXXXXFFGGHHIIJJKKKKKKKKKKKKKKKK
ZZ - (If 0x03 is returned then the radio is not activated)
AA – RX Version
BB/CC/DDEE – RX date in BCD format
FF – CMB Version
GG/HH/IIJJ – CBM date in BCD format
KKKKKKKKKKKKKKKK – XM Radio ID (8 ASCII CHARACTERS)

Command: Power Down
Description: Power down the XM device

Byte codes: **01AA**
AA – Power save mode off if 00 and on if 01

Response bytes: **81**

Command: Power Down

Description: Power down the XM device

Byte codes: **01AA**
AA – 01 for power saving mode, 00 for off

Response bytes: **81**

Command: Get Radio ID

Description: Get the XM Radio ID

Byte codes: **31**

Response bytes: **B1XXXXXXAAAAAAAAAAAAAAAA**
AAAAAAAAAAAAAAAA – XM Radio ID (8 ASCII CHARACTERS)

Command: Set Mute

Description: Set mute on or off

Byte codes: **13AA**
AA – 00 for off 01 for on

Response bytes: **93**

Command: Change Channel

Description: Change the channel on the device

Byte codes: **1002AA000001**
AA – Channel Number

Response bytes: **90**

Command: Get Channel Info

Description: Get the channel information data (Artist, Title, Genre)

Byte codes: **25AABB00**
AA – Selection Method (08 = Channel Info for channel passed in BB, 09 = Next Channel
BB – Channel Number

Response bytes: **A5XXXXAABBCC<DD>EEXX<FF>GG<HH>**
AA – Channel Number
BB – Service ID
CC – Station Name Checksum (If 0 then Station data is bad)
DD – Station Name (16 ASCII Characters, White space is 0x20)
EE – Genre Name Checksum (If 0 then Genre data is bad)
FF – Genre Name (16 ASCII Characters, White space is 0x20)
GG – Song Title Checksum (If 0 then Song Title data is bad)
HH – Song Title (16 ASCII Characters, White space is 0x20)

Command: Get Extended Channel Info

Description: Get the extended channel information data (Long Artist, Long Title)

Byte codes: **22AA**
BB – Channel Number

Response bytes: **A2XXXXXXAA<BB>CC<DD>**
AA – Artist Checksum (If 0 then Artist Data is bad)
BB – Long Artist Name (32 ASCII Characters, White space is 0x20)
CC – Title Checksum (If 0 then Title Data is bad)
DD – Long Title Name (32 ASCII Characters, White space is 0x20)

Command: Monitor Channel Data

Description: Event driven, the XM device will send data when data changes.

Byte codes: **50AABBCCDDEE**
- AA – Channel to Monitor
- BB – Monitor Service ID (true 0x01, false 0x00)
- CC – Monitor Program Type (true 0x01, false 0x00)
- DD – Monitor Channel Info (true 0x01, false 0x00)
- EE – Monitor Extended Channel Info (true 0x01, false 0x00)

Response bytes: **D0XXXXAA**
- AA – Channel Number

Special response bytes to monitor after this command has been sent:
- **D1AABB<CC> –** Channel Name Changed
  - AA – Channel Number
  - BB – Data Checksum (If 0 then data is bad)
  - CC – Data (16 ASCII Characters)
- **D2AAXXBB<CC> –** Channel Genre Changed
  - AA – Channel Number
  - BB – Data Checksum (If 0 then data is bad)
  - CC – Data (16 ASCII Characters)
- **D3AABB<CC><DD> –** Channel Artist / Title Changed
  - AA – Channel Number
  - BB – Data Checksum (If 0 then data is bad)
  - CC – Data (16 ASCII Characters)
- **D4AABB<CC> –** Extended Artist Changed
  - AA – Channel Number
  - BB – Data Checksum (If 0 then data is bad)
  - CC – Data (32 ASCII Characters)
- **D5AABB<CC> –** Extended Title Changed
  - AA – Channel Number
  - BB – Data Checksum (If 0 then data is bad)
  - CC – Data (32 ASCII Characters)
- **D6AABBCCDDEEFFGGHH –** Song Time
  - AA – Channel Number
  - BB – Time Format
  - CC – Duration Time Checksum (If 0 then data is bad)
  - DD – Progress Time Checksum (If 0 then data is bad)
  - EE & FF – Duration Time in seconds
  - GG & HH – Progress Time in seconds

NOTE: This should be called to turn all monitoring off just as you change the channel and then you can send it again to turn it back on… if you do not do this, you can cause excessive data flow. This will cause slow results or unexpected results.

Command: Get Signal Data

Description: Get all satellite and terrestrial signal data

Byte codes: **43**

Response bytes:

C3XXXXSSAATTZZYYOOBBCCDDEEFFGGHHIIJJXXXXXXXXKKLLMMNNXXXX

SS – Sat. Signal Status (0 – None, 1 – Fair, 2 – Good, 3 – Excellent)
AA – Antenna Status (0 – Not Hooked Up, 3 – Hooked Up)
TT – Terrestrial Signal Status (0 – None, 1 – Fair, 2 – Good, 3 – Excellent)
ZZ – Sat. 1 QPSK
YY – Sat. 2 QPSK
OO – Ter. MCM
BB – Sat. 1 TDM
CC – Sat. 2 TDM
DD – Ter. TDM
EE & FF – Sat. 1 BER
GG & HH – Sat. 2 BER
II & JJ – Ter. BER
KK – Sat. AGC
LL – Ter. AGC
MM – Sat. 1 CN
NN – Sat. 2 CN

3.4 Combination of commands

Everything that you want to accomplish can be achieved with all the commands listed. You may have to use a combination of commands together to get certain tasks done. For example, do get a channel lists you will need to use the Get Channel command passing 0 as a channel and the mode 9 for next channel number. You keep doing this until it returns 0 as the channel number again. This will give you the exact channels that the current subscription has access to. Each time the channel number returns you pass that to the get channel info command to get your channel data. This way you can create a list of channels with the appropriate data associated with it.

I will be making constant updates to this document as I find more information out. If you would like to contribute or have corrections, please email me. If you have any questions or need help with the commands, once again do not hesitate to email me.

3.5 Conclusion

I hope this document is what you were looking for when you went looking for XM protocol information. I tried my best to explain it. Please email me if you need anything.

I would like to thank the mp3car.com community, XM Fan, all the XM software developers, and XM for creating cool hardware to play with.