Habitat Version Update

The Habitat C64 system is set up to allow the host to selectively update the object (data) disk. It updates the disk on a track/sector basis. In other words, the smallest unit that it deals with is a C64 disk block. To update the disk, the host sends the asynchronous request to the C64:

<REGION> <UPDATE> <TRACK> <SECTOR> <data...>

Where <REGION> is always noid 0, the <UPDATE> request byte is 11, <TRACK> and <SECTOR> are the track and sector numbers, and <data...> is 256 bytes to be written onto that sector.

Each Habitat data disk has a version number, and this version number is attached to the end of each DESCRIBE request that the C64 sends to the host. Thus the host should know at all times exactly what version of the object software the C64 is running. The host attempts to maintain each user's disk in the most up-to-date state. When a user logs in with an out-of-date object disk, the host should be able to detect this condition with the first DESCRIBE request that it receives. It should then send a series of UPDATE messages to the C64 before responding to the DESCRIBE request. This series of UPDATE messages should bring the user's disk up to the current version. The sector of the disk that contains the version number should always be the very last sector UPDATEd, so that if the series of UPDATE messages is interrupted for any reason, the host will simply recognize the player's disk as out-of-date the next time he logs in and attempt to update him again.

In order to minimize the number of UPDATE messages that need to be sent, the host must keep a record of the differences between each past version and the most up-to-date one. This can simply be an ordered list of the sectors to transmit to update from Version X to the present version. Note that we must keep an update list that brings each obsolete version up-to-date individually. It is not desirable to update from Version X to Version X+1, and then to update from Version X+1 to Version X+2, and so on, because this will almost certainly result in the transmission of (potentially) large numbers of useless UPDATE messages.

For example, let's assume that the present version is Version 5, and that the following differences exist between successive versions:

Version 1 to Version 2:	sectors 4-8, 11, 17	(8 new sectors)
Version 2 to Version 3:	sectors 5-7, 9-11	(6 new sectors)
Version 3 to Version 4:	sectors 6, 7, 10, 18	(4 new sectors)
Version 4 to Version 5:	sectors 8, 11-12, 18	(4 new sectors)

Then if a user logs in with disk Version number...	...send him sectors	
1	4-12, 17, 18	(11 UPDATES)
2	5-12, 18	(9 UPDATES)
3	6-8, 10-12, 18	(7 UPDATES)
4	8, 11-12, 18	(4 UPDATES)
5	none, he's up-to-date	(0 UPDATES)

Doing successive version updates instead of folding the changes together would result in:

22 UPDATE messages for a Version 1 disk (instead of 11)
14 UPDATE messages for a Version 2 disk (instead of 9)
8 UPDATE messages for a Version 3 disk (instead of 7)

Clearly we need to keep a simple data structure around in the host to keep track of the version deltas. However, it isn't a complicated thing to figure out. We could either keep the version-to-version changes and generate the update lists dynamically, or we could keep the update lists and just revise them with each new version. The latter is more work for the system managers, the former is more work for the computer. Also, of course, the host needs to keep a copy of the object disk image for the most recent object disk. We can provide this in whatever format is most suitable to you.