Technical Documentation

Marantz DH9300 XiVA™-Link
Protocol Spec. version 1.02

Jon Green and Richard Shaw, Imerge Ltd.
Revision 0.39, 2001-10-11

Contents

Figures

Tables

Revision history

Please see the end of this document for a detailed revision history.

Related documents

Tag	Full Title	Location	Description
[AppGuide]	XiVA™-Link Applications Guide	Obtain from Imerge Ltd.	[AppGuide] provides guidelines for developers working with XiVA™-Link.

Table i: Related documents

Introduction

Purpose of document

This is the Specification for the XiVA™-Link communications protocol, used to control Imerge Ltd.'s XiVAServer product line.

Intended audience

This Specification is provided to customers, OEMs, installers and other interested parties in order to assist them in producing products to control XiVAServer-based products.

Overview

The Specification documents fully but tersely each command the XiVAServer products understand. It does not attempt to provide guidance on implementation, for which purpose [AppGuide] is the best reference.

Legal notes

XiVALink is a protocol made freely available for control of XiVA based products by third party applications. Implementation of this protocol and distribution of this documentation is made on the following conditions:

1) Whilst every attempt is made to ensure the accuracy of the documentation contained here within Imerge Ltd cannot be held liable for any errors or omissions and any consequential loss of data.

2) Imerge Ltd cannot be held liable for any losses due to incorrect usage of the protocol on the part of the user.

3) Whilst every attempt will be made to ensure backwards compatibility between protocol revisions Imerge Ltd reserves the right to add, remove or amend any aspect of this protocol in future revisions.

4) Imerge Ltd cannot be held liable for the failure of any or all aspects of this protocol to work for every XiVA based product.

5) It is the responsibility of the recipient of this protocol to make themselves aware of changes and upgrades to this protocol. Where possible (if Imerge holds a current email address for the recipient) we will attempt to make you aware of new versions via email however Imerge Ltd cannot be held responsible for any failure to do this. Changes to the protocol will be published on our website (www.imerge.co.uk) by revision number.

XiVA and the XiVA logo are registered trademarks of Imerge Ltd.

All other trademarks used in this document are the properties of their respective owners, and are hereby acknowledged.

About this specification

This specification defines Imerge Ltd.'s definitions for a communications protocol between controllers
and XiVA^{TM} media servers (XiVAServer).

Release notes and urgent alerts for this revision of the Specification

Please see the revision history at the end of this document for a full list of changes: only urgent details are listed here:

Updated to final description of Version 1.02.

What is meant by version numbers and revision numbers

The protocol has a *version* number, in the form *M.mm*, where M is the major number and *mm* the two-digit minor number.

Independently of the protocol's version, this document has a *revision* number (in the form NNN), which defines the version of the document's text.

The protocol version and the document revision are independent of each other.

About the protocol

The nature of the protocol

The structure of this protocol is intended to be bi-directional and symmetric: that is to say, the general structure of the packet is the same whether it is being sent from a controller to a server, or vice-versa.

Of course, many individual commands implemented in this protocol do implicitly define the rôles of server and controller: it would be ridiculous for a server to tell a controller to $PLAY$ a track, for instance.

The protocol is intended to enforce a medium-high level of data integrity. In order to achieve this, messages can contain both a sequence character and checksum information.

Each packet sender maintains a cycle of sequence characters, sending the next sequence character with every successive packet. Replies to a specific message will include that original message's sequence character. This allows the recipient to tell whether a new packet is a resending of a previous one; it also allows the sender of a sequence of commands to work out, for a given reply, to which command it was the response. As a beneficial side-effect, a command with a sequence character can be resent, and the server will repeat the most recent reply to that command.

The style of checksum information is designed to provide a high degree of confidence in the integrity of the received data.

The protocol version number

The exact protocol is defined by a version number (*M.mm*), which has a major number (*M*) and a minor number (*mm*). The initial protocol version number is 1.00.

The major number starts at 1.

The minor number is always in the range 00 to 99, and is always expressed as a two-digit number.

The minor number must always be written in two digits, to avoid confusion. For example, version 1.02 has a minor number of 2 (or 02), whereas 1.20 has a minor number of 20. If the version was written as "1.2", it would not be obvious whether the writer meant 1.02 or 1.20.

In general, when minor additions such as extra commands are added, the minor number will be incremented, so, for example, 1.00 would increase to 1.01. An increase in the major number would indicate either a profound change in the protocols or the unlikely event of a minor number reaching, and passing, 99.

How do protocol versions work?

All servers support version 1.00.

If you want to use commands that were first introduced at a greater version number, use the $VERSION$<SUPPORT> command to find out the greatest protocol version your server supports. You will be able to use all commands defined up to and including this version number.

Once a command is defined at a given version number, that command will always be supported in that form, by servers that support at least that version. The command may be enhanced in a later protocol version, by adding new parameters or specifying new values for arguments to existing parameters, but this will not alter the basic form in which it was first published.

These are the undertakings that we make in respect of commands defined in protocol Specifications:

- Once we have defined a command, it will be honoured <u>in that form</u> in all future protocol versions. Future protocol versions may define extra parameters or different arguments that change the effect of the command. If you do not use those extra parameters or arguments, the behaviour should remain as originally specified (see next for the one exception);
- Once a reply or an update has been defined, it will always remain as originally specified <u>up to the end of the parameters specified for that form of command</u>. We may in future versions add more parameters onto the end of the reply or update, but every time we formally specify a reply or update message we “freeze” the contents of that packet up to that point;
- We may introduce new kinds of unsolicited messages at a later date, some of which may not be the direct result of an issued command.

With these in mind, when you parse messages from a server, you should understand the following rules:

- If you do not recognise a given message at all, ignore it. In future versions, we may send unsolicited messages. If you ignore any messages you do not understand, you will not introduce compatibility issues with controllers that only understand earlier protocol versions;
- Parse only as far as those parameters you expect to receive. Although we undertake not to change the format of existing parameters once they have been specified, we may add more to the end of the packet in later versions. If you are not expecting more parameters, don’t parse for them.

Glossary

Term	Means
command	This can be used to mean one of two things: either the sequence of bytes following *addressing* (see **Protocol syntax** on page 15), or the abstract concept of what they represent.
controller	Some external hardware, which communicates with the RCP (see below) using this protocol.
dynamic playlist	A set of criteria by which tracks are selected at runtime. Which tracks are played when a dynamic *playlist* is loaded depends upon which are currently available in the database, and on local storage, and how they're categorised. Dynamic playlists are <u>not</u> supported at Version 1.00 of the Protocol, but are expected to be introduced at later Versions.
entity	One of the parties communicating with each other.
media	Any media item containing one or more tracks, not necessarily all by the same artist. Media may be an album, a video with several tracks, and so on. In accordance with current usage, in this document we use "media" as both singular and plural.
media ID	(Also known as *media-id*.) Every *media* item is uniquely identified (in some opaque way) in the *server*'s databases. Certain commands can use these media IDs directly.
packet	The sequence of bytes from the start of a transmission until the terminating *end-of-line* sequence.
party	Synonym for *entity*.
playlist	Either an ordered, explicit list of items to play in order (see *static playlist*), or a set of criteria by which to select tracks to play (see *dynamic playlist*).
playlist ID	(Also known as *playlist-id*.) Every *playlist* is uniquely identified (in some opaque way) in the server's databases. Certain commands can use these playlist IDs directly.
RCP	An abbreviation for Remote-Controlled Player; the software for which this document defines the controlling protocol. This term always refers to software; never to the hardware upon which it runs or which it controls.
respondent	The original recipient of a command. In complicated exchanges, where after the original command, the respondent starts to send unsolicited responses itself (the range of update commands is one example), it is still known as the respondent.
revision	The issue number of this document. This word is never used in connection with protocol *version*s (see *version* below): it always refers to this document.
sender	The sender of the original command in a sequence. See *respondent* above.
server	A media server that is controlled by one or more *controllers*. (See *XiVAServer,* below.)

session	A sequence of XiVA-Link requests, replies and asynchronous notifications between a start-point and end-point for which the definitions depend upon the underlying transport protocol. For TCP/IP (e.g. over ethernet), the start-point and end-point are the opening and closing of the socket via which these packets will be communicated and the session is between a XiVAServer and the controller which opens the socket. For serial communications, a session currently begins when a XiVAServer receives the first request sent by a controller using a particular name or when it sends a $PING$<RESET>. The session ends with the next $PING$<RESET> or when the XiVAServer is switched off or put into standby mode.
static playlist	A fixed and ordered list of *track*s to play.
track	A single segment of audio or video, continuous from beginning to end.
track ID	(Also known as *track-id*.) Every *track* is uniquely identified (in some opaque way) in the server's databases. Certain commands can use these track IDs directly.
vDB	Virtual database. <TRACKDB>, <PLAYLISTDB> and <SPLISTDB>, for example.
version	Commands are introduced into the protocol at various versions. The initial version is 1.00. Controllers can query a server's supported protocol versions, to decide which commands to send. The protocol version should never be confused with the revision number of this document, which can change if, for example, typographic errors must be fixed, or stylistic changes made, leaving protocols unchanged.
XiVAServer	The product name for the combination of software and hardware that comprises the *server*. "XiVA" is a registered trademark of Imerge Ltd.

Table ii: Glossary

Protocol syntax

Introduction

This section formally describes the format of a packet. If you are not comfortable with formal grammars, you may wish to skip it (coming back to it for occasional reference), and go straight on to **The same again, in English**, on page 18.

Definitions

This document uses a variant on Extended Backus-Naur Representation to define protocols. In this representation, all white space is declared explicitly:

Symbolism	Means
Name	The symbolic name of a component of the protocol.
‘b’	The literal character b (the ASCII character 98 (decimal)).
‘#42’	The ASCII character with value 42 (decimal), which is ‘*’.
‘#x42’	The ASCII character with value 42 (hexadecimal), which is ‘@’.
‘a’…’z’	One character in the range ‘a’ to ‘z’ inclusive.
“abc”	A character sequence: ‘a’, then ‘b’, then ‘c’.
name1 | name2	Either name1 or name2, but not both.
[name]	Zero or one occurrence of ‘name’.
name*	Zero or more occurrences of ‘name’.
name+	One or more occurrences of ‘name’.
{name1 name2}	The same as ‘name1 name2’, but allows repetition operators such as *, + and so on to affect groups of symbols.
1{name}3	From 1 to 3 (inclusive) occurrences of ‘name’.
allow-set ~~ deny-set	Everything in allow-set, less anything in deny-set (for example, {‘a’ … ‘z’ ~~ { ‘p’ | ‘q’ }} would mean all the characters in the range ‘a’ to ‘z’, excepting ‘p’ and ‘q’ }).
kids ::= jane john	‘kids’ is defined as shorthand for “whatever ‘jane’ means, followed immediately by whatever ‘john’ means, with no intervening white space”.

Table iii: Protocol syntax definitions

Description

Version 1.00 packets are constructed as follows:

```
Packet                      ::= addressing command parameters checksum end-of-line

addressing                  ::= source-id destination-id message-sequence-char

source-id                   ::= ‘#’ identifier ‘#’
destination-id              ::= ‘@’ identifier ‘@’
message-sequence-char             ::= sequence-char
identifier                  ::= 1{alphanum}20

command                     ::= ‘$’  command-word ‘$’ [reply-sequence-character]
command-word                ::= 1 { upperalphanum } 10
reply-sequence-char         ::= sequence-char

parameters                  ::= parameter*
parameter                   ::= param-name [ argument ]
param-name                  ::= ‘<’ param-word ‘>’
argument                    ::= standard-argument [ localised-argument ]
standard-argument           ::= value-details
localised-argument          ::= ‘%’ value-details
value-details               ::= { { { ‘#32’ … ‘#126’ } ~~ delimiter-char } | escape-sequence }+
escape-sequence                   ::= ‘\’ { escape-char | escape-hex }
escape-char                 ::= ‘0’ | ‘n’ | ‘r’ | ‘t’ | delimiter-char
delimiter-char              ::= ‘@’ | ‘#’ | ‘$’ | ‘%’ | ‘<’ | ‘>’ | ‘\’ | ‘~’
escape-hex                  ::= ‘x’ hex-byte
param-word                  ::= 1 { upperalphanum } 12

checksum                    ::= '~' check1 check2
check1                      ::= hex-byte
check2                      ::= hex-byte

end-of-line                 ::= ‘#13’ ‘#10’

sequence-char               ::= alphanum
hex-byte                    ::= most-significant-hex-digit least-significant-hex-digit
most-significant-hex-digit        ::= hex-digit
least-significant-hex-digit       ::= hex-digit

upperalphanum                     ::= upper-alpha | numeric
alphanum                    ::= { upper-alpha | lower-alpha | numeric }
hex-digit                   ::= { upper-hex | lower-hex | numeric }

upper-alpha                 ::= ‘A’ … ‘Z’
lower-alpha                 ::= ‘a’ … ‘z’
upper-hex                   ::= ‘A’ … ‘F’
lower-hex                   ::= ‘a’ … ‘f’
numeric                           ::= ‘0’ … ‘9’
```

The overall packet length, including *end-of-line*, should not exceed 1024 characters.

Version 1.01 packets

As above, except:

addressing	::= source-id destination-id [message-sequence-char]
checksum	::= '~' [check1 [check2]]

The same again, in English

The source-id

The packet starts with a **source-id**, which identifies the sender. This may be no more than 20 alphanumeric characters, enclosed in a pair of hash (‘#’) characters. The command originator is responsible for deciding its own *source-id* name. For controllers, it is a good idea to pick a *source-id* that is not likely to conflict with other devices controlling the same server at the same time.

The destination-id

Immediately following that is the **destination-id**. This is in the same format as the *source-id*, but delimited by ‘@’ characters instead. The *destination-id* uniquely identifies the intended recipient of the message. This may refer to a single unit (the server in general), or the name given to a logical destination (for example, the message could be a play command to the seventh output source of the media server). The exact meaning of the *destination-id* is a matter for negotiation between the communicating parties, or their respective programmers.

The *destination-id* `server` is reserved, and is the first destination with which a controller should communicate, in order to gather information on other possible destinations.

The *destination-id* `broadcast` is also reserved, and is used (only) in the $BROADCAST$ command to indicate that the message is intended for all destinations capable of receiving it.

Sequence characters

Following the source and destination IDs is the **message-sequence-char**. This is the sequence character for <u>this</u> message, and it is of the type **sequence-char**. *Sequence-char*s cycle from ‘0’ to ‘9’, then ‘A’ to ‘Z’, then ‘a’ to ‘z’, then back to ‘0’ again, indefinitely. There is no obligation upon users of this protocol to require their first packet to have a sequence-character of '0': it can begin anywhere in the cycle, providing it follows the sequence outlined here.

Successive packets that are sent by the same sender should be given successive *message-sequence-character*s according to this scheme, except where the sender is re-transmitting a packet for some reason.

At version 1.00, the *message-sequence-char* is mandatory.

For servers supporting version 1.01 onwards, the *message-sequence-char* is optional; however, its use is very strongly recommended. Without it, the server cannot tell if a second command identical to the first is a repeat of that command or a new command in its own right: behaviour may become unpredictable in some circumstances. Furthermore, without it the sender has no way of telling which reply corresponds with which packet it sent. This typically means that a sender must wait for a reply before it can send the next packet. This can slow operations.

The command

The **command** is a sequence of up to nine alphanumerics, delimited by ‘$’ characters.

If the *command* is a reply to a previous packet received, and that packet contained a *message-sequence-char*, the second ‘$’ character is followed by a **reply-sequence-char**. This is a copy of the *sequence-char* of the packet to which it is a reply.

The command’s parameters

The command may be followed by one or more **parameter**s.

Each **parameter** consists of an angle-bracket-delimited alphanumeric sequence (which may include spaces but may not begin with one), followed by an optional **argument**.

Each parameter’s optional argument

The optional *argument* begins with a **standard-argument**, which is what the Protocol formally defines the argument to contain.

The *standard-argument* may optionally be followed by a percentage sign (‘%’), and a **localised-argument**. This is a version of the same argument in the user’s local language, where applicable (for example, where there are localised versions of the error messages for display to users). More of this in a while.

Both *standard-argument* and *localised-argument* are **value-details**. These carry certain constraints upon the characters they can contain. Specifically, they may never contain any of the characters from the **delimiter-chars** set, with the one exception of ‘\’, where it’s used to introduce an **escape-sequence**.

The *escape-sequence* is the means by which characters not normally allowed in *value-details* may be included, by describing them in a special way.

An *escape-sequence* consists of a backslash (‘\’) followed by a special sequence of characters. These are the defined sequences:

ASCII characters	ASCII values (decimal)	How to express them
	0 to 31	**\x*NN*** *NN* is the two-digit hexadecimal for the ASCII value. For example, “\x0d” would represent the CR (carriage-return) character, whose ASCII value is 13, or 0d in hexadecimal. You can use either upper- or lower-case for the A…F hexadecimal characters. Some common characters in the ASCII range 0…31 can be expressed using shorter *escape-sequences*: see below.
NUL	0	\0 A backslash, followed by a single zero.
TAB	7	\t
LF	10	\n
CR	13	\r
@ # $ % < > \ ~	Various	*Precede the character with a backslash* For example, to escape the backslash character, precede it with another (“\\\\” instead of “\\”). The tilde character would be escaped thus: “\\~”.
	128 to 255	\xNN Exactly as for the range 0…31, above.

Table iv: Escape sequences

So, for example, if you wanted to put into an *argument* the phrase “15% of $50 is $7.50”, with a carriage return and linefeed following it, you would encode it as:

```
15\% of \$50 is \$7.50\r\n
```

The *localised-argument* is entirely optional, and is reserved for a version of the first string in the user’s local language.

For example, on a French system a command to obtain the play state of zone 1 (in this case, *destination-id* “Z01”), and its reply, might look like this:

```
#ctrlr#@Z01@1$STATUS$<MODE>~223b

#Z01#@ctrlr@t$ACK$1<OK><MODE>PLAY%JOUE~0f6d
```

(Don’t forget, both packets are completed by CR (ASCII 13) and LF (ASCII 10).)

Note: servers supporting protocols up to version 1.02 do not yet generate *localised-argument* information, but this facility may be implemented in later versions. It is documented here so that controller designers can be forewarned.

The checksum

At version 1.00, the **checksum** is a tilde ('~'), followed by a pair of ASCII hex digits (**check1**), itself followed by a second pair of ASCII hex digits (**check2**).

At version 1.01, the tilde may be followed by:

- *check1*, followed by *check2* (four ASCII hex digits in total)*;*
- *check1 only* (two ASCII hex digits in total)*;*
- *nothing.*

Check1 and *check2* encode two different types of checksum data. These are calculated over every byte in the packet up to (and including) the tilde. *Check1* and *check2 e*ach represents 8 bits of checksum data.

The *check1* checksum is the low eight bits of the sum of those ASCII values. Here is some C code that calculates this value:

```
unsigned char
CalcCheck1(const unsigned char *packet, int index_of_tilde)
{
      unsigned char check1 = 0;
      int index;

      for (index = 0; index <= index_of_tilde; index++)
            { check1 += packet[index]; }
      return check1;
}
```

Figure 1: C code to calculate *check1*

The *check2* checksum is calculated as follows: start with an eight-bit value of zero (0x00). For every successive byte being summed from first to last, exclusive-or that byte into our value, and then rotate left by one bit position (i.e. the previous most significant bit becomes the new least significant bit, and all other bits shift one place towards the most significant bit).

Here is some C code that calculates the *check2* value:

```
unsigned char
CalcCheck2(const unsigned char *packet, int index_of_tilde)
{
      unsigned char check2 = 0;
      int index;

      for (index = 0; index <= index_of_tilde; index++)
      {
            unsigned char overflow;

            check2 ^= packet[index];
            overflow = ((check2 >> 7) & 0x01);
            check2 = (check2 << 1) + overflow;
      }
      return check2;
}
```

Figure 2: C code to calculate *check2*

If you are communicating at version 1.01 or greater, then in order to be able to be confident in packet integrity, we strongly recommend that you use at least *check1* in the packets you generate; preferably also *check2*. If you are communicating at version 1.00, you <u>must</u> include both. The RCP will always generate both checksums in the packets it sends.

Without checksums, parts of packets can go missing. <u>This is a serious risk.</u> Under some rare circumstances, particularly whilst ripping operations are in progress, RS-232 communications may become unreliable. The server relies upon checksum verification to ensure that packets it receives are intact. Equally, any controller you write should verify at least *check1*, to have a degree of confidence in the messages it receives.

And finally…

The *checksum* is followed by **end-of-line**, which is a carriage-return (ASCII 13, or CR) followed by a line feed (ASCII 10, or LF). These <u>must not</u> be escaped with backslashes!

A typical packet

This is a typical reply:

`#server#@ctlr@a$ACK$3<OK>~4f24` (followed by CR and LF, of course)

The table, below, breaks down this packet into its individual bytes. The separate parts of the packet have been shaded for clarity:

ASCII	Char. value		Comments
	Dec.	Hex.	
#	35	0x23	Delimits the *source-id*.
s	115	0x73	The *source-id* (`server`, in this example) – the ID of the entity sending the packet.
e	101	0x65	
r	114	0x72	
v	118	0x76	
e	101	0x65	
r	114	0x72	
#	35	0x23	Delimits the *source-id*.
@	64	0x40	Delimits the *destination-id*.
c	99	0x63	The *destination-id* (`ctlr`, in this example) – the ID of the entity to which the packet is sent.
t	116	0x74	
l	108	0x6c	
r	114	0x72	
@	64	0x40	Delimits the *destination-id*.
a	97	0x61	The *message-sequence-char*.
$	36	0x24	Delimits the *command*.
A	65	0x41	The *command* (`ACK`).
C	67	0x43	
K	75	0x4b	
$	36	0x24	Delimits the *command*.
3	51	0x33	The *reply-sequence-char* – this would have been the *message-sequence-char* in the packet that provoked this reply.
<	60	0x3c	Begins a *parameter*.
O	79	0x4f	A parameter (`OK`).
K	75	0x4b	
>	62	0x3e	Ends a *parameter*.
~	126	0x7e	Introduces the *checksum*.
4	52	0x34	The *check1* checksum.
f	102	0x66	
2	50	0x32	The *check2* checksum.
4	52	0x34	
(CR)	13	0xd	The *end-of-line* sequence.
(LF)	10	0xa	

Table v: Sample packet breakdown by byte content

Notes on the command descriptions

Where a list of parameters is given, this list is order-sensitive. In other words, whilst some of the parameters may be optional, those that are supplied should be supplied in the exact order shown.

In the replies shown for each command, the parameters are also order-sensitive. After the end of the parameters described in the Specification, you may receive some additional parameters. If you do, ignore them. They apply to controllers that understand a protocol version later than described here.

NOTE: for ease of display, we omit the *addressing*, *reply-sequence-char*, *checksum* and *end-of-line* parts in most of the examples we show further on in this specification. Lines may also be broken at convenient places for display. Please assume that newlines and white space are for display purposes only, unless they have explicitly been described as being part of a packet.

Commands – general principles

All packets in version 1.00 are limited to a maximum size of 1024 bytes, including the final *end-of-line* sequence.

In general, the immediate reply to any successfully received command is the ACK command. The parameter will be one of the following:

Parameter following ACK	Means
<RXD>	Message received, but will take some time to process. Expect a further response.
<OK>	Message received, and accepted. This will typically be followed by more parameters that are specific to the command to which this is a reply.
<WARNING> <MESSAGE>*XXmessage*	Message received, and accepted, but there are conditions of which the user must be made aware. *XX* is a two-digit hexadecimal error code (*hex-byte*) pertinent to the command; *message* gives an optional, user-friendly description of the problem. A '%' and a localised string may follow it.
<ERROR> <MESSAGE>*XXmessage*	Message received, but could not be executed. *XX* is a two-digit hexadecimal error code (*hex-byte*) pertinent to the command; *message* gives an optional, user-friendly description of the problem. A '%' and a localised string may follow it.

Table vi: ACK parameters

NOTE: *the sample error messages (as opposed to error codes) given in worked examples later on in this protocol are not fixed, and are only shown for demonstration purposes. The actual messages could, and probably will, differ dramatically from the ones shown.*

All commands can cause an error response, and many can also cause warnings. In the command descriptions that follow, only the successful responses have been detailed, unless there are additional parameters for errors or warning raised by that command.

You should receive an ACK within 5 seconds of completing transmission of the corresponding command. If it is not received within that time, the sender should $PING$ the respondent (see page 30) to check whether the respondent is still alive. If it is, the packet should be resent exactly as previously (i.e. using the same sequence character as in the original).

If your respondent cannot complete the command within this time frame, it should send you ACK<RXD>, using the original message's *message-sequence-char* (if present) as its *reply-sequence-char*. This indicates to you that there will be an indefinite delay before the command's processing is completed.

If after three attempts at sending, the packet is still not acknowledged, you should assume communications have been compromised, or that its command cannot be accepted for some reason (for example, it is badly formatted or addressed). If this happens, $PING$ the destination more than once. If it responds, there is something wrong with the original command, or a problem with the server's software. Either way, the command will not be accepted and you should abandon attempts to send it.

If you send a packet with an invalid destination to the server, you will receive a `1f` error reply (no such destination). The reply will appear to come from the incorrect destination: for example, a packet (omitting checksum and sequence characters) containing:

```
#someone#@noone@$PING$
```

…will receive this reply, if there is no “`noone`” *destination-id*:

```
#noone#@someone@$ACK$<ERROR><MESSAGE>1fNo such destination
```

If a packet is structured correctly, but the command or its parameters are not recognised by the server, you will receive this reply:

```
$ACK$<ERROR><MESSAGE>1eSyntax error
```

A packet that does not satisfy the basic packet format will simply be ignored.

If the controller does not respond at all to $PING$s, assume a complete communications breakdown. If you have the facility to do so, back down to communications defaults (e.g. RS232 speeds and settings). Send $PING$ commands to the `server` destination until you get a reply.

If your controller has had to do this, it should probably assume that the server has been restarted, and any transient settings have been lost. Once it has re-established communications, it should go through its own cold-start sequence, sending whatever requests it would normally send in order to put the server into a state it expects.

Error and warning codes

A number of error and warning codes are common to most commands. Not all of these are used for every command, of course. The explanation string which follows the error or warning code is arbitrary, and for display purposes. It may be followed by a '%' and a localised version of the string.

XX=	Means
00	Hardware problem
01	No media cued to play.
02	Can't accept that value.
03	No media ready to play (i.e. no track is cued, and no useful default is set).
04	Message was corrupt and has been ignored, please resend.
05	Name is not unique (for named saves, such as playlist saves).
06	Wrong database given.
07	Wrong destination. You either sent a message to 'server' that can only be handled by a playout destination, or vice-versa.
08	No playout server available. The RCP could not find a XiVAServer to control. This may go away if you retry the message a few times; the XiVAServer is usually invoked from a script that restarts it if it ceases to run.
09	No media database. This is a fatal error; the RCP cannot recover from this problem.
0a	[reserved]
0b	[reserved]
0c	Timed out communicating with the XiVAServer
0d	Out of memory
0e	Device busy
0f	No such device
10	Not online (attempt to perform an online operation without a TCP/IP connection active)
11	No such search tag
12	No such search database
13	No such ID
14	No such encoding
15	Lookup failed
16	Cache marker no longer valid
17	Network Manager unavailable
18	Network Manager error
19	License Manager unavailable
1a	Missing binary (program)
1b	File error
1c	Read-only setting
1d	Write-only setting
1e	Syntax error
1f	No such destination ID
20	Network configuration failure
21	Network connection failure
22	Network disconnection failure
23	Track not available
24	Media has no available tracks
25	Playlist has no available tracks

Introduced in Protocol version 1.02	
26	Operation not permitted
27	Operation failed

Table vii: Common error codes (<ERROR><MESSAGE>XXmessage)

(Errors 01 and 03 are confusingly similar. Expect one or the other to be revoked and replaced with one or more, better described, errors.)

XX=	**Means**
80	No current state – used when (1) trying to make a change to an unset state, or (2) commit an unset information state to a playing state.
81	Media unavailable (no media file for that track). No effect on server
82	No more media – like 81, but there were no files for any tracks until the end of the media or playlist. The state will be unaffected.
83	Used in wrong play mode – for example, a $SELECT$ by media when the server is playing from a playlist , or vice-versa.
84	Attempt to skip to before start or after end of track.
85	Already in that mode; no change (e.g. already playing when the $PLAY$ command came in). Do not rely on this error; often the RCP will silently ignore duplicate requests of this type rather than complaining.
86	No such track exists.
87	Already idle.
88	Only partial success [note: this may be withdrawn or replaced]
89	Aborted by user
Introduced in Protocol version 1.02	
8a	Superseded
8b	Redundant

Table viii: Common warning codes (<WARNING><MESSAGE>XXmessage)

Version control

All production servers should support requests available in protocol version 1.00 (though they may return more information than specified in that version). If you wish to use commands or a packet format from a later protocol version number, you should first query the server's highest protocol version, to ensure that it can understand your request.

Requesting available versions

Request

Command:	$VERSION$
Parameters:	<SUPPORT>

This command requests the version numbers supported by the respondent.

Reply

Command:	ACK
Parameters:	<OK>
	<SUPPORT>*MM.mm*

The reply gives the highest protocol version number the server supports. Implicitly, it also supports every previous version, back to 1.00.

Pinging

The commands in this section are all related to "pinging" operations – ones in which one entity is attempting to contact another and confirm that it is (still) responding to commands.

Simple ping

This command must be supported by all protocol clients. In other words, all protocol clients, whether server or controller, must respond to this command.

This is just a means by which one entity can check that another is responding. The very fact of receiving a reply indicates a success.

This command can be sent to any *destination-id*. If you receive a reply, it indicates that that destination is functional and responding. (This doesn't necessarily mean that other destinations served by the same connection are also functional, although it does indicate that at least part of the whole system is working.)

If you do not receive a reply within a reasonable amount of time (say, five seconds), send it again, up to three times in total. Particularly in the case of serial communications, the recipient may receive only part of the first packet successfully, so a first $PING$ may simply serve to put the message buffer in a state where it will correctly receive the second one.

Request

Command:	\$PING\$
Parameters:	(none)

Reply

Command:	\$ACK\$
Parameters	<OK>

Starting a new session

The server stores various items of state information for the duration of a communication session. The $PING$<RESET> request is provided to allow a controller to indicate to the server that it is starting a new session and thus does not want the server to continue to store or use any state information from any previous communication session. It is particularly important for a controller connected by RS-232 to send this request when it restarts as the server cannot easily detect this transition by other means.

As of version 1.02 of the Protocol, the $PING$<RESET> request causes the server to :-

- treat all subsequent requests as belonging to a new session (Normally the server “remembers” the previous few commands that any controller sends, so that it can resend lost replies if it sees the same command with the same *message-sequence-char* a second time, rather than processing it as a new request.)
- use default communication settings (see **Configuring communication** on page 32) rather than any in effect before the $PING$<RESET>, unless and until the settings are changed in the new session
- cancel any updates (see **Update notifications** on page 66) requested by the controller before the $PING$<RESET>
- prevent the controller from seeing any replies to any requests sent before the $PING$<RESET>.

In version 1.01 of the Protocol, $PING$<RESET> had only the first effect (clearing the command history) out of those listed above.

This command may be sent to any valid *destination-id* but `server` is recommended. It will clear any information about the previous session regardless of the destination(s) with which that information was stored.

Request

Command:	$PING$
Parameters:	<RESET>

Reply

Command:	ACK
Parameters:	<OK>
	<RESET>

Assuming that the server receives and can reply to the message, this command always succeeds. If you get no reply within five seconds, send it again (with a new *message-sequence-char*, if possible).

Communications

Ethernet

To establish an ethernet connection with the server, open a socket on the server's port 6789. You can then send commands and receive replies using this socket.

Serial clients, front panel managers and so forth all use this interface to communicate with the RCP.

RS232 notes

Communications using RS232 connections are one-to-one: that is to say, there is no hardware "daisy-chaining" at all. It is perfectly allowable to use a "serial concentrator" to connect more than one client to a serial port. If you do, bear in mind that each such client connected to a concentrator must have its own *source-id*, and the concentrator must be responsible for routing replies to the correct client based upon the *destination-id* in the reply.

At present, the serial speed supported at a given serial port is a server configuration setting, and not changeable by software.

NOTE for developers working to previous revisions of this Specification: the $RS232$ commands are no longer supported, although they may be reintroduced in a later version.

Configuring communication

[Introduced in Protocol version 1.02]

The default communications behaviour of a XiVAServer may not be appropriate for all controllers. Some may not be able to cope with reply packets of the default maximum length (1024 bytes) for the XiVA-Link protocol. Alternatives to the default escaping mechanism (see **Table iv: Escape sequences** on page 20) for dealing with reserved characters, non-ASCII characters and control characters within ASCII may be preferred. In addition, new character encodings may be added in future versions.

The purpose of the $COMMS$ command is to allow attributes of the communication between a XiVAServer and a particular controller to be set and queried. Changes of these attributes from the default value affect communication only with that controller and persist only for the duration of the session (see **Glossary**) in which they are made[1]. This is in contrast to configuration of the server (see **Configuring the server** on page 149) which affects the response of the server to all controllers and is (in general) persistent even between reboots. Also, although a similar syntax is used, this command can only be sent to the `server` destination.

[1] Strictly speaking, it is currently possible for a controller communicating with a XiVAServer via a serial port to inherit a session initiated by a controller previously connected to the same serial port and using the same source address. New session management requests are expected to be included in the next version of the Protocol.

The configurable *items* are grouped within *categories.* The current categories and items and their effects on communication are shown in Table ix : Configurable communication items. Where possible values are shown, the default value is shown in bold.

Category	ItemName	Effect
Encode	Type	The character encoding used in reply packets :- • **Latin1** : ISO 8859-1 encoding is used • ASCII : ISO 8859-1 characters in the ranges 0 – 31 and 127 - 255 are mapped to ASCII characters in the range 32 - 126 (e.g. accented letters are mapped to the corresponding letters without accents). • ASCIInd : As for ASCII, except that there are **n**o (XiVA-Link) **d**elimiters, all of them having been mapped to other ASCII characters.
Escape	Type	This specifies the type of escaping applied to characters resulting from the above encoding :- • **Hex** : as detailed in Table iv: Escape sequences on page 20 • None : no escaping is applied to characters other than XiVA-Link delimiter characters (which here includes carriage return, new line and tab).
Pkt	MaxLen	The maximum length (in the range 66 – **1024**) for the reply packet to one of the alternative requests for small-input-buffer controllers (see page 157).[2]

Table ix : Configurable communication items

The six possible combinations of the current Encode/Type and Escape/Type values result in only four distinguishable transformations of the original characters. Once Encode/Type has been set to ASCII or ASCIInd, none of the resulting characters are subject to Hex encoding, so Escape/Type will currently be irrelevant.

Note: The combination of an Encode/Type of Latin1 and an Escape/Type of None implies that an 8-bit clean communication channel is available. Thus it can be used via ethernet but not with the current serial communication configuration.

[2] Future versions of the protocol may guarantee this maximum length for all reply packets; the alternative requests provide replacements for most of the commonly used requests which are likely to contain multiple value strings and hence exceed the requested MaxLen.

Setting communication values

[Request (and response) introduced in Protocol version 1.02]

This allows a controller to set the value of *item* within *category* to *new-value*.

Request

Command:	$COMMS$
Parameters:	<SET>*category*
	<ITEM>*item*
	<HAS>*new-value*

Reply

Command:	ACK
Parameters:	<OK>
	<COMMS>
	<SET>*category*
	<ITEM>*item*
	<HAS>*new-value*

Querying communication values

[Request (and response) introduced in Protocol version 1.02]

This allows a controller to read the value of *item* within *category*.

Request

Command:	$COMMS$
Parameters:	<READ>*category*
	<ITEM>*item*

Reply

Command:	ACK
Parameters:	<OK>
	<COMMS>
	<READ>*category*
	<ITEM>*item*
	<HAS>*value*

Enumerating destinations

A given entity can enumerate the possible destinations it can command. It should always send its request to the destination-id `server`. If no reply is received within one second, it should assume that the server is absent or unwell.

At present, playout destinations are “named” Z01, Z02 and so on. Developers should not rely upon this behaviour! In future, it is likely that we will allow and use more “human-friendly” names for playout zones.

Polling for destinations

Request

Command:	WHO
Parameters:	<DESTINATION>

Controllers use this command to find out the available destinations on a server. Servers must always respond to WHO<DESTINATION> requests with *destination-id* `server`; controllers must never do so unless they also provide server capabilities.

Reply

Command:	ACK	
Parameters:	<DESTINATION>*destination-id*	One or more of these

The reply will contain one or more <DESTINATION> entries.

The *destination-id* values are supplied without '@' delimiters. The *destination-id* `server` will always be listed; all other *destination-id*s may be presumed to be playout destinations.

Example

A typical exchange would be:

Sender:	WHO<DESTINATION>
Respondent:	ACK<OK><DESTINATION>server<DESTINATION>room1 <DESTINATION>room2

The controller could then send packets with a *destination-id* of `@room2@`.

Polling for other entities

This command <u>must</u> be supported by protocol clients (other than the server).

Request

Command:	WHO
Parameters:	<LISTENING>

This command is for use by servers seeking to enumerate controllers on a common bus (for example, Ethernet or GPIB), using some kind of broadcast mode.

Reply

Command:	ACK
Parameters:	<LISTENING>*source-id*

On common buses, the server should expect to receive more than one response (in this form) to this command. It should listen for <u>five</u> seconds, not the usual one second, since bus collisions and retries may cause delays in the receipt of some messages.

Duplicate responses with the same *message-sequence-char* should be assumed to be from the same source, and duplicates ignored.

Duplicate responses with differing *message-sequence-char*s will be from different sources that have unfortunately chosen the same *source-id*. How the server deals with this situation is beyond the scope of this protocol, as it will depend upon how the controllers are connected to the server, and considerations such as whether the server can present a user-interface. It is quite possible that the server will broadcast a message for display (see next) on the offending controllers.

Broadcasts and alerts

Broadcasts are unusual in that the recipients should not attempt to ACK them. Alerts are one-to-one messages and require an acknowledgement.

Typically, servers use broadcasts, and controllers use alerts, but both servers and controllers must honour both types of message.

Broadcast

Request

Command:	$BROADCAST$
Parameters:	<MESSAGE>*message*

If the *destination-id* of this message is a specific name, then this message is targeted at a specific *destination-id*.

If the *destination-id* of the message is `@broadcast@`, then it is intended for all destinations capable of receiving the message.

The purpose of this command is to give the destination(s) a message to put on their displays, for the user's attention.

Reply

None. The sender of this message neither knows nor cares whether it reaches its destination(s).

Alert

This command must be supported by <u>all</u> protocol clients (other than the server).

Request

Command:	$ALERT$
Parameters:	<MESSAGE>*message*

This is similar to $BROADCAST$, but it is targeted at one destination, and requires an acknowledgement. If possible, the destination should display the message. If it cannot, it should raise a warning.

Reply

Command:	ACK
Parameters:	<OK>

Simple play commands

The server is said to be in one of three modes: play, pause or stop. Simple play commands alter the current mode, or how it operates.

Basic play

Request

Command:	$PLAY$
Parameters:	(none)

Play is an imperative command: it should be acted upon as soon as it is received. The currently cued track will be played. This may be the first on a media, or in a playlist, or a previously chosen track from some selection. If, however, playout has stopped as a result of the end of the currently selected item being reached, then a new item must be selected or the playout position moved back within the item (c.f. $PLAY$<SKIP> and $SELECT$<TRACK>) before a further $PLAY$ request will succeed. $PLAY$ will also restart a paused (see later) track from the point at which playback had been paused.

Note for early implementers: if the $PLAY$<RATIO> command (mentioned in Specifications prior to version 1.00) is reintroduced, $PLAY$ will also cancel the effects of a $PLAY$<RATIO> command in progress.

Reply

Command:	ACK
Parameters:	<OK>

Set play flags

Request

Command:	$PLAY$	
Parameters:	<FLAG>	
	<RANDOM>*offon*	Optional – *offon* is OFF or ON
	<REPEAT>*offon*	Optional – *offon* is OFF or ON

One or both of <RANDOM> and <REPEAT> must be present.

This sets the play states for the playout destination to which the command is sent. Any changes take effect after the end of the track currently playing (if any), and persist until explicitly changed again.

The <RANDOM> flag causes the tracks from the current media or playlist to be played in an arbitrary order.

The <REPEAT> flag causes the server to restart play from the beginning of the track, media or playlist currently selected, once it has reached its end.

Reply

Command:	ACK
Parameters:	<OK>

Stop

Request

Command:	$STOP$
Parameters:	(none)

This stops playback and puts the player in STOP mode with immediate effect. STOP mode can always be honoured. If the server is presently playing, or in PAUSE mode, STOP mode has the effect of putting the play position back to the start of the present track.

Reply

Command	ACK
Parameters:	<OK>

Pause

Request

Command:	$PAUSE$
Parameters:	(none)

Pause is an imperative command: the server must honour it immediately if it can. It is possible to put a server into PAUSE mode even if it is not presently playing, provided that a track is currently selected. If the server has no track currently selected, an error will be raised.

The effect of PAUSE mode is to halt playing immediately (if in PLAY mode), and maintain the current position in the current track.

Note that pause is not a toggle. Use the $PLAY$ command to resume playing from the paused point.

Reply

Command:	ACK
Parameters:	<OK>

Skip within track

Request

Command: $PLAY$
Parameters: <SKIP>
then either:

<REL>*nnn*	*nnn* is mandatory; can be negative
or:	
<ABS>*nnn*	*nnn* is optional; zero or positive; default 0

This command sets the play position within a track, and has two forms: relative or absolute.

The parameter for the <REL> form is the number of seconds to skip from the current playing position. It is mandatory, and may be negative if the skip direction is to be negative.

The parameter for the <ABS> form is optional (default 0), and represents the number of seconds from the start of the track.

The <REL> and <ABS> parameters are mutually incompatible. Exactly one must be given.

After having performed the skip, the server will revert to its previous play mode: if playing, it will resume playing from the new position; if paused or stopped, it will be set at the new position.

The $PLAY$<SKIP> command will not skip back beyond the start or end of the current track. If the command would take it beyond the start or end of the current track, it will position at the start or end (respectively) and raise a warning. It will raise an error if there is no selected track.

Reply

Command: ACK
Parameters: <OK>
<POS>*hh:mm:ss*
<MSECS>*MMM*

or*:* <WARNING>
<MESSAGE>*XXmessage*
<POS>*hh:mm:ss*
<MSECS>*MMM*

or: <ERROR>
<MESSAGE>*XXmessage*

Unless an error is reported, the reply includes the new track position, in hours, minutes and seconds, plus milliseconds in the <MSECS> parameter.

Selecting and examining media and tracks

This set of commands is provided in order for controllers to tell the server which media, playlist or track to load, ready for play. There are also commands to select individual tracks within media or playlists, by their index number.

Select media, playlists or individual tracks by ID

Tracks, media and playlists all have unique IDs. This command uses an ID to define the current play selection. The IDs can be obtained by the appropriate search commands (see later).

Request

Command:	$SELECT$	
Parameters:	<*ITEMTYPE*>	Optional : see below *[Introduced at version 1.02]*
	<ID>*id*	

[Introduced at Protocol version 1.02]

then optionally (and only if <ITEMTYPE> has been specified):

<TRACK>	
<NUM>*nnn*	*nnn* is the track number within the item

then optionally:

<PLAY>

<*ITEMTYPE*> may currently be <TRACK>, <MEDIA> or <SPLIST>. If it is specified, then it is an error for it not to match the type of the item referenced by *id.*

If <TRACK><NUM>*tracknum* is included and <*ITEMTYPE*> has been specified as either <MEDIA> or <SPLIST>, then that track within the album or playlist, respectively, will be selected. This assumes that the track is available. If not, the next available one (if any) will be cued instead, where `next' will include wrapping round to the first available track if REPEAT mode is set and no higher-numbered tracks are available. If RANDOM mode is set then *tracknum* is the index of the track within the shuffled set of tracks. (See **Set play flags** on page 43 for discussion of these modes.)

Including this option is theoretically equivalent to a $SELECT$<*ITEMTYPE*><ID>*id* request followed by a $SELECT$<TRACK><NUM>nnn request (see Error! Not a valid bookmark self-reference. on page 48) but has the advantage of being atomic. It removes the possibility that the server, if already in PLAY mode, will start playing the first track within the item before it skips to the requested item.

If <PLAY> is present in the request the server will start playing the selected item. Otherwise, the server's play mode is preserved: it will start playing the item only if it was already in PLAY mode.

The use of all the options added at version 1.02 has the additional benefit that it allows one request to achieve what would have required three separate requests in earlier versions of the Protocol.

Reply

Command:	ACK	
Parameters:	<OK>	
	<ID>*id*	*id* is a track-id
	<NUM>*nnn*	*nnn* is a positive integer
	<ORIG>*nnn*	*nnn* is a positive integer
	<TOTAL>*nnn*	*nnn* is a positive integer
	<LEN>*hhhh:mm:ss*	
	<TYPE>*type*	*type* is TRACK,MEDIA,SPLIST or DPLIST

The <TOTAL> parameter gives the total number of tracks in the media unit identified by the ID. (Of course, for a single track, this will be 1.)

<LEN> gives the total playout time.

<ORIG> gives the loaded track's original (<ORIG>) track number.

<NUM> gives that loaded track's track number in the prevailing track play order (which can be randomised).

Select media by media number

The media are initially numbered on the server from 1 onwards. Once media are assigned a "slot", the number of that slot will not change until the media are deleted, regardless of whether other media have subsequently been deleted or added.

Whether or not slots (media numbers) from deleted media can ever be reused is a configurable option.

The media number 0 is reserved and has a special meaning: the first media still extant on the server. So, for example, if media numbers 1-10 have been deleted at some point, selecting media number 1 will fail (no media available), but selecting media number 0 will actually cause media number 11 (the first media which the server still has) to be loaded.

Request

```
Command:      $SELECT$
Parameters:   <MEDIA>
        then either:
              <SKIP>sss                  To inc/decrement by sss media; default 1
        or:
              <NUM>nnn                   To select media number nnn
```

The *sss* argument is a signed number; *nnn* is unsigned.

This causes the server to increment, decrement or select a specific medium from the total list of media available on the server. <SKIP> and <NUM> are mutually exclusive: the command may use either exactly one, or none, of them.

If a value is supplied to <SKIP>, it must be an integer (it may be negative); if it is not, the default is 1. If a value of 0 is given to the parameter, the server will not change its state, but simply report the current media. An attempt to <SKIP> beyond the last media, or to before the first, will result in a warning and no change in state.

The effect of <SKIP> is to skip the given number of media that are still on the machine. So, if media 3-4 and 6-8 have been deleted, and you are currently playing media number 2, using <SKIP>1 will take you to media number 5; alternatively, <SKIP>2 would have taken you to media number 9.

The value for <NUM> must be a positive integer. Note that media are numbered from 1, not from 0, and 0 is reserved to mean "the first recorded medium still on the server". If you select a media number that no longer exists, there will be no effect, and you will get a warning that tells you the next and previous available media, relative to the number you gave.

Reply

```
Command:      $ACK$
Parameters:   <OK>
        then either:
              <ID>id                     id is a media-id
              <NUM>nnn                   num is the media number
              <TOTAL>nnn                 total is the total number of media available
        or:
```

(For a $SELECT$<SKIP> to beyond or before the range of loaded media)
<WARNING>
<MESSAGE>81Media unavailable
<ID>*id*
<NUM>*nnn*
<TOTAL>*nnn*

(The <TOTAL> parameter in the reply gives the total number of media available; the <NUM> parameter gives the currently selected media number.)

or:

(For a $SELECT$<NUM> to a non-existent media number, or an empty slot)
<WARNING>
<MESSAGE>81Media unavailable
<PREV>*ppp*
<NEXT>*nnn*

(where *ppp* can be 0, to mean no previous media, and *nnn* can be 0, to mean no more media after the number given. If either is non-zero, it indicates the next available media number in that direction relative to the one requested.)

or:

<ERROR>
<MESSAGE>XXmessage

Select track within media or playlist by track number

Request

Command:	$SELECT$	
Parameters:	<TRACK>	Mandatory
then either:		
	<SKIP>nnn	To inc/decrement by nnn tracks; default 1
or:		
	<NUM>nnn	To select track number nnn in the media or list

This causes the server to increment, decrement or select a specific track number. This may be within a media item or a playlist, depending upon what it is playing. <SKIP> and <NUM> are mutually exclusive: the command may use either exactly one, or none, of them.

What is meant by “track number” in this respect is the number in the playing order. That is to say: if you have RANDOM mode selected, the “track number” here refers to the current randomised playing order, not to the original order of tracks in the media or playlist.

If a value is supplied to <SKIP>, it must be an integer (it may be negative); if it is not, the default is 1. If a value of 0 is given to the parameter, the server will not change its state, but simply report the current track.

If the server is not in repeat mode (see **Set play flags** on page 43), an attempt to skip beyond the last track or to before the first will result in a warning and no change in state.

If it is in repeat mode, skipping “wraps around” in either direction. That is to say, a skip to the track before the first will play the final track in the list; a skip to the track after the last in the list will play the first.

The value for <NUM> must be a positive integer. Note that tracks are numbered on media and playlists from 1, not from 0.

Reply

Command:	ACK	
Parameters:	<OK>	
then either:		
	<ID>*nnn*	*nnn* is a track-id
	<NUM>*nnn*	*nnn* is a positive integer
	<ORIG>*nnn*	*nnn* is a positive integer
	<TOTAL>*nnn*	*nnn* is a positive integer
	<LEN>*hhhh:mm:ss*	
or:		
	<WARNING>	
	<MESSAGE>*XXmessage*	
	<NUM>*nnn*	*nnn* is a positive integer
	<ORIG>*nnn*	*nnn* is a positive integer
	<TOTAL>*nnn*	*nnn* is a positive integer
or:		
	<ERROR>	
	<MESSAGE>*XXmessage*	

On success, or upon a warning, the server reports the (new) track number playing (<NUM>), its original position in the media or playlist (<ORIG>), its track-id (<ID>), the total number of tracks in the current media or playlist (<TOTAL>), and the track's running time (<LEN>).
It is worth pointing out the difference between <NUM> and <ORIG>. <NUM> gives the track number in the current playing order (which may be randomised); <ORIG> gives the original track number in the track's media or playlist (depending upon what has been selected previously using, for example, $SELECT$<ID>).

<COMPR> gives the track's compression type.

If the server does not have a media file corresponding to that track, it will find the next available track for which it does have a file.

Status and reporting

Commands in this section are all concerned with requesting information on some aspect of the server's current state.

Operating mode

Request

Command:	$STATUS$	
Parameters:	<MODE>	

Reply

Command:	ACK	
Parameters:	<OK>	
	<MODE>*mode*	*mode* is PLAY, PAUSE or STOP
	[Introduced in Protocol version 1.02]	
	<DONE>	Only if playout has reached the end of the selected item

Play status

Request

Command:	$STATUS$
Parameters:	<PLAY>

This requests information about the current enclosing item (be it playlist, media or singleton track) selected on the server.

Reply

Command:	ACK	
Parameters:	<OK>	
	<PLAY>	
	<TYPE>*type*	for *type,* see below

For *type*=UNSET:

No further parameters.

For *type*=TRACK:

<ID>*id*	*id* is an item id
<LEN>*hhhh:mm:ss*	the length of the track
<NAME>*name*	*name* is a string
<ARTIST>*artist*	*artist* is a string

For *type*=MEDIA:

<ID>*id*	*id* is an item id
<TOTAL>*nnn*	*nnn* is a positive integer
<LEN>*hhhh:mm:ss*	the total length of the media
<NAME>*name*	*name* is a string
<ARTIST>*artist*	*artist* is a string

For *type*=SPLIST:

<ID>*id*	*id* is an item id
<TOTAL>*nnn*	*nnn* is a positive integer
<LEN>*hhhh:mm:ss*	the total length of the playlist
<NAME>*name*	*name* is a string

For *type*=DPLIST:

Not yet supported; watch this space!

Play flags

Request

Command:	$STATUS$	
Parameters:	<PLAY>	
	<FLAG>	

Reply

Command:	ACK	
Parameters:	<OK>	
	<PLAY>	
	<FLAG>	
	<RANDOM>*onoff*	*onoff* is either ON or OFF
	<REPEAT>*onoff*	*onoff* is either ON or OFF

Track status

Request

Command:	$STATUS$
Parameter:	<TRACK>

This requests information about the current track selected on the playout destination to which the command has been sent.

Reply

Command:	ACK	
Parameters:	<OK>	
	<ID>*id*	*id* is an item id
	<NUM>*nnn*	*nnn* is a positive integer
	<ORIG>*nnn*	*nnn* is a positive integer
	<LEN>*hhhh:mm:ss*	the length of the track
	<NAME>*name*	*name* is a string
	<ARTIST>*artist*	*artist* is a string

The <NUM> parameter gives the play order number within the enclosing item (playlist or disc). If the track is being played as a singleton, this is always 1.

As of *version 1.02 of the Protocol* there are alternatives to this request for controllers with input buffers of less than 1024 bytes (see page 157).

Track position

Request

Command:	$STATUS$
Parameters:	<POS>

This requests the current play position of the currently-selected track.

Reply

Command:	ACK
Parameters:	<OK>
	<POS>*hhhh:mm:ss*
	<MSECS>*MMM*

The <MSECS> parameter gives the fractional part in milliseconds.

Track encoding types

There are a number of ways of encoding audio and video data (to name but two types). The tracks stored on the server are sometimes stored uncompressed; sometimes compressed. Each type of encoding has a unique ID. This command allows a controller to discover the encoding IDs for a given type of track.

Request

Command:	$STATUS$	
Parameters:	<COMPR>*tracktype*	*tracktype* is AUDIO or VIDEO.

Reply

Command:	ACK	
Parameters:	<OK>	
	<COMPR>*tracktype*	

followed by zero or more of the following:

<ID>*compr-id*	*compr-id* is a non-negative integer

Details of a track encoding type

This command allows the controller to find out exactly what a given compression ID represents.

Request

Command:	$STATUS$	
Parameters:	<COMPR>	
	<ID>*compr-id*	As obtained from the command above.

Reply

Command:	ACK	
Parameters:	<OK>	
	<COMPR>	
	<ID>*compr-id*	
	<TYPE>*type*	*type* is a name for the compression type
	<BPS>*bps*	*bps* is compressed bytes per second
	<BITS>*bits*	*bits* is bits per channel per sample
	<CHANS>*chans*	*chans* is number of channels
	<FREQ>*freq*	*freq* is samples per second (=Hz)

The *type* argument is just an arbitrary string that describes the encoding type in a reasonably concise and user-friendly way. It has no other significance.

Not all compression formats generate a reliable number of compressed bytes per second, so the *bps* argument can only be regarded as a rough average for “normal” data. In many compression formats, data which are “noisy” compress far less well than those that are mathematically more regular, and therefore more predictable and compressible.

The numeric arguments are integers: that is to say, any fractional part is dropped.

Free space on the server

Request

Command:	$STATUS$
Parameters:	<FREE>

Reply

Command:	ACK	
Parameters:	<OK>	
	<FREE>*freeK*	*freeK* is total free space in Kb.
	<MAX>*maxK*	*maxK* is largest free area in Kb.

These may seem slightly contradictory, but bear in mind that the server may have several disc units (and/or disc partitions) for media storage. If, for example, it had four discs, there could be 8000Kb free in total, but only 2000Kb on any one partition – and thus 2000K would be the greatest volume of data that could be recorded in any one file.

Free space on the server in terms of playback time

[Request (and reply) introduced in Protocol version 1.02]

The $STATUS$<FREE> request reports the free space in kilobytes. It is possible to use that information together with the details of the various types of encoding (c.f. Track encoding types, p 60) to determine the amount of space left on the server in terms of playback time, assuming a particular encoding is used for further recording. For convenience and standardisation across controllers, it is now recommended that the following request be used instead. Note that the result takes into account reservation of some space for server operations.

Request

Command:	$STATUS$	
Parameters:	<FREE>	
	<COMPR>	
	<ID>*compr-id*	*compr-id* must be one of the values resulting from a $STATUS$<COMPR>*tracktype* request.

Reply

Command:	ACK	
Parameters:	<OK>	
	<FREE>	
	<COMPR>	
	<ID>*compr-id*	
	<LEN>*space-as-playing-time*	The value is in the format *hhhh:mm:ss.*
	<MESSAGE>*message*	Optional: only if extra information available

Currently the only *message* is `FULL' , which indicates that subsequent attempts to record are unlikely to succeed until space is freed up on the server.

Status of online lookups

Request

Command:	$STATUS$
Parameters:	<LOOKUP>

Reply

Command:	ACK	
Parameters:	<OK>	
	<LOOKUP>*status*	See below

The *status* argument is one of the following:

ACTIVE – there is a lookup in progress;
DONE – there is no lookup in progress or needed at present;
QUEUE – there is no lookup in progress, but one or more items await lookup.

Status of underruns

This command should be sent to a playout destination, not to `server`.

Request

Command:	$STATUS$
Parameters:	<UNDERRUN>

Reply

Command:	ACK	
Parameters:	<OK>	
	<UNDERRUN>	
	<TOTAL>*num*	Total number of underruns since boot
	<TRACK>*num*	Number of underruns during this track
	<DROP>	
	<TOTAL>*num*	Total number of drops since boot
	<TRACK>*num*	Number of drops during this track.
	<TIME>	
	<TOTAL>*num*	Total playout time (secs) since boot
	<TRACK>*num*	Seconds played of this track.

Note that any *num* argument may be –1, which means that the information is not available to the server (for example, the low-level driver does not support the reporting of that type of event).

An underrun is an incident where the low-level driver (the software that operates the audio or video output hardware) was expecting data, but they were not available. Underruns are usually noticeable by a user.

A drop is a contiguous sequence of one or more underruns. A drop is terminated by successful playout, or the stopping of playout, on the destination. For example, if there were three underrun events in a row, followed by successful playout, followed by a sequence of another four underruns, this would constitute only two drops.

Update notifications

The server can be made to notify controllers of its current state, or of changes in its state, without having to be prompted every time. These messages are known as **updates**. An update message can be triggered for one of two reasons::

Event-triggered – something on the server changes state;
Time-triggered – these messages are sent at regular intervals regardless of any change of state.

These two types of update are independent of each other. If you request both for a given update type, you will get both: an immediate update upon a change of state, plus a time-triggered update at the programmed interval.

There are many different types of update. By default, the server does not send any type of update until it is requested to do so. It is up to a controller to switch on any given update type, and to switch it off when it is no longer needed. Not all update types support time-triggered updates.

Updates work slightly differently from normal commands. You turn on a type of update in the same way you issue any other type of command. However, once an update type is switched on, the controller will start to send $UPDATE$ messages as and when it is appropriate to do so.

$UPDATE$ messages are not linked to any command's *message-sequence-char*.

The commands described here have three sections:
Request – the command to issue to turn an update on or off;
Reply – the immediate reply to the request;
Update – the update message, which happens at some other time.

Development note: at present, and for the immediate future, if you request timed updates – regardless of the frequency you request – you will get them every 5-6 seconds. In other words, a non-zero value for the update frequency will give 5-6 second updates, and a zero value will stop timed updates.

VERY strong recommendation: turn on only those updates that you actually need at any given time. Every update that is switched on (most particularly timed updates) causes the server to perform actions that may slow its operation. An overburden of housekeeping work, such as serving large numbers of unnecessary updates, could lead to the point where the server cannot deliver streaming media to destinations in time, leading to playout glitches or worse.

So, for example, turn on track and mode updates for only the destination your controller is presently displaying to the user. If the user elects to change playout destination, turn off the updates for the old destination, turn them on for the new one, and use $STATUS$ commands to get the up-to-the-moment information on what's playing in the new zone.

Play-state update

This request should be sent only to `server`.

Request

Command:	$STATUS$	
Parameters:	<UPDATE>	
then:	<EVERY>*nnnn*	*nnnn* is zero or a positive integer
and/or:	<TRACK>*onoff*	*onoff* is ON or OFF
and/or:	<MODE>*onoff*	*onoff* is ON or OFF

The <EVERY> parameter requests the respondent to update the sender on the track ID and position on a timed basis. If the value is zero, timed update is to be stopped immediately; otherwise it gives a required reporting frequency, in 0.1s quanta. If the command is accepted, the respondent will update the sender continuously during play, or once every ten seconds whilst paused or stopped. If the server cannot send as often as requested, it will do so as frequently as it can manage.

Strong recommendation: do not request updates more than once per second. (But see comments at the start of this section.)

The <TRACK> parameter requests that updates at the end of every track (or the beginning of the first) be turned either ON or OFF (default: OFF). TRACK updates are also sent if a skip within a track is performed.

The <MODE> parameter requests that updates on change of mode (between PLAY, PAUSE and STOP) be turned either ON or OFF (default: OFF).

Note that the <TRACK> and <MODE> parameters may be supplied singly or together (in that order) in the absence of an <EVERY> parameter. If this is done, the request will not affect the sending of timed play-state updates, just the sending of TRACK and/or MODE updates as requested. The format of the update is the same in all cases; the only difference is in the triggering of updates.

Reply

This is the immediate response, on success:

Command	ACK
Parameters:	<OK>

Update

If the request succeeded, you will receive, at the intervals described above, one of the following (which does not carry a reply sequence character):

Command:	$UPDATE$	
Parameters:	<MODE>*mode*	*mode* is PLAY, PAUSE or STOP
then:	<ID>*id*	*id* is a track ID
	<POS>*hhhh:mm:ss*	*pos* is the track position
	<MSECS>*msecs*	*msecs* is milliseconds into track position above

	<NUM>*num*	*num* is the track number in current play order
	<ORIG>*num*	*num* is track number in the original play order
	[Introduced in Protocol version 1.02]	
	<DONE>	only if playout has stopped at the end of the selected item
or:	<UNSET>	
or:	<ERROR>XXmessage	(note that this follows $UPDATE$, not ACK)

This is an unsolicited packet, and does not require its respondent to acknowledge it with an ACK. If it has been triggered at the end of a track, or on a transition to STOP mode, it indicates details of the track newly selected, not the outgoing track.

The <UNSET> parameter indicates that the server has an empty state, with no playlist or disc selected.

The <ERROR> response indicates a continuing problem, which will need user intervention.

Record state update

This request should be sent only to `server`.

Updates are sent on a timed basis and/or on a change of recording state.

You can set this update for a given type of recording source, but not for individual sources. As at 1.00, only the <CD> source is defined.

Request

Command:	$STATUS$	
Parameters:	<UPDATE>	
	<*SOURCE*>	At 1.00, the only defined *SOURCE* is CD
	<RECORD>*[onoff]*	*onoff* is optional, and is either ON or OFF
	<EVERY>*nnn*	Optional; *nnn* is the interval in 0.1s units.

If you supply ON or OFF as argument to the mandatory <RECORD> parameter, you turn on or off state-triggered updates.

If you use the optional <EVERY> argument, you turn on (at the requested non-zero interval, but see the **Development note** at the start of the section) or off (if you supply 0) timed record-state updates. Note that you will get a timed update for every source of the given type.

Strong recommendation: don't use the <EVERY> version unless you know for certain that there are very few sources of the given type. If your controller happens to be attached to a server with, say, 20 CD units, you will get 20 updates every interval, which could put a severe load on the server.

Reply

If successful:

Command:	ACK
Parameters:	<OK>

Update

Command:	$UPDATE$	
Parameters:	<*SOURCE*>*sourceno*	
	<RECORD>	
followed by:		
	<NONE>	No recording in progress on this source
or:		
	<LEFT>*secs*	*secs* estimated to end of recording sequence (in ripping seconds). Can be –1, if the server cannot yet make an estimate.
	<TRACK>	
	<INDEX>*ix*	*ix* is the 1-based track index within the media
	<ID>*trackid*	*trackid* is the track's ID.
	<TIME>*secs*	*secs* is the total track playout length
	<LEFT>*secs*	*secs* is the amount of the track still unrecorded (in playout seconds, not ripping seconds)
	<NUM>*num*	*num* is the its number in the recording sequence
	<TOTAL>*total*	*total* is the total number of tracks to record
	<MEDIA>	
	<ID>*mediaid*	*mediaid* is the media's ID (can be zero).
	<TIME>*secs*	*secs* is the media's total playout length
	<LEFT>*secs*	*secs* is total time on the media still unrecorded. (in playout seconds, not ripping seconds)
	[Introduced in Protocol version 1.02]	
	<MEDIA>	
	<NUM>*medianum*	*medianum* is the media's media number (c.f. `Select media by media number', p 50).

The <LEFT> and <TIME> parameters need a little explanation, as they can be confusing.

Unless the server is extremely heavily loaded, or there are disc-related features that are slowing the ripping process, ripping happens much more quickly than the real-time length of the CD. In other words, a CD whose total playout time might be 78 minutes could be fully ripped in only 10-20 minutes.

The initial <LEFT> parameter gives the server's estimated time to completion of the ripping process. This is calculated by averaging its current ripping rate, and applying that to the amount of data left to record. It may be given as –1, in which case the server has not had enough ripping time yet to make an accurate estimate. The point to emphasise is that this parameter predicts the amount of time the user will have to wait before the ripping is complete.

The <LEFT> parameter soon after <TRACK> gives the number of seconds still unrecorded for the track currently ripping – but this is in terms of the track's playout time, <u>not</u> ripping time. The total playout time for the track is given by the <TIME> parameter.

The <LEFT> and <TIME> parameters soon after <MEDIA> have the same meanings as their corresponding parameters after <TRACK>, but refer to the media as a whole.

Disc tray update

This request should be sent only to `server`.

Updates are sent whenever a ripping source disc unit (at present, CDs only) ejects or loads a disc.

Request

Command:	$STATUS$	
Parameters:	<UPDATE>	
	<*DISCTYPE*>	
	<TRAY>*onoff*	*onoff* is either ON or OFF

Note that this is requested for a whole *DISCTYPE* class (at present, only CD), not for individual drives in that class. The controller will be notified whenever <u>any</u> drive in that class opens or closes its tray. (Or ejects or loads a disc, if it's a slot-loader.)

Tray updates are always event-triggered. There is no time-triggered option.

Reply

Command:	ACK
Parameters:	<OK>

Update

Command:	$UPDATE$	
Parameters:	<*DISCTYPE*>*discnum*	
	<TRAY>*openclose*	*openclose* is either OPEN or CLOSE

NOTE: at present, these updates are <u>only</u> issued when the tray is opened or closed under <u>software</u> control. If the user has an eject button and uses it, or if the user closes the tray manually, an update may not happen. You should therefore note this information if you receive it, but not rely upon doing so.

Media availability update

This request should be sent only to `server`.

Updates are sent when new media become available (or unavailable) for recording on any of the ripping source drives (at present, only CDs).

Request

Command:	$STATUS$	
Parameters:	<UPDATE>	
	<*DISCTYPE*>	
	<MEDIA>*onoff*	*onoff* is either ON or OFF

Note that this is requested for a whole *DISCTYPE* class (at present, only CD), not for individual drives in that class. The controller will be notified whenever media become available (or unavailable) for <u>any</u> drive in that class.

Media availability updates are always event-triggered. There is no time-triggered option.

Reply

Command:	ACK
Parameters:	<OK>

Update

Command:	$UPDATE$
Parameters:	<*DISCTYPE*>*discnum*

then if media became available:

<MEDIA>

[Introduced in Protocol version 1.02]

otherwise (media became unavailable):

<NONE>

On/offline update

This request should be sent only to `server`.

Updates are sent when the online status of the server changes. *As of Protocol version 1.02*, updates will also be sent while the server is going online as it passes through each stage of each connection attempt.

Request

Command:	$STATUS$	
Parameters:	<UPDATE>	
	<ONLINE>*onoff*	*onoff* is either ON or OFF

Reply

Command:	ACK
Parameters:	<OK>

Update

Command:	$UPDATE$	
Parameters:	<ONLINE>*status*	See below

[Introduced in Protocol version 1.02]

then if the server is online:

<USER><COUNT>*num-users*	See below

then if the server is attempting to go online or has just succeeded:

<TRY>	
<NUM>*try-num*	*try-num* is in the range 1 to *total-tries*
<TOTAL>*total-tries*	
<STATUS>*try-state*	See below

then if the above <TRY> information is present and addition information is available:

<INFO>*IXXdetail*	See below

The *status* argument is one of the following:

- YES (the server is online);
- NO (the server is not online, and is not attempting to connect);
- UNKNOWN (the online status is unknown at present);
- CONNECTING (the server is in the process of trying to connect);
- DISCONNECTING (the server is in the process of trying to disconnect).

num-users is the number of entities which have requested that the server go online (using the current network interface) and not yet countermanded that request.

The *try-state* is the state of the current connection attempt, as reported by the network interface. The states:

- `connecting’ (the initial state during a connection attempt) and
- `connected’,
- `failed’ and
- `aborted’ (one of which will be the final state of a connection attempt)

are common to all types of network interface. During a dialup connection attempt, the intermediate states: `initialising’, `dialling’ and `authenticating’ will be reported in turn, if and when these stages are reached. No intermediate states are currently reported by the ethernet interface but this may change. It should be assumed that, while existing states will not be withdrawn, additional states may be added for existing network interface types or introduced by new ones.

The standard use of the <INFO> parameter is to allow a network interface to convey the reason for failure of a connection attempt. The components of *IXXdetail* are :-

- *I* : a single hex digit indicating the network interface involved (currently `0’ for the first ethernet interface or `1’ for the first dialup modem interface);
- *XX* : an error detail code (in the form of two hex digits) which is specific to the type of network interface and
- *detail*: a textual description of the type of failure.

The error detail codes are defined in the section `Networking error detail codes’ (p 126) but the detail code `00’ is reserved across all types of network interface to mean that no error has occurred. Network interfaces may thus make use of the <INFO> parameter to report additional detail about stages of a connection, provided that the detail code is `00’. Currently the dialup modem interface uses this approach to report the phone number being dialled when its *try-state* is `dialling’.

TRACKDB change update

This request should be sent only to `server`.

Updates are sent when TRACKDB is changed for some reason.

Request

Command:	$STATUS$	
Parameters:	<UPDATE>	
	<TRACKDB>*onoff*	*onoff* is either ON or OFF

Reply

Command:	ACK
Parameters:	<OK>

Update

Command:	$UPDATE$	
Parameters:	<TRACKDB>*reason*	See below

Optionally followed by:
<MEDIA>
<ID>*mediaid*

Optionally followed by:
<TRACK>
<ID>*trackid*

The *reason* argument is one of the following:

- NEW_CD – this is deprecated, but should be supported by 1.00 clients. It is exactly synonymous with MEDIA (see below);
- MEDIA – new media have been added to the TRACKDB. This may be a fully-described album (for example), or simply a previously-encountered disc inserted into the server. This will replace NEW_CD;
- RECORD – the recording of media has caused a change to the TRACKDB;
- LOOKUP – an online lookup has caused TRACKDB changes;
- DELETE – a track or media deletion has changed the TRACKDB;
- ALTER – existing information on a track or media has been changed. This could be anything: artist name, track name, media name and so on;
- RESET – someone has changed the database to the point where controllers should make no further assumptions about the database's contents: all cached information should be abandoned and reloaded.

If the RCP is aware of the identity of a single track and/or media whose change has triggered this update, it will append <MEDIA><ID> and/or <TRACK><ID> to the packet. Sometimes, things can change in the database of which the RCP has no exact knowledge: in this case it can tell only that a change has occurred.

Controllers receiving these updates should therefore not rely upon the presence of <MEDIA><ID> or <TRACK><ID>, although, if present, they will be accurate.

PLAYLISTDB change update

This request should be sent only to `server`.

Updates are sent when something changes the PLAYLISTDB: in other words, when a playlist is created, deleted or modified.

Request

Command:	$STATUS$	
Parameters:	<UPDATE>	
	<PLAYLISTDB>*onoff*	*onoff* is either ON or OFF

Reply

Command:	ACK
Parameters:	<OK>

Update

Command:	$UPDATE$
Parameters:	<PLAYLISTDB>

Play flags update

This request should only be sent to playout destinations, and never to `server`.

Updates are sent when the play flags for the playout destination in question are changed.

Request

Command:	$STATUS$	
Parameters:	<UPDATE>	
	<PLAY>	
	<FLAG>*onoff*	*onoff* is either ON or OFF

Reply

Command:	ACK
Parameters:	<OK>

Update

Command:	$UPDATE$	
Parameters:	<PLAY>	
	<FLAG>	
	<RANDOM>*onoff*	*onoff* is either ON or OFF
	<REPEAT>*onoff*	*onoff* is either ON or OFF

Online lookups update

This request should be sent only to `server`.

Updates are sent when a lookup succeeds, is aborted, or fails completely after a number of retries (e.g. the Internet connection or the online media information database are unreachable).

Request

Command:	$STATUS$	
Parameters:	<UPDATE>	
	<LOOKUP>*onoff*	*onoff* is either ON or OFF

Reply

Command:	ACK
Parameters:	<OK>

Update

Command:	$UPDATE$	
Parameters:	<LOOKUP>*status*	See below

The *status* argument is one of the following:

- ACTIVE (lookup is in progress);
- DONE (lookup complete);
- QUEUE (items are awaiting lookup);
- ERROR (a problem has occurred; detailed status information cannot be obtained).

Underrun update

This request should only be sent to playout destinations, and never to `server`.

Updates are sent when audio playout for the playout destination in question causes underruns.

Request

Command:	$STATUS$	
Parameters:	<UPDATE>	
	<UNDERRUN>*onoff*	*onoff* is either ON or OFF

Reply

Command:	ACK
Parameters:	<OK>

Update

Command:	$UPDATE$	
Parameters:	<UNDERRUN>	
	<TOTAL>*nnn*	*nnn* is total number of underruns since last boot
	<TRACK>*nnn*	*nnn* is number of underruns during current track
	<DROP>	
	<TOTAL>*nnn*	*nnn* is total number of drops since last boot
	<TRACK>*nnn*	*nnn* is number of drops during current track
	<TIME>	
	<TOTAL>*nnn*	*nnn* is total no. of seconds playout since boot
	<TRACK>*nnn*	*nnn* is no. of seconds played of current track

Note that any of the *nnn* figures above can be –1, if that information is not available (for example, if the playout destination does not support underrun or drop reporting).

An **underrun** occurs when the server cannot supply data quickly enough to supply the destination. This usually occurs when the server is under heavy load. Each underrun is one such event reported by the destination hardware drivers. It may mean only one sample lost; it may mean a whole buffer's-worth.

A **drop** is a contiguous sequence of one or more underruns. That is to say, if the low-level drivers report drops three times in a row, this would constitute one drop. If playout was re-established, and the drivers started reporting underruns again, this would constitute another drop.

Cache close update

This request should be sent only to `server`.

Updates are sent when a cache is invalidated. The cache is implicitly closed, and no further action is needed by a controller (except perhaps reopening and resynchronising with the cache in question).

Request

Command:	$STATUS$	
Parameters:	<UPDATE>	
	<CACHE>	
	<CLOSE>*onoff*	*onoff* is either ON or OFF

Reply

Command:	ACK
Parameters:	<OK>

Update

Command:	$UPDATE$	
Parameters:	<CACHE>*cache*	See below
	<CLOSE>	

The *cache* argument is one of the following:

- MEDIA – this means that the MEDIA, GENREMEDIA and ARTISTMEDIA caches are now invalid;
- ARTIST – this means that the MEDIA and ARTISTMEDIA caches are now invalid;
- GENRE – this means that the GENRE and GENREMEDIA caches are now invalid;
- PLAYLIST – this means that the PLAYLIST cache is now invalid.

Configuration update

[Request (and reply and update) introduced in Protocol version 1.02]

This request may be sent to any destination to which configuration requests may be sent.

Updates are sent when a configuration setting is changed and will have as source address the destination to which the request which changed the setting was sent.

Request

Command:	$STATUS$	
Parameters:	<UPDATE>	
	<CONFIG>*onoff*	*onoff* is either ON or OFF

Reply

Command:	ACK
Parameters:	<OK>

Update

Command:	$UPDATE$	
Parameters:	<CONFIG>*category*	See below
	<ITEM>*item*	See below
	<HAS>*value*	See below

The source address and the *category* and *item* arguments together uniquely identity a configuration setting within the overall system. This hierarchical addressing system for configuration settings is described in more detail in Configuring the server (p 149). The *value* argument is the new value to which the configuration setting has been changed.

Power mode update

[Request (and reply and update) introduced in Protocol version 1.02]

This request should be sent to `server.`

Updates are sent when the power mode changes (see **Querying and setting the power mode** on page 157).

Request

Command:	$STATUS$	
Parameters:	<UPDATE>	
	<POWER>	
	<MODE>*onoff*	*onoff* is either ON or OFF

Reply

Command:	ACK
Parameters:	<OK>

Update

Command:	$UPDATE$	
Parameters:	<POWER>	
	<MODE>*power-mode*	*power-mode* may be RUN, STANDBY, RESTART or SHUTDOWN

Simple search facilities

The user may want, through the controller, to browse the databases on the server, and maybe even to create playlists. These interfaces are intended to make this easy.

The type of an ID

This allows a controller to find out what an item ID represents. Note that this applies only to media, track and playlist IDs.

Request

Command:	$SEARCH$	
Parameters:	<INFO>	
<ID>*item-id*		

Reply

Command:	ACK	
Parameters:	<OK>	
	<INFO>	
	<ID>*item-id*	
	<TYPE>*type*	See below.
	<NAME>*name*	This could be empty

The *type* will be one of the following:

- TRACK – a track;
- MEDIA – one of the media;
- SPLIST – a static playlist;
- DPLIST – a dynamic playlist.

This command can also be used to determine whether an *item-id* is valid. If it is not. you will receive “ACK<ERROR><MESSAGE>13No such ID” instead of the ACK<OK> message above.

Track details

Request

Command:	$SEARCH$	
Parameters:	<TRACK>	
	<ID>*track-id*	*track-id* is optional, <ID> is not optional
	<FULL>	

This command can be sent to `server` or to a playout destination.

The <ID> parameter is mandatory.

If the command is sent to `server`, *track-id* <u>must</u> be supplied.

If it is sent to a playout destination, *track-id* is optional: if absent, it is implicitly the current media selection <u>on that playout destination</u>. If the playout destination does not have a track currently selected, an error will be generated.

The <FULL> parameter causes more detailed information to be supplied.

Reply

Command:	ACK	
Parameters:	<OK>	
	<ID>*track-id*	
	<TYPE>*track-type*	
	<LEN>*track-len*	
	<COMPR>*compr*	Optional: only if information available
	<NAME>*track-name*	
	<ARTIST>*track-artist*	
	<MEDIA><ID>*media-id*	

If the <FULL> parameter was given, then the following also follow, and refer to the track's media:

<LEN>*media-len*
<NUM>*track-num*
<TOTAL>*media-tracks-total*
<NAME>*media-name*
<ARTIST>*media-artist*
<GENRE>*media-genre*

track-id is an opaque designation that the server gives the current track.
track-type is AUDIO or VIDEO.
track-len is the track's length, in the format *hhhh:mm:ss*. This may be 0000:00:00 if the track has not yet been recorded onto the server.
compr is the track's encoding type: a non-negative integer.
track-name is an arbitrary string giving the track's name
track-artist is an arbitrary string naming the artist of the track.
media-id is an opaque designation that the server gives the track's media. This can be 0, meaning that the track has no media associations.

media-len is the total playing time of the media, in the format *hhhh:mm:ss*.
track-num gives the track's index within its source media, numbered from 1.
media-tracks-total gives the total number of tracks in the media.
media-name, media-artist and *media-genre* are arbitrary strings giving the name, artist and genre of the media.

If the track is a member of more than one medium (at present this does not happen, but this should not be relied upon), there will be more than one section starting <MEDIA>.

As of *version 1.02 of the Protocol* there are alternatives to this request for controllers with input buffers of less than 1024 bytes (see page 157).

Basic media details (disc etc.)

Request

Command:	$SEARCH$	
Parameters:	<MEDIA>	
	<ID>*media-id*	*media-id* is optional, <ID> is not

This command can be sent to `server` or to a playout destination.

If it is sent to `server`, *media-id* must be supplied. If it is sent to a playout destination, *media-id* is optional: if absent, it is implicitly the current media selection on that playout destination. If the playout destination does not have media currently selected (the server has just booted, or a track or playlist has been selected), an error will be generated.

Reply

Command: ACK
Parameters: <OK>
<MEDIA>
<ID>*media-id*
<TYPE>*media-type*
<TOTAL>*media-total*
<SOURCE>*media-source*
<LEN>*media-len*
<NAME>*media-name*
<ARTIST>*media-artist*
<GENRE>*media-genre*

media-id is an opaque designation that the server gives the current media.
media-type is AUDIO, VIDEO or MIXED.
media-total is the total number of tracks in the media.
media-source depends upon media-type. If *media-type* is AUDIO, *media-source* is CD, DVD, LP, OTHER, UNKNOWN. If *media-type* is VIDEO, *media-source* is DVD, VTR, UNKNOWN.
media-len gives the total running time of the media in the format *hhhh::mm:ss*.
media-name is an arbitrary string naming the media (the disc, video etc.).
media-artist is an arbitrary string naming the artist of the media (which could be “Original Artists” for a compilation album, for example).

You should not rely upon the value for *media-source*: it is provided for information value only.

As of *version 1.02 of the Protocol* there are alternatives to this request for controllers with input buffers of less than 1024 bytes (see page 157).

Media track details

Request

Command:	$SEARCH$	
Parameters:	<MEDIA>	
	<ID>*id*	*id* must be provided
	<TRACK>	
	<FROM>*from*	
	<FOR>*for*	
	<NONE>	Optional

Reply

Command:	ACK
Parameters:	<OK>
	<SEARCH>
	<MEDIA>
	<ID>*media-id*
	<TRACK>
	<FROM>*from*
	<FOR>*for2*

Followed by' for2' entries of the following form:

<AT>*at-val*	In the range *from* to (*from* + *for2* – 1)
<ID>*track-id*	
<NAME>*track-name*	
<NONE>	See below.

Followed, if this completes information for the media in question, by:

<EOF>

If the track name is not known, *track-name* has the content "**Unknown**" (without the quotes, of course).

The <NONE> parameter in the per-track information is only given if: (1) the optional <NONE> parameter was given to the original command, and; (2) if there are no track data files associated with this entry (this can happen for tracks which have not yet been ripped).

Note that the value of *for2* (in the reply) may not be the same as *for* (in the command), if not all track details could be fitted into one packet.

As of *version 1.02 of the Protocol* there is an alternative to this request for controllers with input buffers of less than 1024 bytes (see page 157).

Playlist details

Request

Command:	$SEARCH$
Parameters:	<PLAYLIST>
	<ID>*playlist-id*

This command can be sent to `server` or to a playout destination.

If it is sent to `server`, *playlist-id* must be supplied. If it is sent to a playout destination, *playlist-id* is optional: if absent, it is implicitly the playlist media selection on that playout destination. If the playout destination does not have a playlist currently selected (the server has just booted, or media or a track has been selected), an error will be generated.

Reply

Command:	ACK
Parameters:	<OK>
	<PLAYLIST>
	<ID>*playlist-id*
	<*playlist-type*>
	<TOTAL>*playlist-total*
	<LEN>*playlist-len*
	<NAME>*playlist-name*

playlist-id is an opaque designation that the server gives the current playlist.
playlist-type is either SPLIST (static playlist) or DPLIST (dynamic playlist)
playlist-total gives the number of tracks in the playlist[1].
playlist-name is an arbitrary string naming the playlist.
playlist-len gives the total running time of the playlist[1] in the format *hhhh:mm:ss*.

As of *version 1.02 of the Protocol* there are alternatives to this request for controllers with input buffers of less than 1024 bytes (see page 157).

[1] These are guaranteed valid only for static playlists. The track count and length of dynamic playlists depend upon the tracks available on the server when the playlist is loaded. If new tracks are added between using $SEARCH$<PLAYLIST> and selecting the playlist to play, these figures will be inaccurate.

Playlist track details

[Request (and reply) officially introduced in Protocol version 1.02]

Request

Command:	$SEARCH$	
Parameters:	<PLAYLIST>	
	<ID>*id*	*id* must be provided
	<TRACK>	
	<FROM>*from*	
	<FOR>*for*	
	<NONE>	optional

Reply

Command:	ACK	
Parameters:	<OK>	
	<SEARCH>	
	<PLAYLIST>	
	<ID>*playlist-id*	
	<TRACK>	
	<FROM>*from*	
	<FOR>*for2*	

Followed by' for2' entries of the following form:

<AT>*at-val*	In the range *from* to (*from* + *for2* – 1)
<ID>*track-id*	
<NAME>*track-name*	
<NONE>	See below.

Followed, if this completes information for the playlist in question, by:

<EOF>

If the track name is not known, *track-name* has the content "**Unknown**" (without the quotes, of course).

The <NONE> parameter in the per-track information is only given if: (1) the optional <NONE> parameter was given to the original command, and; (2) if there are no track data files associated with this entry (this can happen for tracks which have not yet been ripped).

Note that the value of *for2* (in the reply) may not be the same as *for* (in the command), if not all track details could be fitted into one packet.

As of *version 1.02 of the Protocol* there is an alternative to this request for controllers with input buffers of less than 1024 bytes (see page 157).

The database cache

Controllers, especially the simpler ones, have a number of problems when it comes to displaying track and album names:

- How do they know to refresh their display?
- Is it really necessary to use advanced search commands, and iterate over the whole list generated (which could easily be, for example, over ten thousand tracks), in order to:
- Refresh their display?
- Update their locally-stored data?
- List (say) five tracks either side of the first one whose name starts with 'P'?

The database cache has been created to help solve these problems. It should also make simpler controllers easier to design, and minimise the amount of local data a controller must store in order to maintain browse lists.

How to use the caches

A controller first "opens" a cache on a given list. This operation returns a *marker* (essentially a throw-away handle), and alerts the RCP to take an interest in the list in question. The exact content of the marker is unimportant: it's essentially a "magic cookie". The number of characters in this marker is not fixed and may grow in the future up to a limit of 20; controllers should cope with any length up to that limit.

The controller can then use that marker to iterate over, or search into, the list in question, for as long as the marker remains current. If the database changes, rendering the cached information invalid, the marker becomes invalid, and is considered closed. The controller should reopen the list, and get a new marker.

Marker FIND operations make it reasonably easy to resynchronise with the new cache and pick up where the controller left off. The controller should hold on to the details under the display cursor (for example) when the cache was invalidated. It can then use the information to find the entry (or the nearest entry, if it has disappeared) again in the new cache.

All of the $SEARCH$<CACHE> commands should be sent to destination-id `server`.

The caches

A cache can be considered to be a table, whose rows are indexed, starting from row 1.

The columns are called *elements*. Each cache type has its own list of elements. These elements contain further information related to the NAME element, which is always present.

Each cache is sorted (alphabetically, and case-independently[3]) upon NAME. For each unique NAME (ignoring case) that they contain, the ARTISTMEDIA and GENREMEDIA caches are further sorted by MEDIA.

Caches can be used as fast lookup tables, as they can be searched both on NAME and ID (where present). They are designed to contain the additional information controller designers often wish to have displayed alongside the names of each type of browsable information.

The caches are:

Cachename	Intended purpose
MEDIA	Browsing media alphabetically. Alongside the name of the media, the browser can display the artist name and genre, and the controller can store the media ID in order to select it for playing. There is a single entry for each of the media on the server.
ARTIST	Listing all available artists.
PLAYLIST	Listing available playlists. The controller can store the playlist ID in order to select it for playing. There is a single entry for each playlist on the server.
GENRE	Listing all available genres.
ARTISTMEDIA	Browsing media alphabetically by artist name. There is an entry for each of the media on the server, so a given artist can appear more than once. The controller can store the media ID in order to select it for playing.
GENREMEDIA	Browsing media alphabetically by genre name. There is an entry for each of the media on the server, so a given genre can (and almost certainly will) appear more than once. The artist name for each media entry is also supplied. The controller can store the media ID in order to select it for playing.

Table x: Caches and their intended purposes

Element / Cachename	NAME	MEDIA	ID	ARTIST	GENRE
	String	string	integer	string	string
MEDIA	Media's name	-	Media ID	Yes	Yes
ARTIST	Artist's name	-	-	-	-
PLAYLIST	Playlist's name	-	Playlist ID	-	-
GENRE	Genre	-	-	-	-
ARTISTMEDIA	Artist's name	Yes	Media ID	-	-
GENREMEDIA	Genre	Yes	Media ID	Yes	-

Table xi: Cache elements

[3] For those developers familiar with C programming, the order is as defined by *strcasecmp()*.

Element	Contents, and intended purpose
NAME	The name of the principal subject of the cache (for example, the media name for MEDIA; the artist's name for ARTIST, and so on).
MEDIA	The name of the associated media.
ID	An associated ID. Precisely whose ID is dependent upon the cache type.
ARTIST	The name of the artist of the media.
GENRE	The genre of the media.

Table xii: Purposes of cache elements

Opening a cache

Request

Command:	$SEARCH$	
Parameters:	<CACHE>	
	<OPEN>*cachename*	

Reply

Command:	$ACK$	
Parameters:	<OK>	
	<SEARCH>	
	<CACHE>	
	<OPEN>*cachename*	
	<MARKER>*marker*	*marker* may be up to 20 characters long
	<COUNT>*rrr*	

Once the list has been opened, the RCP has the option of maintaining a cache upon it. The marker remains valid until closed (see later), or the cache becomes invalid. The <COUNT> parameter in the reply gives the number of items in the cache list.

Listing a cache

Request

Command:	$SEARCH$	
Parameters:	<CACHE>	
	<LIST>	
	<MARKER>*marker*	See below
	<FROM>*fff*	Optional
	<FOR>*rrr*	Optional

Reply

Command: ACK
Parameters:

Either:

<ERROR>
<MESSAGE>16Cache marker no longer valid

Or the following:

<OK>	
<SEARCH>	
<CACHE>	
<LIST>	
<MARKER>*marker*	
<FROM>*fff*	
<FOR>*nnn*	Note – *nnn* may be less than *rrr* above

Followed by nnn *entries of the following form:*

<AT>*xxx*	
<NAME>*name*	
<MEDIA>*medianame*	Optional
<ID>*id*	Optional
<ARTIST>*artist*	Optional
<GENRE>*genre*	Optional

After which there may be:

<EOF>	Optional

The *marker* value sent in the request must be a marker returned in the reply to a previous cache open request. In the reply, the <NAME> entry is always present: however, which (if any) of <ID>, <ARTIST>, <GENRE> and <MEDIA> are also present is dependent upon the list identified by the *marker*. Different lists expose different fields.

If the list is successful, the reply may not return as many entries as requested. This may be for one of two reasons: either they could not all be fitted into the packet length, or there are simply not that many left (this is not an error).

If, and only if, the reply gives all the remaining entries, up to the end of the list, it will be terminated with <EOF>.

If the error occurs, the marker should be considered invalid, and closed. The controller does not need to (and shouldn't) close it.

As of *version 1.02 of the Protocol* there is an alternative to this request for controllers with input buffers of less than 1024 bytes (see **Retrieving a single field from a single database cache item** on page 192).

Searching within a cache’s ID fields (for those caches that support an ID)

Request

```
Command:      $SEARCH$
Parameters:   <CACHE>
              <FIND>
              <MARKER>marker
              <ID>id
```

Reply

```
Command:      $ACK$
Parameters:
        Either:
              <ERROR>
              <MESSAGE>16Cache marker no longer valid

        Or the following:
              <OK>
              <SEARCH>
              <CACHE>
              <FIND>
              <MARKER>marker
              <ID>id

              Followed by either:
              <NONE>

              or:
              <FROM>fff
```

This searches the ID field for the list in question (if it has one; if it does not, this is an error).

If the command succeeds, it returns either <NONE> (meaning that no entries have that ID), or <FROM>*fff*, which gives the index of the item in the list matching that ID. The controller can use the $SEARCH$<CACHE><LIST> command to obtain the corresponding name.

Note that not all lists may support IDs. Initially, the ARTIST list will not, for instance.

The error has the same meaning as for the $SEARCH$<CACHE><LIST> command.

Searching within a cache’s NAME fields by matching a substring

Request

Command:	$SEARCH$	
Parameters:	<CACHE>	
	<FIND>	
	<MARKER>*marker*	
	<START>*str*	
	<PREV>	Optional; may not be used with <NEXT>
	<NEXT>	Optional; may not be used with <PREV>
	<FOR>	Optional

Reply

Command: ACK
Parameters:

Either:

<ERROR>
<MESSAGE>16Cache marker no longer valid

Or the following:

<OK>	
<SEARCH>	
<CACHE>	
<FIND>	
<MARKER>*marker*	
<START>*str*	
<PREV>	Only if given in the command
<NEXT>	Only if given in the command

Followed by either:
<NONE>

or:

<FROM>*fff*	
<FOR>*nnn*	Only if <FOR>given in the command

This searches the NAME field for the list in question.

The <START> parameter introduces a case-insensitive initial substring.

Normally, the search is exact: if an entry cannot be found to match the criteria, the response is <NONE>. The controller can use the optional <PREV> or <NEXT> parameters to change this behaviour, if there is no exact match. If one of these is set, the response will be either the nearest match in that direction, if there are any entries in that direction, or <NONE> only if there are no entries at all in the given direction.

Note that this search finds the first entry (the lowest-sorting) that matches (or the nearest entry if <PREV> or <NEXT> are given and there is no exact match).

The <FOR> entry in the reply only occurs if the corresponding <FOR> has been included in the command parameters. If present, it indicates the number of entries matching the criteria. So, for example, you could do the following (using a *marker* over the ARTISTMEDIA list):

> $SEARCH$<CACHE><FIND><MARKER>*marker*<START>Elton John<FOR>

…and get the reply:

> ACK<OK><SEARCH><CACHE><FIND><MARKER>*marker*<START>Elton John <FROM>78<FOR>6

…and this would indicate to you that there are exactly six entries (media) whose artist name has the initial substring “Elton John” (ignoring case, of course).

The error has the same meaning as for the $SEARCH$<CACHE><LIST> command.

Searching using an exact string – NAME only

Request

Command:	$SEARCH$	
Parameters:	<CACHE>	
	<FIND>	
	<MARKER>*marker*	
	<NAME>*str*	
	<FOR>	Optional

Reply

Command: ACK
Parameters:

Either:

<ERROR>
<MESSAGE>16Cache marker no longer valid

Or the following:

<OK>
<SEARCH>
<CACHE>
<FIND>
<MARKER>*marker*
<NAME>*str*

Followed by:
<NONE>

or:

<FROM>*fff*	
<FOR>*nnn*	Only if <FOR> given in the command

This searches the NAME field for the list in question. It is the same in all respects as $SEARCH$<FIND>…<START>, except that you are matching the exact name (case-independent), and there are no <NEXT> or <PREV> options.

Searching using an exact string – NAME and MEDIA (for those caches that support it)

Request

Command:	$SEARCH$	
Parameters:	<CACHE>	
	<FIND>	
	<MARKER>*marker*	
	<NAME>*namestr*	
	<START>*startstr*	
	<PREV>	Optional; may not be used with <NEXT>
	<NEXT>	Optional; may not be used with <PREV>
	<FOR>	Optional

Reply

Command: ACK

Parameters:

Either:

<ERROR>

<MESSAGE>16Cache marker no longer valid

Or the following:

<OK>	
<SEARCH>	
<CACHE>	
<FIND>	
<MARKER>*marker*	
<NAME>*namestr*	
<START>*startstr*	
<PREV>	Only if given in the command
<NEXT>	Only if given in the command

Followed by either:

<NONE>

or:

<FROM>*fff*	
<FOR>*nnn*	Only if <FOR>given in the command

This command can only be performed on markers for ARTISTMEDIA and GENREMEDIA caches.

If the NAME field for the list in question does not exactly match *namestr* (case-independent, of course), you will get <NONE>.

If it matches, then all MEDIA entries for that NAME are searched for an initial substring of *startstr*, and behaviour is analogous to <FIND>…<START>, except that MEDIA, instead of NAME, is being searched.

It is worth noting that the same result (<NONE>) is produced in each of these two conditions:

- The <NAME> didn’t exactly match any cache entry’s NAME field;
- The <NAME> matched, you didn’t give a <NEXT> nor <PREV> parameter, and the <START> substring didn’t match a corresponding <MEDIA> entry.

Closing a marker

Although not strictly necessary, it is polite to close markers, to allow the RCP to release any associated resources.

Request

Command:	$SEARCH$
Parameters:	<CACHE>
	<CLOSE>
	<MARKER>*marker*

Reply

Command:	ACK
	<OK>

This command always succeeds, even if the marker is invalid or already closed.

Advanced search facilities

This section gives a very general overview of the searching commands, their syntaxes, and a few examples of their uses -- however, this is primarily a protocol specification. You should refer to the companion Reference Manual for more information on how to use these commands.

What to use, and what not to use

We do **not** recommend the use of $SEARCH$<COUNT> and $SEARCH$<LIST>. Whilst these are remarkably powerful commands, they can take a considerable time to execute on larger databases (200 or more CD media, for example).

Whilst they are in progress, they prevent other commands to the `server` destination from executing, and this leads to a profound slow-down in performance, particularly for multi-room servers.

We are investigating alternative ways of implementing the same functionality. This may mean any of the following:

- The withdrawal of $SEARCH$<COUNT> and $SEARCH$<LIST> in favour of other commands offering similar capabilities;
- Significant changes to the syntaxes of those commands, to allow changes in the underlying implementation that would speed up command execution;
- A full reimplementation of the functionality, leaving those commands intact in their current form, and making them fast enough to be usable on a reasonably-populated system.

For all of these reasons, you should consider the following commands to be only partially supported under version 1.00, and therefore are not considered to constitute part of the formal 1.00 Specification. You use them at your risk. They may not continue to work exactly as described here:

- $SEARCH$<CATEGORIES>
- $SEARCH$<COUNT>
- $SEARCH$<LIST>
- $SEARCH$<JOIN>
- $SEARCH$<COMMIT><TAG>*tag* [4]
- $SEARCH$<DELETE><TAG>*tag*…

In general, any $SEARCH$ command that uses or refers to search tags is not considered formally specified at 1.00, although many servers which support 1.00 may also support the commands.

[4] Note that the $SEARCH$<COMMIT><ID>… variant is fully supported.

Of databases and categories

We define three databases: <TRACKDB>, <PLAYLISTDB> and <SPLISTDB>. These behave like individual databases, although this may not be how the underlying implementation arranges the information they represent. This is why we call a database of this time a virtual database (vDB). The columns or fields in a conventional database we call *categories*, and we call individual rows *lines*, to differentiate them from their similar, but not identical roles in classic databases.

First, a few general principles:

- Searches are always made with respect to searchable categories;
- A successful search will limit the number of possible selections (lines) from the database;
- The results of a search can be saved for future reference;
- Searches can be made incrementally upon the results of a previous search – in other words, you can use a chain of searches, each feeding upon the results of the previous, to minimise the number of matching entries;
- Successive searches <u>must</u> be made within the same database: mixing different databases in a chained (*incremental*) search is not permitted.

Before any search can be made on a database, the controller will need to know which searchable categories are available for it. The $SEARCH$<CATEGORIES> command is available for this purpose.

Categories identify classes of information. A category in the track database, for example, could be artist name, genre (rock? classical?), decade, or any other such. Any given category could have few or many members – for example, the list of genres will be much shorter than that of artist names.

Many categories are not unique: genre, for example, will have a limited number of possible values, and each value may apply to many entries in the track database. Equally, a playlist's TYPE field can only have the values SPLIST or DPLIST.

For any given database, there will be a number of categories guaranteed to be present. The descriptions of specific databases, below, will detail these. A few of those defined categories will be guaranteed to have a unique value for each entry.

A number of databases are available for search. For convenience, below, we identify the database parameter as <*vdb*>, which stands (for version 1.0) for one of the following:

- <TRACKDB>
- <PLAYLISTDB>
- <SPLISTDB>

The following sections provide detailed information on specific databases.

How to search

All searching begins with the $SEARCH$<COUNT> command. This counts the number of unique members of a given category that match the given criteria.

The simplest form of search on the track database would simply count the total number of entries in a given category: for example, the number of different genres. If it is not using the results of a previous search to constrain its scope, this will give the total number of matches over the whole database.

However, as mentioned above, searches can be chained. The results of a search can be saved, producing a **search tag**, which describes all the search criteria to date. This tag can then be provided to subsequent searches upon the same database**,** to constrain the list of entries over which the second search can operate.

Listing the entries

We’ve used the $SEARCH$<COUNT> command, possibly several times in succession, to constrain the list of tracks down to a small number which we want to examine in detail. The way to list these entries is through the $SEARCH$<LIST> command, which allows an iterative search for the contents of any given category. The $SEARCH$<LIST> command uses search tags saved from $SEARCH$<COUNT>.

There are two ways of using $SEARCH$<LIST>: unique and non-unique. If you use the <UNIQUE> parameter, the list will be of only the unique values for that category (as constrained). Without it, the value of that category for every entry in the constrained database will be listed, which will probably mean duplicates.

The command has mandatory <FROM> and <FOR> parameters. <FROM> specifies the start index; <FOR> specifies the maximum number of entries to list, for that category.

The list of entries is always enumerated starting from 1. Of course, there are usually a different number of entries for unique and non-unique lists from the same search, so the actual meaning of the index differs depending upon the type of list you’re making.

The track database (TRACKDB)

The track database is identified by the parameter <TRACKDB>. It lists all of the available tracks, along with a wealth of information naming the track, its artist, the medium from which it comes, the artist of the medium (which may be different, of course), and so forth. It also lists means by which the track can be categorised: for example, the decade, the genre of music, the tempo et cetera.

In the track database, the following categories are guaranteed to be present:

Category	Type	Meaning
TRACKID	Numeric	ID of the track
NAME	String	Name of the track
ARTIST	String	Name of the track's artist
MEDIAID	Numeric	ID of the track's media
MEDIANAME	String	Name of the track's media
TRACKNUM	Numeric	The track's number in its media (from 1)
GENRE	String	The track's media genre.

Table xiii: TRACKDB categories

The playlist database (PLAYLISTDB)

The playlist database contains all of the available information about playlists. It is identified by the parameter <PLAYLISTDB>. These are the guaranteed categories:

Name	Type	Meaning
PLAYLISTID	Numeric	ID of the playlist
PLAYLISTNAME	String	Name of the playlist
TYPE	String	Type (SPLIST or DPLIST) of the playlist.

Table xiv: PLAYLISTDB categories

The static playlist database (SPLISTDB)

The static playlist database contains a list of tuples of (static playlist ID, item ID). This makes it rather powerful. You can search the SPLISTDB with a given playlist ID, to get all the items (usually track IDs) saved under that ID; you can search it with a given item ID to find all the static playlists which contain it. You can even find out how many static playlists contain items whose name contains the word "hatstand"!

These are the guaranteed categories:

Name	Type	Meaning
PLAYLISTID	Numeric	ID of the playlist
PLAYLISTNAME	String	Name of the playlist
ITEMID	Numeric	ID of the item
ITEMNAME	String	Name of the item

Table xv: SPLISTDB categories

Search tags

The results of a $SEARCH$<COUNT>, which begins a search, can be saved as a search tag. A search tag represents the following pieces of information:

- The search criteria to date (the constraints);
- The database being searched;
- The list of items in that database which, at the time of searching, matched the criteria.

Tags have an indefinite lifetime. If they are not deleted, they persist. It’s important, therefore, to track them and to delete those no longer needed, otherwise they will accumulate. That accumulation may eventually cause degradation in performance, or even cause resources (memory or disk) to run low. A number of commands can cause tags to be deleted implicitly or explicitly, and one in particular can delete all outstanding tags.

Almost all versions of the XiVA software cause all outstanding search tags to be deleting when the server reboots. Since you cannot know how long it will be before the next reboot, however, you should not rely upon this behaviour to take care of your “housekeeping” for you.

Obtaining categories

WARNING – we do not recommend the use of this command. See **What to use, and what not to use**, on page 105.

Request

Command:	$SEARCH$	
Parameters:	<CATEGORIES>	
	<*vdb*>	Mandatory; identifies the database.

The sender wants to obtain the current list of searchable categories for the given database.

Remember, *vdb* is one of the following:

- TRACKDB
- PLAYLISTDB
- SPLISTDB

Reply

Command: ACK
Parameters: <OK>
<CATEGORIES>
Followed by one or more of these:
<CAT>name

Searching and counting

WARNING – we do not recommend the use of this command. See **What to use, and what not to use**, on page 105.

$SEARCH$<COUNT> is an extremely powerful command. It is used to examine the available databases at many levels of detail.

In its simplest form, it simply allows the caller to obtain the count of all of the unique names (or those matching pattern-matching criteria) in a given category.

In a slightly more advanced usage, it can create a tag (a stored record of the results of this search), which can be used to enumerate and list the unique names.

It can allow the results of previous searches to be progressively filtered by successive searches, allowing extremely sophisticated searches to be made.

Request

Command:		$SEARCH$	
Parameters:		<COUNT>	Mandatory
	either:		
		<vdb>	Optional – see later
	or:		
		<TAG>*tag*	Optional – see later
	and then:		
		<CAT>*name*	Mandatory
		<NOT>	Optional – modifies <MATCHING> or <LIKE>
		<LIKE>*string*	Optional – incompatible with <MATCHING>
		<MATCHING>*string*	Optional – incompatible with <LIKE>
		<CASE>	Optional – modifies <MATCHING>
		<SAVE>	Optional

If supplied, *<vdb>* will be the identifier for a specific database. At version 1.0, it must be either <TRACKDB>, <PLAYLISTDB> or <SPLISTDB>.

If supplied, the <TAG> must be a valid tag from a previous search, in which case this count progressively narrows that search, and the database to use is inherited from that search.

One of *<vdb>* or <TAG> must be supplied, but not both; the two are mutually incompatible.

<CAT> introduces the name of a category. It is mandatory.

<NOT> may only be used if <LIKE> or <MATCHING> is also specified, and reverses their meaning.

<LIKE> gives simple string matching, using a wildcard character of '*', which matches zero or more characters. Matching using <LIKE> is case sensitive. <LIKE> is incompatible with <MATCHING>.

<MATCHING> allows Unix-style regular expression matching. See man egrep(1) for more details of how this can be used. <MATCHING> is incompatible with <LIKE>. By default, <MATCHING> is case insensitive:

<CASE> may only be used if <MATCHING> is also specified. <CASE> makes <MATCHING> case sensitive.

You do not have to supply either <LIKE> or <MATCHING>. If you supply neither, you will match all items in the category, the identical effect to "<LIKE>*".

<SAVE> requires that the results of the search be saved, and that the result be given an arbitrary tag to allow incremental searches to be made. If this command includes a <TAG>, making it an incremental filter on a previous search, that previous tag remains valid afterwards, allowing a search to be "branched".

Reply

Command:	ACK	
Parameters:	<OK>	
	<COUNT>*nnn*	*nnn* is a positive integer or zero
	<TAG>*tag*	Optional

The <COUNT> is the number of unique names that satisfy the matching criteria given. If no matching criteria (<LIKE> or <MATCHING>) were given, the <COUNT> is the total number of names in the category.

If the request included a <RECYCLE> or <SAVE> parameter, and if this search produces a non-zero count, the reply will include a <TAG> assigned by the server.

In the case of <RECYCLE>, this will be the same as the <TAG> given in the request.

In the case of <SAVE>, this will be a newly created tag.

The meaning of any given tag is opaque to the sender, who should simply regard it as a "magic cookie", and not attempt to divine any meaning from it.

Enumerating a search

WARNING – we do not recommend the use of this command. See **What to use, and what not to use**, on page 105.

Note that searches are enumerated from 1, not from 0. Enumeration relies upon the use of tags from prior searches (see previous). The database is not specified; this is implicit in the search tag.

Request

Command:	$SEARCH$	
Parameters:	<LIST>	Mandatory
	<TAG>*tag*	Mandatory
	<ORDERED>	Optional
	<CAT>*category*<CAT>*category…*	At least one of these
	<UNIQUE>	Optional
	<FROM>*nnn*	Mandatory; *nnn* is a positive integer
	<FOR>*nnn*	Mandatory; *nnn* is a positive integer

This requests the contents of the given categories for the matching database lines.

The tag can be from a $SEARCH$<COUNT>…<SAVE> on any of the vDBs.

The order of the <CAT>s is significant. The first <CAT> is the most important. If you supply the <ORDERED> parameter, the list will be ordered (sorted) by this first category; if you supply the <UNIQUE> parameter, the list will be unique in this first category (but not necessarily by others).

If <UNIQUE> is not given, there will be as many results as there are individual entries which match the search criteria, and the index will be over that many entries.

The <FROM> parameter specifies the initial index (starting from 1, not 0); the <FOR> parameter specifies the number of lines to return. You may not get back as many lines as you request, if there is not enough space in the packet, or not enough remaining that satisfy your criteria.

It is not an error to request more than there are available. In fact, one perfectly reasonable way of iterating over the results of a search is to make the first <LIST> as <FROM>1<FOR>999, and then keep asking <FOR>999, starting <FROM> where the previous ACK list left off, until you get an <EOF>.

Reply

Command:	ACK	
Parameters:	<OK>	
	<LIST>	
	<TAG>tag	Same as request
	<FROM>*nnn*	Same as request
	<FOR>*mmm*	May be less than request

Then 'mmm' entries as follows:

<AT>*xxx*	(For *xxx* = *nnn* to (*nnn*+*mmm*-1))

Followed by one or more of this:

<HAS>*entry*	One for each <CAT> given

After which:

<EOF>	Optional; if this completes enumeration

Note that the number in the <FOR> entry may be less than requested, either because there are fewer entries remaining than requested, or because not all of them could be fitted into the packet maximum size.

Each 'line' of data starts with <AT>*xxx* (where *xxx* is the line number in the list)

Note that a <HAS> entry can contain no data (in other words, there is no argument to the <HAS> parameter), if the corresponding category in that line of the database is empty.

If there are no more matching lines after this enumeration, there will be an <EOF> parameter.

Joining searches

WARNING – we do not recommend the use of this command. See **What to use, and what not to use**, on page 105.

This command allows the results of more than one search to be combined. All search tags must be over the same database. For vDBs that support the <COMMIT> operation, you cannot <COMMIT> a <JOIN>ed tag to a <DPLIST>, only an <SPLIST>.

Request

Command:	$SEARCH$	
Parameters:	<JOIN>	
	<TAG>*tag*	Mandatory, one or more.
	<DELETE>	Optional

If the <DELETE> parameter is supplied, all tags listed will be deleted after the new tag has been created. This is to assist client programs "housekeep" their search tags.

Reply

Command:	ACK
Parameters:	<OK>
	<TAG>*tag*

This command always generates a new tag regardless.

Tricks and tips

If, for some reason, you want to duplicate an existing search, you can use this command with a single <TAG> argument and without the <DELETE> argument to create a duplicate search. Changes to the tag from which the new one was derived will not affect the new tag. Note carefully the comments above about not being able to <COMMIT> a <JOIN>ed tag to a <DPLIST>, though.

Committing category searches

WARNING – we do not recommend the use of this command. See **What to use, and what not to use**, on page 105.

Request

Command:	$SEARCH$	
Parameters:	<COMMIT>	
	<TAG>*tag*	Mandatory
	<DPLIST>	Optional; incompatible with <SPLIST>
	<SPLIST>	Optional; incompatible with <DPLIST>
	<NAME>*name*	Mandatory
	<DELETE>	Optional
	<REPLACE>	Optional

The <TAG> must have been the result of a previous search over <TRACKDB> or <SPLISTDB>, or the results of a $SEARCH$<JOIN>.

The <NAME> parameter gives the playlist created by the search a name, and saves it permanently for future use. Unless the <REPLACE> parameter is present, this name must be unique over all existing playlists, regardless of type. The command will fail otherwise.

<DPLIST> (dynamic playlist) and <SPLIST> (static playlist) determine the type of playlist being created.

One of <DPLIST> or <SPLIST> (but not both) must be given.

There are restrictions upon which types of search tag are suitable for saving as a <DPLIST>. At present, the only tags suitable are those which are the result of a $SEARCH$<COUNT><TRACKDB>. You cannot save a tag which resulted from a $SEARCH$<COUNT><TAG>… or $SEARCH$<JOIN> as a <DPLIST>.

If the <DELETE> parameter is given, and the command succeeds, the tag is discarded after the playlist has been saved; otherwise, the tag remains valid.

If the <REPLACE> parameter is given, any existing playlist named *name* will be overwritten by the newly created one.

It is an error to attempt to <COMMIT> a search that has zero entries, or an invalid tag, or a search tag from any database other than the track database, or a search tag which is unsuitable for the type of playlist being saved.

Reply

Command:	ACK	
Parameters:	<OK>	
	<PLAYLIST>*id*	ID of the newly-created playlist

Committing a playlist without searches

If you have a list of track or media IDs, you can create or append to a static playlist without having to search first.

Request

Command:	$SEARCH$	
Parameters:	<COMMIT>	
	<ID>*id*<ID>*id*…	*Up to version 1.01:* One or more of these
		From version 1.02: Zero or more of these
and then either (provided at least one <ID> is specified above):		
	<PLAYLIST>id	The ID for a static playlist already known
or:		
	<NAME>*name*	A new, unique name.
	<REPLACE>	Optional

If the <PLAYLIST> parameter is used, the IDs are appended to the preexisting static playlist with that ID; if the <NAME> parameter is used instead, the command will try to create a new static playlist with the given name.

It is an error to provide an invalid <PLAYLIST> ID, or one for a dynamic playlist. It is also an error to supply a <NAME> which duplicates one already extant, unless the <REPLACE> parameter is present; in the latter case the original playlist is overwritten by the new one.

Reply

Command:	ACK	
Parameters:	<OK>	
	<PLAYLIST>*id*	*id* is an opaque identifier

If the command used <PLAYLIST>, the one in the reply will be the same. If the command used <NAME>, the <PLAYLIST> in the reply will be that of the newly-created static playlist.

The $SEARCH$<COMMIT><NAME>*name* variant of this command (*introduced in Protocol version 1.02)* simply allows an empty playlist to be created. Previously this was not permitted.

Deleting search tags

WARNING – we do not recommend the use of this command. See **What to use, and what not to use**, on page 105.

Request

Command:	$SEARCH$	
Parameters:	<DELETE>	
	<TAG>*tag*<TAG>*tag…*	At least one of these

This seeks to delete the given search tags. This is a good thing to do, as search tags remain valid until destroyed, and will use up precious memory and disk resources. This command affects all types of search tags, however created.

Reply

Command:	ACK
Parameters:	<OK>

Renaming a playlist

Request

Command:	$SEARCH$
Parameters:	<RENAME>
	<PLAYLIST>*old-name*
	<TO>*new-name*

The replacement name given by <TO> must be unique over all existing playlists, regardless of type. The command will fail if it is not.

Reply

Command:	ACK
Parameters:	<OK>

Reporting an external modification of the database

[Request (and reply) introduced in Protocol version 1.02]

This request is provided for the use of systems that change the database in a manner which otherwise bypasses XiVA™-Link Protocol. It is unlikely that a normal controller will need to use it. It essentially forces the server to reload all its caches and send appropriate updates to those controllers which have switched them on.

Request

Command:	$SEARCH$
Parameters:	<RESET>
	<TRACKDB>

Reply

Command:	ACK
Parameters:	<OK>

Online operations

The server must be put online in order to communicate with Internet CD databases. These commands deal with the management of this process.

Bear in mind that more than one source may want the server online, or offline. These are the principal features of the operation:

- When no source is still requesting the server to be online, it will go offline;
- If at least one source requests it go to online, it will attempt to do so;
- It may not succeed in the attempt to go online;
- It may take an indefinite period (including possible retries) to do so;
- If an online connection is idle for more than a given period of time, the connection may drop.

As a result of these, any requests to change the online status are considered to be suggestions, rather than mandates. The server will notify interested parties when its online status changes, if they have requested updates.

All of these commands should be sent to the **`server`** destination-id.

Requesting on/offline state

Request

Command: $ONLINE$
Parameters: <WANT>
then either:
<ON>
or:
<OFF>

Exactly one of <ON> or <OFF> must be supplied.

Reply

Command: ACK
Parameters: <OK>

if the server enters or was already in the requested online state or:

Command: ACK
Parameters: <ERROR>
<MESSAGE>*XXmessage*

[Introduced in Protocol version 1.02]
<TYPE>*IXXdetail* Optional: if detail about the failure is available

The components of *IXXdetail* are :-

- *I* : a single hex digit indicating the network interface involved (currently `0' for the first ethernet interface or `1' for the first dialup modem interface);
- *XX* : an error detail code (in the form of two hex digits) which is specific to the type of network interface and
- *detail*: a textual description of the type of failure.

Networking error detail codes

The following table provides an interpretation of the error detail codes for a dialup modem connection. They are grouped by shading into general errors, errors related to establishing a telephone connection, errors related to negotiating an Internet connection and account management errors. Some should only arise in the event of a faulty installation. Informative error detail codes have not yet been specified for ethernet connections but this is expected in a future version of the Protocol.

XX=	Means
00	OK : The connection succeeded.
01	Timed out : The connection attempt did not succeed within the timeout period.
02	Aborted by user
03	Port busy
04	Modem not responding
05	Bad modem init string
06	No carrier
07	No dial tone
08	Line busy
09	Bad dial command
0a	Voice line detected
0b	Fax line detected
0c	Number blacklisted (Power cycle the box to clear this error for a given phone number.)
0d	Unknown dialling error
0e	Fatal PPP (Point-to-Point Protocol) error
0f	Bad PPP options
10	PPP incorrectly configured
11	PPP interrupted
12	PPP port problem
13	Connect script failed
14	PPP negotiation failed
15	Peer authentication failed
16	Idle connection timeout
17	Connection timeout
18	Peer not echoing
19	Modem hung up
1a	Loopback detected
1b	Init script failed
1c	Authentication (of system by ISP) failed
1d	Unknown PPP error
1e	Unknown error
1f	No dial-up account configured
20	Missing default ISP account
21	Bad default ISP account
22	Default ISP account problem
23	Problem updating ISP account

Table xvi: Dialup modem connection error detail codes (<TYPE>1XXdetail)

Checking online status

Request

Command:	\$STATUS\$
Parameters:	<ONLINE>

Reply

Command:	\$ACK\$
Parameters:	<OK>
	<ONLINE>*status*

The *status* argument is one of the following:

- YES (the server is online);
- NO (the server is not online, and is not attempting to connect);
- UNKNOWN (the online status is unknown at present);
- CONNECTING (the server is in the process of trying to connect);
- DISCONNECTING (the server is in the process of trying to disconnect).

Requesting a TRACKDB update

Request

Command:	$ONLINE$
Parameters:	<UPDATE>
	<TRACKDB>

The command is slightly confusingly named: unlike the $STATUS$<UPDATE> commands, it is not requesting status updates (see the section "**Update notifications**" elsewhere in this Specification).

This command requests the server (if it is online) to update the contents of its database from online disc databases – for example, CDDB or similar. It is the controller's responsibility to ensure that the server is indeed online before performing this operation, which will fail if it is not.

When you initiate this command, it triggers a sequence of operations that can take some time, or even fail.

Reply

Command:	ACK
Parameters:	<RXD>

then if the lookup is completed (some or all of the discs may not be found in the database):

Command:	ACK	
Parameters:	<OK>	
	<TOTAL>*total*	*total* is the number of items to be looked up
	<FAILED>*failed*	*failed* is the number not found in the database

else if the lookup failed (communication with the database failed or was never established):

Command:	ACK	
	<ERROR>	
	<MESSAGE>15Lookup failed	
	<TOTAL>*total*	As above
	<FAILED>*failed*	As above
	<LEFT>*left*	*left* is the number of items which remained to be looked up when communication failed

[Introduced in Protocol version 1.02]

else if the lookup was aborted (see below):

Command:	ACK	
Parameters:	<WARNING>	
	<MESSAGE>89Aborted by user	
	<TOTAL>*total*	As above
	<FAILED>*failed*	As above
	<LEFT>*left*	*left* is the number of items which remained to be looked up when lookup was aborted

Note that early versions of the server did not report failure of communication with the online database as described above; instead they returned: ACK<OK><TOTAL>*total*<FAILED>*total*. This situation was thus indistinguishable from successful communication with a database which did not contain any of the items looked up.

Aborting a TRACKDB update

[Introduced in Protocol version 1.02]

The database update operation initiated by an $ONLINE$<UPDATE><TRACKDB> request (See above) may take some time, particularly if requests to the remote services involved (possibly including DNS servers) must be repeated, e.g. due to heavy loading of the latter. This new request allows the update operation to be cancelled directly. It will fail if there is no lookup in progress or a lookup is already in the process of being cancelled.

Request

Command:	$ONLINE$
Parameters:	<UPDATE>
	<TRACKDB>
	<ABORT>

Reply

Command:	ACK
Parameters:	<OK>

Recording – ripping discs

All of these commands (unless otherwise stated) should be sent to the `server` *destination-id*.

You will see, in each command description that refers to a disc, the cryptic phrase <*DISCTYPE*>*discnum*. Initially, the only *DISCTYPE* will be CD. The *discnum* refers to the disc unit of that type, and these are numbered from 1 onwards, with leading zeroes ignored. So, where you see <*DISCTYPE*>*discnum*, you will probably be using <CD>1 (or <CD>01, which is the same thing) initially.

In future, we intend to expand this (adding more *DISCTYPE*s) to include DVDs, minidiscs, CD recording technology, and so on.

When performing any command upon a given disc unit, you may get the error:

ACK<ERROR><MESSAGE>0eDevice busy

This indicates that the drive is in use and cannot perform the requested command.

What happens when ripping

First, the user places a new CD in the CD drive.

At this point, if possible, or at a later time when prompted by a controller, the server inspects the CD, and derives from information on the disc a signature (this has no relation to track or media IDs). It can use this signature to identify the disc to online databases.

The server checks to see if the CD is already logged in the media database. If it is, there is no further work to do until ripping. If it is not, the new CD is logged in the media database with only the information that can be obtained from the disc: signature, number of tracks and (maybe) track lengths. A later online update can obtain from the online database (or similar) the extra information, such as album, track and artist names, genre and so on, and update the server's database, but ripping can proceed without having gone online first.

Finding out which discs are configured

Request

Command:	$STATUS$
Parameters:	<DRIVES>

Reply

Command:	ACK
Parameters:	<OK>
	<DRIVES>
	followed by zero or more of these:
	<DISCTYPE>nn	*nn* is the number of drives for this *DISCTYPE*

The only *DISCTYPE* defined at 1.00 is CD. In future, you may also see parameters such as “<DVD>*nn*”, and similar.

Note: some early versions of the server software had a bug in which the <DRIVES> parameter in the reply was accidentally omitted.

Querying a drive’s status

Request

Command: $STATUS$
Parameters: *<DISCTYPE>discnum*

Reply

```
Command:     $ACK$
Parameters:  <OK>
             <DISCTYPE>discnum
       then either:
             <NONE>                    No disc loaded
       or:
             <INVALID>                 Disc corrupt or of unknown type
       or one or both of:
                   <DATA>              Disc contains data tracks
             and/or:
                   <AUDIO>nn           Disc contains nn audio tracks,
                   <MEDIA>
                   <ID>id              and has media id id
```

Describing a drive's media

Request

Command:	$STATUS$
Parameters:	<*DISCTYPE*>*discnum*
	<MEDIA>

Reply

Command:	ACK	
Parameters:	<OK>	
	<*DISCTYPE*>*discnum*	
	<MEDIA>	
then:		
	<NONE>	Meaning no disc is loaded
or:		
	<ID>*id*	
	<NONE>	Meaning only the media ID is available
or:		
	<ID>*id*	
	<NAME>*media-name*	*media-name* is a string
	<ARTIST>*media-artist*	*media-artist* is a string
	<GENRE>*genre*	*genre* is a string
	<LOOKUP>*yesno*	*yesno* is either YES or NO
	[Introduced in Protocol version 1.02]	
then, provided a disc is loaded:		
	<MEDIA>	
	<NUM>*media-num*	*media-num* is the media's media number (c.f. `Select media by media number', p 50).

If an online database lookup (see $ONLINE$<UPDATE><TRACKDB>, p 128) has not been performed or the media was not found in the online database used and the media details have not been edited, then *media-name* and *media-artist* are likely to be temporary names assigned by the server. The *yesno* argument for <LOOKUP> is NO unless the media has been successfully looked up.

If an online database lookup (see $ONLINE$<UPDATE><TRACKDB>, p 128) has not been performed, this will lead to less information being available. Unless the media details have been edited,

As of *version 1.02 of the Protocol* there are alternatives to this request for controllers with input buffers of less than 1024 bytes (see page 184 and after).

Describing a drive’s media’s tracks

Request

Command:	$STATUS$	
Parameters:	<*DISCTYPE*>*discnum*	
	<TRACK>	
	<FROM>*mm*	Optional; default is 1
	<TO>*nn*	Optional; default is last track

Note that track numbers are numbered from one, not zero. Note also that you may not get all the track information you requested, if it could not be made to fit into one packet. If so, you should make repeated calls, each <FROM> the entry following the last in the previous reply, until you get all of the entries you need.

Reply

Command:	ACK	
Parameters:	<OK>	
	<*DISCTYPE*>*discnum*	
	<TRACK>	

then either:

<NONE>	Meaning that no disc is loaded

or:

<FROM>*mm*	*mm is* same as for request
<TO>*tt*	*tt* may be less than in request

followed by either:

<NONE>	No information is available at present

or (tt-mm+1) entries, each with up to three parameters of the following format:

<TRACK>*xx*	For *xx* from *mm* to *tt* inclusive
<LEN>*hhhh:mm:ss*	Optional; only if this information is available
<NAME>*track-name*	Optional; only if this information is available

[Introduced in Protocol version 1.02]

<EOF> if the last track listed above is the last on the disk

If an online database lookup (see <ONLINE><UPDATE><TRACKDB>) has not been performed, this will lead to less information being available.

Opening or closing the drive door (ejecting or loading a disc)

Request

Command: $RECORD$
Parameters: *<DISCTYPE>discnum*

and then:

<OPEN >

or*:*

<CLOSE>

[Introduced in Protocol version 1.02]

or:

<TOGGLE>

If the disc device does not have a drive door or tray (for example, if it is a slot-loader), the OPEN or CLOSE operations will correspond to the equivalent operations for ejecting or loading a disc, respectively. The <TOGGLE> variant of this command toggles the state of the drive tray. It will only succeed if the tray has reached a stable state (open or closed) before it is received.

Loading a disc has the effect of causing it to be registered with the server's database, even if no tracks are recorded.

Reply

Command: ACK
Parameters: <OK>

If the operation is not supported at all by the device, the command will normally succeed silently without effect.

Rereading the drive’s media information

Request

Command:	$RECORD$
Parameters:	*<DISCTYPE>discnum*
	<READ>

Reply

Command:	ACK
Parameters:	<OK>

Recording a disc's contents

Request

Command:	$RECORD$	
Parameters:	*<DISCTYPE>discnum*	
	<COMPR>*compr-id*	Optional : see below

then either:

zero, one or both of the following:

<FROM>*mm*	Optional; *mm* defaults to 1
<TO>*nn*	Optional; *nn* defaults to last track

or one or more of:

<TRACK>*ttt*

If a <COMPR> parameter is supplied, *compr-id* (which specifies the encoding to use) must be one of the values returned in the reply to a $STATUS$<COMPR>*tracktype* request (see **Track encoding types**, page 60). The encoding used in the absence of a <COMPR> parameter is the server's default encoding for that track type.

Up to *version 1.01 of the Protocol*, the default encoding was an uncompressed one (e.g. WAV for AUDIO).

As of *version 1.02 of the Protocol*, the default encoding is a configuration item (e.g. AUDIO) under the DefaultEncodings category (see **Configuring the server**, page 149). Note that the factory setting of these defaults may not match the pre-1.02 defaults.

Reply

Command:	ACK
Parameters:	<RXD>

then when recording ends (whether it completes, fails or is aborted):

Command:	ACK	
Parameters:	<OK>	
	<RECORD>	
	<DISCTYPE>discnum	
	<TOTAL>*total*	*total* is the number of tracks requested
	<FAILED>*failed*	*failed* is the number which could not be recorded
	<LEFT>*left*	*left* is the number which were not attempted (only non-zero if recording ended prematurely)

Note: If you need to be notified of recording progress, request the corresponding update (see **Record state update** on page 69).

Aborting a ripping operation

Request

Command:	$RECORD$
Parameters:	*<DISCTYPE>discnum*
	<ABORT>

Reply

This indicates that the operation has been terminated. You may get an <RXD> first if it is likely to take a while.

Command:	ACK
Parameters:	<OK>

Finding out the ripping status

Request

Command:	$STATUS$	
Parameters:	<*DISCTYPE*>*discnum*	
<RECORD>		

Reply

Command:	ACK	
Parameters:	<OK>	
	<*DISCTYPE*>*discnum*	
	<RECORD>	
then:		
	<NONE>	No operation in progress
or:		
	<TRACK>*tracknum*	The track number (from 1) currently recording
	<POS>*pp*	*pp* is a percentage-complete for this track

Altering database contents

The following commands give a quick and easy way of making simple changes to information held in the track database.

They must all be sent to `server`.

Changing track information

Request

Command:	$ALTER$	
Parameters:	<TRACK>	
	<ID>*track-id*	
	<NAME>*name*	Optional
	<ARTIST>*artist*	Optional

At least one of <NAME> and <ARTIST> must be supplied.

If the <NAME> parameter is supplied, the name of the track is changed to the given text.

If <ARTIST> is supplied, the artist name of the track is changed to the given text.

Reply

Command:	ACK
Parameters:	<OK>

This reply indicates that the changes have successfully been implemented. This does not mean that the changes are permanent: if the server were to lose power, crash or reboot within a very few seconds of the command completing, the changes may not have been written to permanent storage in time.

Changing media information

Request

Command:	$ALTER$	
Parameters:	<MEDIA>	
	<ID>*id*	
	<NAME>*name*	Optional; see below
	<ARTIST>*artist*	Optional; see below
	<GENRE>*genre*	Optional; see below

At least one of <NAME>, <ARTIST> and <GENRE> must be supplied.

If the <NAME> parameter is supplied, the name of the media is changed to the given text.

If <ARTIST> is supplied, the artist name of the media is changed to the given text.

If <GENRE> is supplied, the media will be classified under the given genre, although **NOTE** that this must only be one of the already-entered genre types.

You can use use the $SEARCH$ commands to find out the available genres: in particular, $SEARCH$<CACHE> on the GENRE cache is useful.

Reply

Command:	ACK
Parameters:	<OK>

This reply indicates that the changes have successfully been implemented. This does not mean that the changes are permanent: if the server were to lose power, crash or reboot within a very few seconds of the command completing, the changes may not have been written to permanent storage in time.

Changing playlist information

Request

Command:	$ALTER$	
Parameters:	<PLAYLIST>	
	<ID>*id*	
	<NAME>*name*	
	<REPLACE>	Optional

Changes the name of the playlist to the given text.

Normally, you may not change to a pre-existing name; however, if the optional <REPLACE> parameter is supplied, and there is a pre-existing playlist with the same name, that playlist will be deleted before the list given by *id* is renamed.

Reply

Command:	ACK
Parameters:	<OK>

This reply indicates that the changes have successfully been implemented. This does not mean that the changes are permanent: if the server were to lose power, crash or reboot within a very few seconds of the command completing, the changes may not have been written to permanent storage in time.

Deleting database contents

These commands are used to delete tracks, media or playlists from the database. Use them with caution!

Delete a track (and its corresponding media file)

Request

Command:	$DELETE$
Parameters:	<TRACK>
	<ID>*id*

Reply

Command:	ACK
Parameters:	<OK>

If successful, the track, and its corresponding media file, have been removed from the server.

Delete media

Request

Command:	$DELETE$	
Parameters:	<MEDIA>	
	<ID>*id*	
	<TRACK>	Optional

If the optional <TRACK> parameter is present, all track entries and their corresponding media files will also be deleted.

If it is not present, only the entry for the media itself will be removed; the tracks and their media files still exist, and can be searched for. In this way, you can remove the knowledge of the media from the server, whilst keeping unchanged any playlists which may include the tracks the media contained.

Reply

Command:	ACK
Parameters:	<OK>

If successful, the media has been deleted from the server.

Delete playlist

Request

Command:	$DELETE$
Parameters:	<PLAYLIST>
	<ID>*id*

This seeks to delete a playlist of a known ID. To prevent accidental deletions, you must have the ID; there is no provision for deleting a playlist by name.

Reply

Command:	ACK
Parameters:	<OK>

If successful, the playlist has been deleted from the server. The items that playlist references still exist: this command does not delete associated tracks etc.

Configuring the server

The following commands are all related to changing parts of the server's permanent configuration.

Configuration items are grouped under *categories*. Within each category, there are one or more *items*.

An item will have an *itemtype*, which identifies how the item is stored. Valid itemtypes are:

- STRING – a string which can be modified;
- RO_STRING – a string which cannot be modified, and is given for information only;
- LIST – a list of items, effectively a "menu" from which only one can be selected;
- BOOL – a true/false value, given as a string. If the first character is 't', 'T', 'y' or 'Y' the value *true* is assumed: any other value is taken to mean *false*;
- INTEGER – a 32 bit signed number with no fractional part;
- ADDRESS – a dotted-quad IP address in the form "*a.b.c.d"* (without the quotes), where *a, b, c* and *d* are values in the range 0…255. They do not have to be zero-padded to three digits. For example, "192.168.0.1" is a valid ADDRESS;
- TIME – a date/time value of the form: "*hh*:*mm*:*ss YYYY-MM-DD"* (without the quotes, but with the space in the middle). *hh*,*mm* and *ss* are hours, minutes and seconds respectively; *YYYY* is four-digit year; *MM* and *DD* are numeric month and day respectivel. For example, "23:59:59 1999-12-31" is a valid TIME field).

As at 1.00, the TIME type is not yet used.

The LIST type

The LIST type requires a little more explanation. It has a number of possible values, all of which are preset.

You can use $CONFIG$<READ>*category*<LIST>*item* to obtain all of its possible values, and the $CONFIG$<SET>*category*<ITEM>*item* to set it to one of those possible values.

To obtain all of its possible values, use the $CONFIG$<READ>*category*<LIST>*item* command.

The effect of configuration

When you change a configuration item, that changed value is stored permanently.

It may also have an immediate effect, or may only come into effect following the next system reboot.

The following table lists some of the configuration items defined under version 1.00. The ***reboot*** column indicates whether the server must be rebooted in order for the change to take effect:

Configuration and destination-id

Configurable items are specific to each *destination-id*. Those that are configurable under `server` affect the system as a whole; those that are configurable under playout destinations affect only that specific destination for which the configuration is set.

You should obtain the lists of categories and items for each *destination-id* you are planning to configure.

Category	**ItemName**	**Type**	***Reboot***	***Dest. type***
Network	Device	LIST	No	`server`
Ethernet	Mode	LIST	No	`server`
	Hostname	STRING	No	`server`
	IPAddress	ADDRESS	No	`server`
	IPMask	ADDRESS	No	`server`
	Gateway	ADDRESS	No	`server`
	IPDNS1	ADDRESS	No	`server`
	IPDNS2	ADDRESS	No	`server`
	AssignedIPAddress	RO_STRING	-	`server`
	AssignedIPMask	RO_STRING	-	`server`
Modem	BaudRate	LIST	No	`server`
	Port	STRING	No	`server`
	InitialisationString	STRING	No	`server`
	NumberOfRetries	INTEGER	No	`server`
	AssignedIPAddress	RO_STRING	-	`server`
	ServerIPAddress	RO_STRING	-	`server`
Account	Username	STRING	No	`server`
	Password	STRING	No	`server`
	PhoneNumber	STRING	No	`server`
Diagnostic	UnderrunThreshold	INTEGER	No	(playout)
	DropThreshold	INTEGER	No	(playout)
	SingleShotMode	BOOL	No	(playout)
Information	SerialNumber	RO_STRING	-	`server`
(This section	ProductionDate	RO_STRING	-	`server`
is read-only)	XiVASoftwareVersion	RO_STRING	-	`server`
[Introduced in Protocol version 1.02]				
DefaultEncodings	AUDIO	INTEGER	No	`server`
	VIDEO	INTEGER	No	`server`
PortUsers	*<a port, e.g. serial0>*	LIST	No	`server`
PortUserSettings	*<port>:<app>:<setting>*	*(per item)*	*(per item)*	`server`
NumCtrlrChans	*<a controller>*	RO_STRING	-	`server`
Controllers	*<a controller>*	STRING	No	(playout)
Modem	Autoanswer	BOOL	No	`server`

Table xvii: Configuration items

The items under the new configuration categories introduced in version 1.02 of the Protocol are generally used in more complicated ways than their predecessors :-

- Items under DefaultEncodings store the per-track-type encodings which will be used for a recording if no <COMPR> parameter is specified in a $RECORD$ request (See **Recording a**

disc's contents, page 138). Valid values for these items may be obtained using $STATUS$<COMPR>*tracktype* requests (see **Track encoding types**, page 60).

- The items under PortUsers are the names of available hardware ports, e.g. *serial0* for the first serial port. The value for each port will indicate the type of communication which is expected to occur via that port; internally it specifies the application which should listen on that port. For example, valid values for *serial1* might be *XiVALink* and *Infrared*. Any configurable settings for these applications are stored under PortUserSettings. The format of the item reflects the fact that these settings are per-port as well as per-application. For example, *serial1:XiVALink:baud* specifies the baud rate for XiVALink communication using the second serial port.
- The items under Controllers are the names of supported remote controls. The string value associated with each remote control under the Controllers category for a particular playout zone comprises a comma-separated list of the channels of that remote control which control that playout zone. The number of control channels provided by any supported remote control may be read from the corresponding entry under the server-only NumCtrlrChans category.

Note: the string values read from Username and Password under the Account category will comprise eight asterisks, so as to conceal the real values once they have been entered. Any editor for these settings should either not read the concealed values at all (since they convey no information) or convert each of them to an empty string if and when the user starts to edit it. As a safeguard, attempts to write an eight-asterisk string as the actual value of one of these settings will be rejected.

Note: the Autoanswer item under Modem is the only (*at version 1.02 of the Protocol*) configuration item which is accessible when the server is in standby mode (see **Querying and setting the power mode** on page 157). None of the other items may be read or set in this mode.

Note: if Autoanswer is set to YES, the server will pick up any incoming call on the phone line to which it is attached. This mode is intended for remote diagnostics and it is recommended that any controller using it should make it obvious to the user that the server is in this mode. For example, a server in this mode and sharing a phone line with telephone handsets would intercept voice calls destined for the latter.

Note: some early servers have a bug that means they do not support reading values under the "Diagnostic" category, although you can still set the values. You should not rely upon being able to read these values.

Getting the categories

Use this command to get the list of configurable categories.

Request

Command:	$CONFIG$
Parameters:	<CATEGORIES>

Reply

Command:	ACK
Parameters:	<OK>
	<CATEGORIES>

Followed by zero of more of the following:

<CAT>*category*

Examining a category’s list of items

Request

Command:	$CONFIG$	
Parameters:	<LIST>*category*	As supplied by $CONFIG$<CATEGORIES>
	<FROM>*fff*	Optional
	<FOR>*rrr*	Optional; only allowed if <FROM> given

This command seeks to enumerate the configurable items under category *category*, which must be exactly as given in the reply to the $CONFIG$<CATEGORIES> command. The list numbers from 1 onwards.

If <FROM> is not used, the default start number is 1.

The <FOR> parameter can only be used if <FROM> has been given. If it is not used, the default is the last in the list.

Reply

Command:	ACK	
Parameters:	<OK>	
	<CONFIG>	
	<LIST>*category*	
	<FROM>*fff*	
	<FOR>*nnn*	May not be the same as *rrr*

Followed by nnn *entries of the form:*

<AT>*iii*	
<HAS>*itemname*	
<TYPE>*itemtype*	Optional – only if information available

And finally:

<EOF>	Optional – see below

The *nnn* argument may be less than *rrr* in the original request, if there were not that many entries, or if not all of them could be fitted into the reply packet. If this is the case, you can repeat the command, starting with *newfff* = *fff* + *nnn*, as many times as you need in order to get the full list.

If the type of the item is known, the <TYPE> parameter is given. If it is not, you should treat it as a string.

If last entry in the packet is also the last entry of the full list of items available for that category, the last entry will be followed by <EOF>.

It is not an error to specify a *fff* parameter that is greater than the total number of items available for the category. If this happens, *nnn* will be zero, there will be no <AT> entries, and there will be an <EOF> to indicate that there are no further entries to be had.

Read the values of a LIST item

This command applies only to items whose *itemtype* (see $CONFIG$<LIST>) is LIST.

Request

Command:	$CONFIG$	
Parameters:	<READ>*category*	
	<LIST>*item*	
	<FROM>*fff*	Optional
	<FOR>*rrr*	Optional; only allowed if <FROM> given

This command seeks to enumerate the elements of an item whose *itemtype* is LIST. The list is indexed from 1 onwards. (The first item is no. 1, the second no. 2, and so on.)

If <FROM> is not used, the default start number is 1.

The <FOR> parameter can only be used if <FROM> has been given. If it is not used, the default is the number of items remaining between *fff* and the end of the list (which may be zero, if the list has less than *fff* items – this is not an error).

Reply

Command:	ACK	
Parameters:	<OK>	
	<CONFIG>	
	<READ>*category*	
	<LIST>*item*	
	<FROM>*fff*	
	<FOR>*nnn*	
Followed by nnn *entries of the form:*		
	<AT>*iii*	
	<HAS>*listitemval*	
	<TYPE>*listitemtype*	Optional; only if information available
Followed by:		
	<EOF>	Optional; see below.

The format of *value* will be given by the *listitemtype*, if available. If the <TYPE> parameter is missing, treat the *listitemval* as a string. The *listitemtype* is as in $CONFIG$<LIST> (see page 149), but, since nested lists are not supported, it cannot have the value LIST.

The *nnn* argument may be less than *rrr* in the original request, if there were not that many entries, or if not all of them could be fitted into the reply packet. If this is the case, you can repeat the command, starting with *newfff* = *fff* + *nnn*, as many times as you need in order to get the full list.

If the last entry in the packet is also the last entry of the full list of items available for that category, the last entry will be followed by <EOF>.

It is not an error to specify a *fff* parameter that is greater than the total number of items available for the category. If this happens, *nnn* will be zero, there will be no <AT> entries, and there will be an <EOF> to indicate that there are no further entries to be had.

Read the value of an item

Request

Command:	$CONFIG$
Parameters:	<READ>*category*
	<ITEM>*item*

The *category* must be exactly as given in the reply to $CONFIG$<CATEGORIES>, and the *item* as given in $CONFIG$<LIST>*category*.

Reply

Command:	ACK
Parameters:	<OK>
	<CONFIG>
	<READ>*category*
	<ITEM>*item*
	<HAS>*value*

The format of *value* will depend upon that item's *itemtype*, as supplied by the reply to $CONFIG$<LIST>*category*.

If the *itemtype* was LIST, then the *value* will be exactly as given in one of the values from the reply to $CONFIG$<READ>*category*<LIST>*item*, and represents the currently-selected item in that list.

Set the value of an item

Request

Command:	$CONFIG$
Parameters:	<SET>*category*
	<ITEM>*item*
	<HAS>*value*

The *category* must be exactly as given in the reply to $CONFIG$<CATEGORIES>, and the *item* as given in $CONFIG$<LIST>*category*. That item must not have the *itemtype* of RO_STRING.

The format of *value* will depend upon that item's *itemtype*, as supplied by the reply to $CONFIG$<LIST>*category*.

If the *itemtype* was LIST, then the *value* must be exactly as given in one of the values from the reply to $CONFIG$<READ>*category*<LIST>*item*.

Reply

Command:	ACK	
Parameters:	<OK>	
	<CONFIG>	
	<SET>*category*	
	<ITEM>*item*	
	<REBOOT>*yesno*	*yesno* is either YES or NO.

If the *yesno* argument has the value YES, the server must be rebooted before any change has effect.

Querying and setting the power mode

[Introduced in Protocol version 1.02]

The server currently has two stable power modes (RUN and STANDBY) as well as two transient modes (RESTART and SHUTDOWN). In the STANDBY mode most of the normal (RUN mode) functionality of the server is unavailable. For example, the databases cannot be browsed, media can not be played or recorded, most updates will be discontinued and most status queries will fail. The server transitions from STANDBY mode to RUN mode via the RESTART mode; the reverse transition is via the SHUTDOWN mode.

Querying the power mode

[Request (and response) introduced in Protocol version 1.02]

This request must be sent to `server`.

Request

Command:	$STATUS$
Parameters:	<POWER>
	<MODE>

Reply

Command:	ACK
Parameters:	<OK>
	<POWER>
	<MODE>*power-mode*

power-mode may currently be `RUN’, `STANDBY’, `RESTART’ or `SHUTDOWN’. There is also an implicit `OFF’ mode in which the server is fully powered down and simply will not respond to requests including this one. Future versions of the protocol may provide support for a server to advise controllers that it is about to enter this mode in a software-controlled manner.

Changing the power mode

[Request (and response) introduced in Protocol version 1.02]

This request must be sent to `server`.

Request

Command:	$SYSTEMS$
Parameters:	<POWER>
	<MODE>*stable-power-mode*

Reply

Command:		ACK	
Parameters:			
	Then:		
		<OK>	The request has succeeded
	or:		
		<ERROR>	
		<MESSAGE>0eDevice busy	The server was in a transient power mode

stable-power-mode must currently be either \`STANDBY' or \`RUN'. A successful request to take the server from STANDBY to RUN will cause the server to pass through the transient \`RESTART' mode; the reverse transition is via the \`SHUTDOWN' mode. The error case occurs if the server is in one of these transient modes, i.e. it is already processing an earlier such request or changing power mode for some other reason.

When the server enters the RUN or STANDBY mode it will broadcast (see **Broadcasts and alerts** on page 38) this information as follows :-

Command:	$BROADCAST$
Paramters:	<MESSAGE>*Msg*
	<TYPE>RECONFIG
	<DETAIL>*Detail*

If it enters the RUN mode, *Msg* will be \`Server start' and *Detail* will be \`SERVERSTART'.
If it enters the STANDBY mode, *Msg* will be \`Server stop' and *Detail* will be \`SERVERSTOP'.

Alternative requests for controllers with small input buffers

[Introduced in Protocol version 1.02]

Standard XiVA-Link protocol requests result in a reply packet which is limited in length to a maximum of 1024 bytes. This packet size is sufficient to allow, for example, multiple textual attributes of a track (such as its title and artist and the name of the album to which it belongs) to be returned in the reply to a single request. Some controllers, however, have more limited input buffers. To enable such controllers to use the XiVA-Link protocol, alternatives to a subset of the standard XiVA-Link requests have been introduced. Each original individual request, which returns multiple string arguments and possibly other values, is replaced with a set of requests, each of which returns one of the original values.

The reply packets to these new requests are limited to a configurable maximum length, which defaults to the standard 1024 bytes but may be reduced to any value down to 66 bytes (see **Configuring communication** on page 32). Where the single piece of information returned is a number it is guaranteed not to be truncated. If it is a string it may be truncated to ensure that the return packet does not exceed the requested size limit. Currently no indication is given to the caller as to whether truncation has occurred.

These requests should in general only be used in controllers which cannot cope with a 1024-byte reply packet, since multiple such requests are required to build up a set of data that is normally returned using a single standard request.

The name of a track

[Request (and response) introduced in Protocol version 1.02]

This allows a controller to request the title of a track specified by ID.

This request is one of the alternatives to the standard $SEARCH$<TRACK><ID>*track-id* request (see **Track details** on page 86) and must be sent to `server`.

Request

Command:	$SEARCH$	
Parameters:	<TRACK>	
	<ID>*track-id*	Note that *track-id* is not optional.
	<NAME>	

Reply

Command:	ACK	
Parameters:	<OK>	
	<NAME>*name*	*name* may be truncated

The artist of a track

[Request (and response) introduced in Protocol version 1.02]

This allows a controller to request the name of the artist of a track specified by ID.

This request is one of the alternatives to the standard $SEARCH$<TRACK><ID>*track-id* request (see **Track details** on page 86) and must be sent to `server`.

Request

Command:	$SEARCH$	
Parameters:	<TRACK>	
	<ID>*track-id*	Note that *track-id* is not optional.
	<ARTIST>	

Reply

Command:	ACK	
Parameters:	<OK>	
	<ARTIST >*artist*	*artist* may be truncated

The compression format of a track

[Request (and response) introduced in Protocol version 1.02]

This allows a controller to find out the format in which a track is encoded.

This request is one of the alternatives to the standard $SEARCH$<TRACK><ID>*track-id* request (see **Track details** on page 86) and must be sent to `server`.

Request

Command:	$SEARCH$	
Parameters:	<TRACK>	
	<ID>*track-id*	Note that *track-id* is not optional.
	<COMPR>	

Reply

Command:	ACK
Parameters:	<OK>
	<COMPR>*compression-id*

The length of a track

[Request (and response) introduced in Protocol version 1.02]

This allows a controller to request the length of a track specified by ID.

This request is one of the alternatives to the standard $SEARCH$<TRACK><ID>*track-id* request (see **Track details** on page 86) and must be sent to `server`.

Request

Command:	$SEARCH$	
Parameters:	<TRACK>	
	<ID>*track-id*	Note that *track-id* is not optional.
	<LEN>	

Reply

Command:	ACK
Parameters:	<OK>
	<LEN>*hhhh:mm:ss*

The media id of a track

[Request (and response) introduced in Protocol version 1.02]

This allows a controller to request the ID of the media to which a track (specified by ID) belongs.

This request is one of the alternatives to the standard $SEARCH$<TRACK><ID>*track-id* request (see **Track details** on page 86) and must be sent to `server`.

Request

Command:	$SEARCH$
Parameters:	<TRACK>
	<ID>*track-id*
	<MEDIA>
	<ID>

Reply

Command:	ACK
Parameters:	<OK>
	<MEDIA>
	<ID>*media_id*

Media name

[Request (and response) introduced in Protocol version 1.02]

This allows a controller to request the title of the media specified by an ID.

This request is one of the alternatives to the standard $SEARCH$<MEDIA><ID>*media-id* request (see **Basic media details (disc etc.)** on page 88) and must be sent to `server`.

Request

Command:	$SEARCH$	
Parameters:	<MEDIA>	
	<ID>*media-id*	
	<NAME>	

Reply

Command:	ACK	
Parameters:	<OK>	
	<NAME>*title*	*May be truncated*

Media artist

[Request (and response) introduced in Protocol version 1.02]

This allows a controller to request the artist name for the media specified by an ID.

This request is one of the alternatives to the standard $SEARCH$<MEDIA><ID>*media-id* request (see **Basic media details (disc etc.)** on page 88) and must be sent to `server`.

Request

Command:	$SEARCH$	
Parameters:	<MEDIA>	
	<ID>*media-id*	
	<ARTIST>	

Reply

Command:	ACK	
Parameters:	<OK>	
	<ARTIST>artist-*name*	*May be truncated*

Media genre

[Request (and response) introduced in Protocol version 1.02]

This allows a controller to request the genre of the media specified by an ID.

This request is one of the alternatives to the standard $SEARCH$<MEDIA><ID>*media-id* request (see **Basic media details (disc etc.)** on page 88) and must be sent to `server`.

Request

Command:	$SEARCH$
Parameters:	<MEDIA>
	<ID>*media-id*
	<GENRE>

Reply

Command:	ACK
Parameters:	<OK>
	<GENRE>*genre*

Media track count

[Request (and response) introduced in Protocol version 1.02]

This allows a controller to request the number of tracks in the media specified by an ID.

This request is one of the alternatives to the standard $SEARCH$<MEDIA><ID>*media-id* request (see **Basic media details (disc etc.)** on page 88) and must be sent to `server`.

Request

Command:	$SEARCH$
Parameters:	<MEDIA>
	<ID>*media-id*
	<TOTAL>

Reply

Command:	ACK
Parameters:	<OK>
	<TOTAL>*number*

Media length

[Request (and response) introduced in Protocol version 1.02]

This allows a controller to request the length of the media specified by an ID.

This request is one of the alternatives to the standard $SEARCH$<MEDIA><ID>*media-id* request (see **Basic media details (disc etc.)** on page 88) and must be sent to `server`.

Request

Command:	$SEARCH$
Parameters:	<MEDIA>
	<ID>*media-id*
	<LEN>

Reply

Command:	ACK
Parameters:	<OK>
	<LEN>*hhhh:mm:ss*

Media type

[Request (and response) introduced in Protocol version 1.02]

This allows a controller to request the type of the media specified by an ID.

This request is one of the alternatives to the standard $SEARCH$<MEDIA><ID>*media-id* request (see **Basic media details (disc etc.)** on page 88) and must be sent to `server`.

Request

Command:	$SEARCH$	
Parameters:	<MEDIA>	
	<ID>*media-id*	
	<TYPE>	

Reply

Command:	ACK	
Parameters:	<OK>	
	<TYPE>*media-type*	*media-type* is AUDIO, VIDEO or MIXED.

Media source

[Request (and response) introduced in Protocol version 1.02]

This allows a controller to request the source of the media specified by an ID.

This request is one of the alternatives to the standard $SEARCH$<MEDIA><ID>*media-id* request (see **Basic media details (disc etc.)** on page 88) and must be sent to `server`.

Request

Command:	$SEARCH$
Parameters:	<MEDIA>
	<ID>*media-id*
	<SOURCE>

Reply

Command:	ACK
Parameters:	<OK>
	<TYPE>*media-source*

media-source depends upon *media-type* (see **Media type** on page 170) . If *media-type* is AUDIO, *media-source* is CD, DVD, LP, OTHER or UNKNOWN. If *media-type* is VIDEO, *media-source* is DVD, VTR or UNKNOWN.

ID of track within media

[Request (and response) introduced in Protocol version 1.02]

This allows a controller to find out the track id of a media item track.

This request is an alternative to the standard $SEARCH$<MEDIA><ID>*media-id*<TRACK> request (see **Media track details** on page 89). Instead of using one request to obtain the list of tracks, with multiple attributes per track included in the reply, this alternative request must be sent once for each track in the media to obtain the ID of that track. Additional alternative requests based on $SEARCH$<TRACK><ID>*id* must then be made to obtain further track attributes. This request must be sent to `server`.

Request

Command:	$SEARCH$
Parameters:	<MEDIA>
	<ID>*media-id*
	<TRACK>
	<NUM>*track-number*

Reply

Command:	ACK
Parameters:	<OK>
	<TRACK>
	<ID>*track_id*

The name of a playlist

[Request (and response) introduced in Protocol version 1.02]

This allows a controller to request the title of a playlist specified by ID.

This request is one of the alternatives to the standard $SEARCH$<PLAYLIST><ID>*playlist-id* request (see **Playlist details** on page 90) and must be sent to `server`.

Request

Command:	$SEARCH$	
Parameters:	<PLAYLIST>	
	<ID>*playlist-id*	
	<NAME>	

Reply

Command:	ACK	
Parameters:	<OK>	
	<NAME>*name*	*May be truncated*

The number of entries in a playlist

[Request (and response) introduced in Protocol version 1.02]

This allows a controller to find out how many entries a playlist contains.

This request is one of the alternatives to the standard $SEARCH$<PLAYLIST><ID>*playlist-id* request (see **Playlist details** on page 90) and must be sent to `server`.

Request

Command:	$SEARCH$
Parameters:	<PLAYLIST>
	<ID>*playlist-id*
	<TOTAL>

Reply

Command:	ACK
Parameters:	<OK>
	<TOTAL>*number*

The length of a playlist

[Request (and response) introduced in Protocol version 1.02]

This allows a controller to request the length of a playlist specified by ID.

This request is one of the alternatives to the standard $SEARCH$<PLAYLIST><ID>*playlist-id* request (see **Playlist details** on page 90) and must be sent to `server`.

Request

Command:	$SEARCH$
Parameters:	<PLAYLIST>
	<ID>*playlist-id*
	<LEN>

Reply

Command:	ACK
Parameters:	<OK>
	<LEN>*hhhh:mmm:ss*

The type of a playlist

[Request (and response) introduced in Protocol version 1.02]

This allows a controller to request the type of a playlist specified by ID.

This request is one of the alternatives to the standard $SEARCH$<PLAYLIST><ID>*playlist-id* request (see **Playlist details** on page 90) and must be sent to `server`.

Request

Command:	$SEARCH$	
Parameters:	<PLAYLIST>	
	<ID>*playlist-id*	
	<TYPE>	

Reply

Command:	ACK	
Parameters:	<OK>	
	<TYPE>*type*	*type* may be SPLIST or DPLIST

The track id of a playlist entry

[Request (and response) introduced in Protocol version 1.02]

This allows a controller to request the ID of the track at a specified position within a playlist.

This request is an alternative to the $SEARCH$<PLAYLIST><ID>*playlist-id*<TRACK> request (see **Playlist track details** on page 91). Instead of using one request to obtain the list of tracks, with multiple attributes per track included in the reply, this alternative request must be sent once for each track in the playlist to obtain the ID of that track. Additional alternative requests based on $SEARCH$<TRACK><ID>*id* must then be made to obtain further track attributes. This request must be sent to `server`.

Request

Command:	$SEARCH$	
Parameters:	<PLAYLIST>	
	<ID>*playlist-id*	
	<TRACK>	
	<NUM>*track-number*	*track-number* is the track position (starting at 1)

Reply

Command:	ACK
Parameters:	<OK>
	<TRACK>
	<ID>*track_id*

The name of the current track

[Request (and response) introduced in Protocol version 1.02]

This allows a controller to request the title of the currently loaded track.

This request is one of the alternatives to the standard $STATUS$<TRACK> request (see **Track status** on page 58) and must be sent to a playout zone. If no item has been selected for playout, then all of these requests (like the standard request) will result in the error, ‘ 01No media cued to play’.

Request

Command:	$STATUS$	
Parameters:	<TRACK>	
	<NAME>	

Reply

Command:	ACK	
Parameters:	<OK>	
	<NAME>*name*	*May be truncated*

The artist of the current track

[Request (and response) introduced in Protocol version 1.02]

This allows a controller to request the name of the artist for the currently loaded track.

This request is one of the alternatives to the standard $STATUS$<TRACK> request (see **Track status** on page 58) and must be sent to a playout zone. If no item has been selected for playout, then all of these requests (like the standard request) will result in the error, ‘ 01No media cued to play’.

Request

Command:	$STATUS$	
Parameters:	<TRACK>	
	<ARTIST>	

Reply

Command:	ACK	
Parameters:	<OK>	
	<ARTIST>*artist*	*May be truncated*

The position of the current track within its enclosing item

[Request (and response) introduced in Protocol version 1.02]

This allows a controller to request the position (starting from 1) of the currently loaded track within the enclosing item currently selected. Note that the tracks within that item may have been shuffled if random mode is switched on.

This request is one of the alternatives to the standard $STATUS$<TRACK> request (see **Track status** on page 58) and must be sent to a playout zone. If no item has been selected for playout, then all of these requests (like the standard request) will result in the error, ‘ 01No media cued to play’.

Request

Command:	$STATUS$
Parameters:	<TRACK>
	<NUM>

Reply

Command:	ACK
Parameters:	<OK>
	<NUM>*number*

The original position of the current track within its enclosing item

[Request (and response) introduced in Protocol version 1.02]

This allows a controller to request the original position (starting from 1) of the currently loaded track within the (unshuffled) enclosing item currently selected.

This request is one of the alternatives to the standard $STATUS$<TRACK> request (see **Track status** on page 58) and must be sent to a playout zone. If no item has been selected for playout, then all of these requests (like the standard request) will result in the error, ‘ 01No media cued to play’.

Request

Command:	$STATUS$
Parameters:	<TRACK>
	<ORIG>

Reply

Command:	ACK
Parameters:	<OK>
	<ORIG> *original-number*

The length of the current track

[Request (and response) introduced in Protocol version 1.02]

This allows a controller to request the length of the currently loaded track.

This request is one of the alternatives to the standard $STATUS$<TRACK> request (see **Track status** on page 58) and must be sent to a playout zone. If no item has been selected for playout, then all of these requests (like the standard request) will result in the error, ‘ 01No media cued to play’.

Request

Command:	$STATUS$
Parameters:	<TRACK>
	<LEN>

Reply

Command:	ACK
Parameters:	<OK>
	<LEN>*hhhh:mm:ss*

The track id of the current track

[Request (and response) introduced in Protocol version 1.02]

This allows a controller to request the ID of the currently loaded track.

This request is one of the alternatives to the standard $STATUS$<TRACK> request (see **Track status** on page 58) and must be sent to a playout zone. If no item has been selected for playout, then all of these requests (like the standard request) will result in the error, ‘ 01No media cued to play’.

Request

Command:	$STATUS$
Parameters:	<TRACK>
	<ID>

Reply

Command:	ACK
Parameters:	<OK>
	<ID>*track_id*

The media ID of a drive’s media

[Request (and response) introduced in Protocol version 1.02]

This request is one of the alternatives to the standard $STATUS$<*DISCTYPE*>*discnum*<MEDIA> request (see **Describing a drive’s media** on page 134) and must be sent to `server`.

Request

Command:	$STATUS$	
Parameters:	<*DISCTYPE*>*discnum*	*DISCTYPE* may currently only be CD
	<MEDIA>	
	<ID>	

Reply

Command:	ACK	
Parameters:	<OK>	
	<*DISCTYPE*>*discnum*	
then:		
	<NONE>	Meaning no disc is loaded
or:		
	<ID>*media-id*	

The title of a drive’s media

[Request (and response) introduced in Protocol version 1.02]

This request is one of the alternatives to the standard $STATUS$<*DISCTYPE*>*discnum*<MEDIA> request (see **Describing a drive’s media** on page 134) and must be sent to `server`.

Request

Command:	$STATUS$	
Parameters:	<*DISCTYPE*>*discnum*	*DISCTYPE* may currently only be CD
	<MEDIA>	
	<NAME>	

Reply

Command:	ACK	
Parameters:	<OK>	
	<*DISCTYPE*>*discnum*	
then:		
	<NONE>	Meaning no disc is loaded
or:		
	<NAME>*media-name*	*media-name* is a string

If an online database lookup (see $ONLINE$<UPDATE><TRACKDB>, p 128) has not been performed or the media was not found in the online database used and the media details have not been edited, then *media-name* is likely to be a temporary name assigned by the server, e.g. `Album 4’.

The artist for a drive’s media

[Request (and response) introduced in Protocol version 1.02]

This request is one of the alternatives to the standard $STATUS$<*DISCTYPE*>*discnum*<MEDIA> request (see **Describing a drive’s media** on page 134) and must be sent to `server`.

Request

Command:	$STATUS$	
Parameters:	<*DISCTYPE*>*discnum*	*DISCTYPE* may currently only be CD
	<MEDIA>	
	<ARTIST>	

Reply

Command:	ACK	
Parameters:	<OK>	
	<*DISCTYPE*>*discnum*	
then:		
	<NONE>	Meaning no disc is loaded
or:		
	<ARTIST>*media-artist*	*media-artist* is a string

If an online database lookup (see $ONLINE$<UPDATE><TRACKDB>, p 128) has not been performed or the media was not found in the online database used and the media details have not been edited, then *media-artist* is likely to be a temporary name assigned by the server, e.g. `Artist 4’.

The genre of a drive’s media

[Request (and response) introduced in Protocol version 1.02]

This request is one of the alternatives to the standard $STATUS$<*DISCTYPE*>*discnum*<MEDIA> request (see **Describing a drive’s media** on page 134) and must be sent to `server`.

Request

Command:	$STATUS$	
Parameters:	<*DISCTYPE*>*discnum*	*DISCTYPE* may currently only be CD
	<MEDIA>	
	<GENRE>	

Reply

Command:	ACK	
Parameters:	<OK>	
	<*DISCTYPE*>*discnum*	
then:		
	<NONE>	Meaning no disc is loaded
or:		
	<GENRE>*genre*	*genre* is a string

If an online database lookup (see $ONLINE$<UPDATE><TRACKDB>, p 128) has not been performed or the media was not found in the online database used and the media details have not been edited, then *genre* is likely to be `Unknown’.

Whether a drive’s media has been looked up

[Request (and response) introduced in Protocol version 1.02]

This request is one of the alternatives to the standard $STATUS$<*DISCTYPE*>*discnum*<MEDIA> request (see **Describing a drive’s media** on page 134) and must be sent to `server`.

Request

Command:	$STATUS$	
Parameters:	<*DISCTYPE*>*discnum*	*DISCTYPE* may currently only be CD
	<MEDIA>	
	<LOOKUP>	

Reply

Command:	ACK	
Parameters:	<OK>	
	<*DISCTYPE*>*discnum*	
then:		
	<NONE>	Meaning no disc is loaded
or:		
	<LOOKUP>*yesno*	*yesno* is either YES or NO

yesno is only YES if the media has been successfully looked up in an online database.

The media number of a drive’s media

[Request (and response) introduced in Protocol version 1.02]

This request is one of the alternatives to the standard $STATUS$<*DISCTYPE*>*discnum*<MEDIA> request (see **Describing a drive’s media** on page 134) and must be sent to `server`.

Request

Command:	$STATUS$	
Parameters:	<*DISCTYPE*>*discnum*	*DISCTYPE* may currently only be CD
	<MEDIA>	
	<NUM>	

Reply

Command:	ACK	
Parameters:	<OK>	
	<*DISCTYPE*>*discnum*	
then:		
	<NONE>	Meaning no disc is loaded
or:		
	<NUM>*media-num*	*media-num* is the media’s media number (c.f. `Select media by media number’, p 50).

The title of a track in a drive’s media

[Request (and response) introduced in Protocol version 1.02]

This request is one of the alternatives to the standard $STATUS$<*DISCTYPE*>*discnum*<TRACK> request (see **Describing a drive’s media’s tracks** on page 135) and must be sent to `server`. It allows the name of an individual track in the media to be retrieved.

Request

Command:	$STATUS$	
Parameters:	<*DISCTYPE*>*discnum*	*DISCTYPE* may currently only be CD
	<TRACK>	
	<AT>*track-pos*	*track-pos* is the track number (starting from 1).
	<NAME>	

Reply

Command:	ACK	
Parameters:	<OK>	
	<*DISCTYPE*>*discnum*	
	<TRACK>	
then:		
	<NONE>	Meaning no disc is loaded
or:		
	<AT>*track-pos*	
	<NAME>*track-name*	*track-name* is a string
or:		
	<AT>*track-pos*	
	<NAME>	
	<NONE>	Meaning there is no track number *track-pos*

If an online database lookup (see $ONLINE$<UPDATE><TRACKDB>, p 128) has not been performed or the media was not found in the online database used and the media details have not been edited, then *track-name* is likely to be a temporary name assigned by the server, e.g. `Track 7’.

The length of a track in a drive’s media

[Request (and response) introduced in Protocol version 1.02]

This request is one of the alternatives to the standard $STATUS$<*DISCTYPE*>*discnum*<TRACK> request (see **Describing a drive’s media’s tracks** on page 135) and must be sent to `server`. It allows the length of an individual track in the media to be retrieved.

Request

Command:	$STATUS$	
Parameters:	<*DISCTYPE*>*discnum*	*DISCTYPE* may currently only be CD
	<TRACK>	
	<AT>*track-pos*	*track-pos* is the track number (starting from 1).
	<LEN>	

Reply

Command:	ACK	
Parameters:	<OK>	
	<*DISCTYPE*>*discnum*	
	<TRACK>	
then:		
	<NONE>	Meaning no disc is loaded
or:		
	<AT>*track-pos*	
	<LEN>*hhhh:mm:ss*	
or:		
	<AT>*track-pos*	
	<LEN>	
	<NONE>	Meaning there is no track number *track-pos*

Retrieving a single field from a single database cache item

[Request (and response) introduced in Protocol version 1.02]

This request is an alternative to the standard $SEARCH$<CACHE><LIST><MARKER>*marker*<FROM>*fff*<FOR>*rrr* request (see **Listing a cache** on page 96) and must be sent to `server`. It allows an individual field of the item at a particular position in the specified cache to be retrieved.

Request

Command:	$SEARCH$	
Parameters:	<CACHE>	
	<LIST>	
	<MARKER>*marker*	The value returned when the cache was opened
	<AT>*item-pos*	Position (starting at 1) within the cache
	<FIELD>	See below

Reply

Command: ACK
Parameters:

Either:

<ERROR>
<MESSAGE>16Cache marker no longer valid

Or:

<OK>
<AT>*item-pos*
<FIELD>value

Or if item-pos is beyond the last item in the cache:

<OK>
<AT>*item-pos*
<EOF>

FIELD may be any field (e.g. NAME, MEDIA, ID, ARTIST, GENRE) which is valid for the type of cache specified by *marker*. See **Table xi: Cache elements** on page 93.

If the error occurs, the marker should be considered invalid, and closed. The controller does not need to (and should not) close it.

Appendix A: Differences between 1.00 and 1.01

These are the differences between Protocol versions 1.00 and 1.01:

- The *message-sequence-char* becomes optional (with caveats that we do not recommend making use of this feature);
- The *checksum* can be zero, two- or four-digit hex, with differing meanings for each. We do not recommend using zero-digit checksums.

Appendix B: Differences between 1.01 and 1.02

These are the differences between Protocol versions 1.01 and 1.02 :-

1. There is a new request to obtain playlist track details (**Playlist track details**, page 91);

2. A <DONE> parameter will be appended to a play-state update (**Play-state Update**, page 67) and to the reply to a $STATUS$<MODE> request (**Operating mode**, page 55) if playout has reached the end of the selected item;

3. There is a new update to report configuration changes (**Configuration update**, page 82);

4. There is a new command to report external modification of the database (**Reporting an external modification of the database**, page 123);

5. There is a new request to obtain the free space remaining on the server in terms of playing time (**Free space on the server in terms of playback time**, page 63);

6. An reply to an $ONLINE$<WANT>{<ON>|<OFF>} request reporting that the requested state has not been attained may now have <TYPE>*IXXdetail* appended giving an indication of why the transition failed (**Requesting on/offline state**, page 125).

7. ONLINE updates are now sent to report each stage of each connection attempt made when responding to a request to go online and thus can have a number of new parameters appended to them (**On/offline update**, page 73).

8. The reply to the request to list tracks on a disc (in one of the server's drives) now has <EOF> appended if the tracks listed end with the last track on the disc (**Describing a drive's media's tracks**, page 135). This brings it into line with other list replies.

9. The media number for the disc concerned has been appended to RECORD **updates (Record state update**, page 69) and to the reply to $STATUS$<*DISCTYPE*>*discnum*<MEDIA> (**Describing a drive's media**, page 134).

10. A new variant of the media availability update ($UPDATE$<*DISCTYPE*>*discnum*<NONE>) is sent when media becomes unavailable (**Media availability update**, page 72).

11. The state of a disc drive tray may now be toggled in addition to explicitly opening or closing it (**Opening or closing the drive door (ejecting or loading a disc)**, page 136).

12. Options have been added to the $SELECT$<ID>*id* request to allow playout to be started (or restarted) at a particular track within a playlist or media, all within one operation (**Select media, playlists** or individual tracks by ID, page 48).

13. The operation of looking up media in an online database can now be cancelled using $ONLINE$<UPDATE><TRACKDB><ABORT> (**Aborting a TRACKDB update**, page 130).

14. Empty playlists can now be created using $SEARCH$<COMMIT><NAME>*name* (**Committing a playlist without searches**, page 120).

15. A number of new configuration items have been added (**Configuring the server**, page 149). Items under *DefaultEncodings* allow the default encoding for a particular track type to be queried or set. This is the encoding which will be used in response to a $RECORD$ request with no <COMPR> parameter. (See the note added to **Recording a disc's contents**, page

138). The *PortUsers* and *PortUserSettings* categories allow specification and configuration of the applications listening on particular ports. Which controllers manage which playout zones is now configurable under the *Controllers* category, with the number of channels which supported controllers are capable of managing being available under *NumCtrlrChans*.

16. Some new warning and error codes have been added (**Error and warning codes**, page 27)

17. A set of alternative XiVA-Link requests have been introduced for use in controllers with input buffers smaller than the standard maximum size (1024 bytes) for XiVA-Link packets (**Alternative requests for controllers with small input buffers**, page 159)

18. New requests have been added to provide access to the power mode of the server **(Querying and setting the power mode**, page 157)(**Power mode update** on page 83).

19. The response of the server to $PING$<RESET> has been enhanced so that this request can now be used for proper session management (**Starting a new session**, page 31).

Appendix C: Fixed errata

Errors in Revisions up to 33 fixed in Revision 34 :-

1. In Revision 33 (**Play-state Update**), the specified order of the parameters in a <MODE> update was incorrect. The <POS> and <MSECS> parameters were swapped with the <NUM> and <ORIG> parameters; the former precede rather than follow the latter.

2. In Revision 33 (**Select by track Reply**), the list of parameters in the ACK to a $SELECT$<TRACK> request was specified as including a final <COMPR> parameter to indicate the compression of the track. This was a transient development feature and was never released.

3. In Revision 33 (**Requesting on/offline state**), it was stated that an ACK<OK> response indicated only that the request had been received. It actually means that the server successfully entered (or was already in) the requested online state.

4. In Revision 33 (**Requesting a TRACKDB update**), it was stated that an ACK<OK> response indicated only that the operation to look up an item in an online database had begun. In fact, it means that the operation has succeeded; additional parameters specify the number of items that were looked up and how many of these were not found. If the lookup fails or is aborted, an additional parameter specifies the number of items that remained to be looked up at the time.

5. In Revision 33 (**Recording a disc's contents**), it was stated that an ACK<OK> response indicated only that the recording operation had begun. Such a response is not actually sent until the record operation ends (for whatever reason) and there are additional parameters to specify the number of tracks which were requested to be recorded, the number which failed to be recorded and the number which remained to be attempted when recording ended.

6. In Revision 33 (**Recording a disc's contents**), the optional <COMPR> parameter in a $RECORD$ request was not described. This may be used to specify the type of encoding used in the recording.

Errors in Revisions up to 36 fixed in Revision 37

1. In Revision 36 and earlier (**Committing a playlist without searches** and **Committing category searches**), the optional <REPLACE> parameter in $SEARCH$<COMMIT> requests used to create playlists was not documented. This parameter allows an existing playlist to be overwritten by a new one of the same name and has actually been available since version 1.01 of the Protocol.

2. In Revision 36 and earlier (**Commands – general principles** and **Error and warning codes**), there were several cases in which the <MESSAGE> parameter was omitted from descriptions of ACK<ERROR><MESSAGE>XXmessage or ACK<WARNING><MESSAGE>XXmessage.

Error in Revision 37 fixed in Revision 38

1. In Revision 37 (**Querying the power mode**), the <MODE> parameter was omitted from the description of the $STATUS$<POWER><MODE> request.

Errors in Revisions up to 38 fixed in Revision 39

1. In Revision 38 and earlier (**Play status**), the description of the reply to $STATUS$<PLAY> when the selected item is a static playlist incorrectly included an <ARTIST> parameter. Unlike tracks and media, no artist is associated with a playlist.

2. In Revision 38 and earlier (**TRACKDB change update**), the description of the $STATUS$<UPDATE><TRACKDB>*onoff* request omitted the *onoff* (ON or OFF) argument.

3. In Revision 38 and earlier (**PLAYLISTDB change update**), the descriptions of the $STATUS$<UPDATE><PLAYLISTDB>*onoff* request and the $UPDATE$<PLAYLISTDB> update were incorrect.

Appendix D: Publication approvals

Unless the Managerial sign-off post-dates the most recent change in the revision history, this document is not approved for external release.

The Manager signing off must ensure that any changes since the previously released version have been checked for technical and legal issues.

Approved for	By	When
Technical sign-off	Andrew Fyfe	2001/10/22
Legal sign-off	Chris Janes	2001/10/22
Managerial sign-off for release	Andrew Fyfe	2001/10/22

Table xviii: Publication approvals

Appendix E: Revision history

Revision	Date	Who	Changes
039	2001-10-11	Richard Shaw	• Fixed some errors as detailed in Appendix C: Fixed errata.
038	2001-09-25	Richard Shaw	• Added specification of maximum cache marker length. • Corrected an error in the power mode query syntax as detailed in Appendix C: Fixed errata.
037	2001-08-06	Richard Shaw	• Updated to incorporate further revisions of the 1.02 version definition (more small-reply-buffer requests, session management changes, standby mode control). • Updated the comments on $STATUS$<*DISCTYPE*>*discnum*<MEDIA>. • Corrected further errata as detailed in Appendix C: Fixed errata.
036	2001-07-20	Richard Shaw	• Updated to incorporate revisions of the 1.02 version definition. • Added the current definition of `session' to the Glossary.
035	2001-07-18	Jon Green	• No semantic changes; just converted to Imerge Ltd.'s new house style for published documents.
034	2001-07-10	Richard Shaw	• Defined 1.02, with changes from 1.01 as detailed in Appendix B • Fixed some errata as detailed in Appendix C: Fixed errata. • Added comments about play-state. • Added note about what must be done when playout stops due to the end of the selected item being reached to allow a further $PLAY$ request to succeed.
033	2000-11-10	Jon Green	• Define 1.01, in which the only significant changes are that the *message-sequence-char* becomes optional, and the *checksum* contents can be simplified.
032	2000-11-06	Jon Green	• Protocol specification goes live! • Version negotiation removed indefinitely: $VERSION$<CURRENT>*M.mm*<WANT>*M.mm* deprecated • *Command-word* upper size limit increased from 9 to 10 • *Param-word* upper size limit increased from 11 to 12 • $RS232$ commands removed completely • (Clarification) Playout destination names will not always be Z01, Z02, and so on • Enhanced description of version numbering • Description (in **Commands – General Principles**) of what happens to packets with invalid destinations or invalid contents. • Full list of error messages and warnings, current

			to 1.00 release version. • $STOP$<AFTER> deprecated – it may be reintroduced into future versions • $PLAY$<RATIO> deprecated – it will probably be reintroduced into future versions • $PLAY$<FLAG> and $STATUS$<PLAY><FLAG> added. • <MSECS> parameter added to $PLAY$<SKIP> • $PLAY$<SKIP> warning reply format corrected • Track update $UPDATE$ format corrected. • Updates get their own chapter • Update types considerably expanded • DPLISTs (dynamic playlists) not supported at 1.00. • Fixed incorrect descriptions of $SEARCH$<*type*><ID> commands (*type* is TRACK, MEDIA or PLAYLIST) to describe more accurately how to use the no-*id* variants. • $SEARCH$<TRACK> reply format corrected • $SEARCH$<TRACK><ID><FULL> added • $SEARCH$<MEDIA> reply format corrected. • $SEARCH$<MEDIA><TRACK> and variants added. • $SEARCH$<DELETE><ENTRY> deprecated. • $SEARCH$<PLAYLIST> reply format corrected. • $SEARCH$<CACHE> commands added. • $ALTER$ commands added. • $DELETE$ commands added. • $CONFIG$ commands added.
031	2000-08-24	Jon Green	• "Future Directions" section integrated into main body of specification; • Extensions for new update types, disk information and database modification added • **WARNING**: this does not include all current commands. A future update will correct omissions.
030	2000-06-06	Jon Green	• $SELECT$<MEDIA> enhanced and clarified • New $SEARCH$<INFO> command • $SEARCH$<COUNT>: ***correction***: comparison using <LIKE> is case *sensitive* • $SEARCH$<COUNT>: <RECYCLE> parameter withdrawn • $SEARCH$<LIST>: more than one <CAT> now permitted • $SEARCH$<LIST>: reply format changed • $SEARCH$<DELETE><TAG><ALL> deprecated • $SEARCH$<RENAME> moved to 0.80 • New command: $SEARCH$<INFO><ID> • $SEARCH$<DELETE><TAG> can now take more than one tag at a time • $SEARCH$<DELETE><ENTRY> is temporarily withdrawn, pending a review • $SEARCH$ operations on <DPLIST>s have been

			postponed until 0.80 at least • $SEARCH$<COMMIT><TAG>: parameter <NAME> is now mandatory • $SEARCH$<COMMIT>: reply enhanced. • $SEARCH$<COMMIT>: clarification on which types of tags can be <COMMIT>ted to <DPLIST>s. • <SPLISTDB> database introduced. • Much revision of search examples. • All occurrences of <WARN> are changed to <WARNING>, correcting an historical mistake. The server has always generated <WARNING>s.
029	2000-04-13	Jon Green	• Fixed the background information in Appendix A (Future Directions) that wasn't customer-friendly.
028	2000-04-12	Jon Green	• Formal grammar fixed • ACK<RXD> no longer guarantees to repeat after 10s delay • $PLAY$<RATIO> and $STOP$<AFTER> not supported until at least 0.80. • Added "Future directions" appendix
027	2000-03-08	Jon Green	• Changed format of replies to $SEARCH$ variants <TRACK>, <MEDIA> and <PLAYLIST><ID>.
026	2000-01-25	Jon Green	• Now defining version 0.70; removed all temporary details (0.30, 0.50). • Checksum moved to end of packet. • Quoting mechanism changed to backslash from HTML-like. • 0.30 MEDIA operations made permanent. • 0.30-specific details removed. • Source-ID is no longer optional. • Minimum response time to ACK a command increased from 1s to 5s. • Note that timed updates always happen at 5-6s intervals regardless of requested frequency.
025 (0.25)	1999-11-02	Jon Green	• Updated heavily to reflect feedback from customers; • Changed 'DISC' to 'MEDIA' throughout; • Added revision history; • Added restricted subset 0.50 for CES. • Changed <PLAYLIST-ID> to <PLAYLIST> in $SEARCH$<COMMIT><ID>. • There are no parameters with white-space or hyphens in their names any more. Grammar updated to reflect this.
023 (0.23)	1999-10-05	Jon Green	• Improved formatting; included TOC
022 (0.22)	1999-10-04	Jon Green / Richard Caton	• Initial document created

Table xix: Revision history