Xbox Debug Monitor

From xboxdevwiki
Jump to: navigation, search

The **Xbox Debug Monitor** (**XBDM**) is a feature of Xbox Development Kits that provides remote debugging, file management, console discovery, and other services on TCP/UDP port 731. It is loaded by debug kernels at startup from `C:\xbdm.dll` and its configuration is read from `E:\xbdm.ini`. XBDM is distinct from KD and uses a different wire protocol.

Contents

Name Answering Protocol

An Xbox Development Kit (XDK) can be assigned a *debug name* that identifies it on the local network. XBDM provides the ability to resolve a debug name to an IP address (forward lookup), resolve an IP address to a debug name (reverse lookup), and discover all XDKs on the local network using a very simple UDP-based protocol.

Name Answering Protocol Packet

Offsets **Octet**	**0**								**1**							
Octet	**Bit**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	**0**	Type								Name Length							
2	**16+**	Name															

A NAP packet contains 3 fields, the last of which is variable-length. The minimum length of a NAP packet is 2 bytes and the maximum is 257. Invalid packets are silently dropped by XBDM.

Type
: This unsigned 8-bit field may contain the values 1 (lookup), 2 (reply), or 3 (wildcard).

Name Length
: This unsigned 8-bit field specifies the length of the Name field and should be a value from 0 to 255. For Type 3 packets, this field should always be 0. For Type 1 and Type 2 packets, this field should never be 0.

Name
: This variable-length field contains the ASCII-encoded debug name for Type 1 and Type 2 packets. The number of bytes in this field is given by the Length field. It should not contain any `NUL` characters.

Forward Lookup

To resolve a debug name to an IP address, send a Type 1 NAP packet containing the debug name to be resolved to UDP address 255.255.255.255:731. The XDK with that name will respond with a Type 2 NAP packet and its IP address can be retrieved from the UDP header. There is no way to prevent multiple XDKs being assigned the same debug name, so it's possible that the client may receive replies from multiple IP addresses.

Reverse Lookup

To resolve an IP address to a debug name, send a Type 3 NAP packet with no name (length 0) to the IP address on UDP port 731. Assuming the target is actually an XDK, it will respond with a Type 2 NAP packet containing its name. This is very similar to the Console Discovery process (below), except that by sending the wildcard packet to a single IP address, only that XDK will respond.

Console Discovery

To discover all XDKs on the local network, send a Type 3 NAP packet with no name (length 0) to the UDP address 255.255.255.255:731. Each XDK will respond with a Type 2 NAP packet containing its name. As with a forward lookup, the client may receive multiple replies with the same name, but different IP addresses.

Remote Debugging and Control Protocol

The Remote Debugging and Control Protocol (RDCP) is a text-based protocol transmitted over a TCP connection on port 731. RDCP resembles protocols like FTP and SMTP, making it possible to communicate with XBDM using just a Telnet client in many cases.

When a connection is established, XBDM sends `201- connected` (or `200- connected` in version 3944) followed by <CR><LF> (that is, a carriage return character followed by a line feed character). The RDCP client is then free to send a command followed by <CR><LF> or <CR><NUL>.

After executing a command, XBDM replies with a response line consisting of a three-digit status code and message of the form `999- message text<CR><LF>`. Note that unlike similar protocols, the `-` (dash) is always present in responses and messages cannot span multiple lines.

Status codes

In responses, 2xx status codes indicate success and 4xx codes indicate failure. Most codes have a default message, but some commands leave the message field empty while others use the message field to hold whatever data was requested by the client or additional information about an error.

2xx Success

200- OK
: Standard response for successful execution of a command.

201- connected
: Initial response sent after a connection is established. The client does not need to send anything to solicit this response.

202- multiline response follows
: The response line is followed by one or more additional lines of data terminated by a line containing only a `.` (period). The client must read all available lines before sending another command.

203- binary response follows
: The response line is followed by raw binary data, the length of which is indicated in some command-specific way. The client must read all available data before sending another command.

204- send binary data
: The command is expecting additional binary data from the client. After the client sends the required number of bytes, XBDM will send another response line with the final result of the command.

205- connection dedicated
: The connection has been moved to a dedicated handler thread (see #Connection dedication).

4xx Failure

400- unexpected error
: An internal error occurred that could not be translated to a standard error code. The message is typically more descriptive, such as "out of memory" or "bad parameter".

401- max number of connections exceeded

The connection could not be established because XBDM is already serving the maximum number of clients (4).

402- file not found
: An operation was attempted on a file that does not exist.

403- no such module
: An operation was attempted on a module that does not exist.

404- memory not mapped
: An operation was attempted on a region of memory that is not mapped in the page table.

405- no such thread
: An operation was attempted on a thread that does not exist.

406-
: An attempt to set the system time with the `setsystime` command failed. This status code is undocumented.

407- unknown command
: The command is not recognized.

408- not stopped
: The target thread is not stopped.

409- file must be copied
: A move operation was attempted on a file that can only be copied.

410- file already exists
: A file could not be created or moved because one already exists with the same name.

411- directory not empty
: A directory could not be deleted because it still contains files and/or directories.

412- filename is invalid
: The specified file contains invalid characters or is too long.

413- file cannot be created
: The file cannot be created for some unspecified reason.

414- access denied
: The file cannot be accessed at the connection's current privilege level (see #Security).

415- no room on device
: The target device has run out of storage space.

416- not debuggable
: The title is not debuggable.

417- type invalid
: The performance counter type is invalid.

418- data not available
: The performance counter data is not available.

420- box not locked
: The command can only be executed when security is enabled (see #Security).

421- key exchange required
: The client must perform a key exchange with the `keyxchg` command (see #Security).

422- dedicated connection required
: The command can only be executed on a dedicated connection (see #Connection dedication).

Connection dedication

Connection dedication is the process of moving a client connection from the *global server thread* to a *threaded command handler thread* with the `dedicate` command.

By default, commands sent to XBDM are processed on the global server thread. Built-in commands and custom command handlers registered with `DmRegisterCommandProcessor` are run on this thread and may only execute kernel APIs. Commands that need to call CRT or XAPI functions must run in a threaded command handler, registered with `DmRegisterCommandProcessorEx` or `DmRegisterThreadedCommandProcessor`.

A client that has not dedicated its connection will receive a `422- dedicated connection required` response if it tries to execute a threaded command on the global server thread. After dedicating itself to a threaded command handler, the client can no longer send built-in or non-threaded commands until it re-dedicates itself to the global server thread.

Security

TODO

Commands

See also: XBDM commands by version

A command consists of a name and zero or more parameters separated by whitespace characters. The format of the parameters is defined by the command, but most commands use the form `key=value`. Parameter values that contain whitespace characters must be surrounded by double quotes (e.g. `"some value"` or `key="some value"`).

In the command descriptions below, the following data types are used:

Type	Description
DWORD	A 32-bit integer in hexadecimal format (e.g. `0x1234ABCD`).
QWORD	A 64-bit integer in hexadecimal format, but prefixed with 0q instead of 0x (e.g. `0q0123456789ABCDEF`).
STRING	An ASCII-encoded string, optionally surrounded by double quotes.

adminpw (Set administrator password)

4039+ `adminpw none` 🔒 manage

Clear the administrator password.

4039+ `adminpw passwd=QWORD` 🔒 manage

Set the administrator password to the value of the `passwd` parameter. Note that `passwd` is a 64-bit integer instead of a string. The details of the conversion from a string password to a 64-bit integer are currently unknown.

altaddr

authuser (Authenticate user)

boxid

break

bye (Close connection)

capcontrol/capctrl

continue

crashdump

d3dopcode

dbgname (Get/set debug name)

dbgoptions

debugger

debugmode

dedicate (Dedicate connection)

deftitle

delete (Delete file or directory)

3944+ `delete name=STRING dir` 🔒 -

Deletes a file or directory.

To delete a directory, the optional `dir` attribute must be present.

dirlist (List files in directory)

dmversion (Get debug monitor version)

drivefreespace (Get free space on drive)

drivelist (List drive letters)

3944+ `drivelist` 🔒 -

Returns a string which contains the drive-letter for each accessible drive.

dvdblk (Read block from DVD)

dvdperf

fileeof

flash (Flash BIOS image)

fmtfat (Format FAT partition)

funccall

getcontext (Get thread context)

getd3dstate (Get Direct3D state)

getextcontext (Get extended thread context)

getfile (Download file)

3944+ `getfile name=STRING` 🔒 -

Retrieve the entire contents of the named file. The received data is prefixed with a 32 bit little endian value, which contains the number of bytes which have been read.

3944+ `getfile name=STRING offset=DWORD size=DWORD` 🔒 -

Retrieve `size` bytes starting at `offset` from the named file. The received data is prefixed with a 32 bit little endian value, which contains the number of bytes which have been read.

getfileattributes (Get file attributes)

getgamma (Get gamma table)

getmem (Read memory)

getmem2 (Read memory)

getpalette

getpid

getsum (Generate memory checksums)

5120+ `getsum addr=DWORD length=DWORD blocksize=DWORD` 🔒 -

Generates one or more checksums from memory.

The function will return `length` divided by `blocksize` 32-bit little endian checksums for the memory starting at virtual address `addr`.

The `addr`, `length` and `blocksize` must be multiples of 8. Picking bad values can lead to crashes.

Each checksum is equal to `ReverseBitOrder(CRC32(address + blockoffset, blocksize) XOR 0xFFFFFFFF)` for the respective block.

getsurf

getuserpriv (Get user's privilege level)

getutildrvinfo (Get utility drive information)

go

gpucount (Toggle GPU counters)

halt

irtsweep

isbreak

isdebugger

isstopped

kd (Enable/disable kernel debugger)

keyxchg (Perform key exchange)

lockmode

lop

magicboot (Boot into new title)

3944+ `magicboot title=STRING debug` 🔒 -

Boots into another title, specified by the path to XBE in `title`.

If the optional `debug` is provided, XBDM will remain loaded while the title is running.

memtrack

mkdir (Create directory)

mmglobal

modlong

modsections

modules

nostopon

notify

notifyat

pbsnap

pclist (List performance counters)

pdbinfo

pssnap

querypc (Query performance counter)

reboot

rename (Rename file)

resume

screenshot (Take screenshot)

sendfile (Upload file)

servname

setconfig

setcontext (Set thread context)

setfileattributes (Set file attributes)

setsystime (Set system time)

setuserpriv (Set user's privilege level)

signcontent

stop

stopon

suspend

sysfileupd (Update system file)

systime (Get system time)

threadinfo (Get thread information)

threads (List threads)

title

user

userlist (List users)

vssnap

walkmem

writefile

xbeinfo

xtlinfo

See Also

- Xbox Neighborhood – An XDK tool that utilizes the XBDM protocols

External Links

- ViridiX – An open-source collection of Xbox debugging libraries and tools written in C#.
- xbdm-rs - An open-source XBDM client written in Rust.

Navigation menu

Personal tools

- Create account
- Log in

Namespaces

- Page
- Discussion

Variants

Views

- Read
- View source
- View history

More

Search

Search xboxdevwiki Search Go

Navigation

- Main page
- Recent changes
- Random page
- Help

Tools

- What links here
- Related changes
- Special pages
- Printable version
- Permanent link
- Page information

- This page was last modified on 16 February 2019, at 16:09.

- Privacy policy
- About xboxdevwiki
- Disclaimers

-