Appendix D

GEMS Protocol

This Appendix describes the communications protocol used when a central count system is exchanging data with a GEMS database server. This specification is included here because it provides details of the data compression used by various Central Count System DLL functions.

This chapter describes protocol version 7. Version 8 is virtually identical, but the differences will be covered in the next release of this Guide.

Protocol extensions for handling write-in votes are tentative, are subject to review and change, and will be implemented in a future version of the protocol. They are *not* supported by versions 7 or 8.

This chapter makes reference to 17-inch ballots, which are not yet supported by other GES products and systems.

Protocol Overview

The GEMS protocol can be summarized as follows:

- Communications between the client and GEMS is done via a TCP/IP connection on port 3030.
- Ballots are processed in batches. A batch is opened with a 'Batch Start Card' and closed with an 'Ender Card'. Cards are not committed (and therefore not counted) until the 'Ender Card' is proceeded.
- While a batch is open the client must send GEMS a ping ("keep-alive") message every 10 seconds. GEMS will respond to each ping with a "pong". If GEMS does not receive a ping for 30 seconds, it assumes the client has "died", discards the current batch, and closes the port.
- When the client processes a new ballot, it requests the ballot oval mask from GEMS. Ballot masks are identified by the "front ID marks". This is the set of 32 timing marks along the bottom edge of the front of the card. The client does not

have to request the mask if it has buffered the same mask from an earlier ballot.

- The requested oval mask is transmitted back to the client.
- When the client has read the ovals on a ballot, the vote data is sent back to GEMS.
- GEMS replies with a message indicating acceptance or rejection of the vote data.
- If write-in votes are being tabulated, equivalent write-in specific messages for mask request, mask reply, data submission, and acceptance are exchanged with GEMS.

Most messages involve the use of compressed data to minimize network traffic, and to provide some measure of data security.

The format of the messages and their compressed data is described in the following sections.

Message Definitions

Listing 75. *Data types and message IDs*

```
Data Types:
  BYTE     1 byte (C++ UCHAR)
  WORD     2 byte LE unsigned int
  BYTE[n]  Array of n bytes
  String   NULL terminated string

Message IDs:
  MID_PING                 0
  MID_PONG                 1
  MID_MASK_REQUEST         2
  MID_MASK_REPLY           3
  MID_BALLOT_IMAGE         4
  MID_BALLOT_DESTINATION   5
  MID_PROTOCOL             6
  MID_WRITEIN_REQUEST      7
  MID_WRITEIN_REPLY        8
  MID_WRITEIN_IMAGE        9
  MID_WRITEIN_DESTINATION 10
```

Listing 76. *Message Structures*

```
Protocol Version:
  MID_PROTOCOL              BYTE
  Version                   WORD

Ballot Mask Request:
  MID_MASK_REQUEST          BYTE
  CardRotId                 WORD
```

```
  ElectionId               WORD

Ballot Mask Reply:
  MID_MASK_REPLY           BYTE
  CardLength               BYTE
  Found                    BYTE
  MaskSize                 WORD
  Mask or ErrorText        BYTE[MaskSize] if Found, else
                           BYTE[32]

Ballot Image:
  MID_BALLOT_IMAGE         BYTE
  ButtonStatus             BYTE
  CardLength               BYTE
  CardRotId                WORD
  ReportUnitId             WORD
  ElectionId               WORD
  ImageSize                WORD
  Image                    BYTE[ImageSize]

Ballot Destination:
  MID_BALLOT_DESTINATION   BYTE
  Destination              BYTE
  BufferLength             WORD
  Buffer                   STRING (Display, RejectReason or
                             Print message)

Ballot Write-in Mask Request:
  MID_WRITEIN_REQUEST      BYTE
  CardRotId                WORD
  ElectionId               WORD

Ballot Write-in Mask Reply:
  MID_WRITEIN_REPLY        BYTE
  CardLength               BYTE
  Found                    BYTE
  MaskSize                 WORD
  Mask or ErrorText        BYTE[MaskSize] if Found, or
                           BYTE[32]

Ballot Write-in Image
  MID_WRITEIN_IMAGE        BYTE
  ButtonStatus             BYTE
  CardLength               BYTE
  CardRotId                WORD
  ReportUnitId             WORD
  ElectionId               WORD
  DataSize                 WORD
  Data                     BYTE[DataSize]

Ballot Write-in Destination:
  MID_WRITEIN_DESTINATION  BYTE
  Destination              BYTE
  BufferLength             WORD
  Buffer                   STRING (Display, RejectReason or
                             Print message)
```

In the following descriptions, oval location (0, 0) refers to the top left usable oval position, *not* the top left timing mark. Thus oval (0, 0) is located at position (1, 1) in the timing mark matrix.

Field Descriptions

Version is the protocol version (20007 or 20008)

CardRotID is a number that uniquely describes the layout of voting "ovals" on the ballot. This number is used when retrieving the ballot's mask from the GEMS system.

ElectionID is the election type code (see page 10).

CardLength is the number of timing mark rows required on the ballot (41, 53, 65 or 69 for 11, 14, 17 and 18" ballots respectively).

Found indicates whether the mask for **CardRotID** was found (1) or not found (0) in the GEMS database.

MaskSize indicates the number of bytes of data that must be read from the **Mask** field.

ErrorText is an error message returned in an array of 32 bytes. **ButtonStatus** is always 0 when the GEMS protocol is used with AccuVote Central Count System.

ReportUnitID is either the batch or precinct number.

ImageSize indicates the number of bytes of data that must be read from the **Image** field.

Destination indicates whether ballot vote data was rejected (0) or accepted (1) by GEMS.

BufferLength indicates the number of characters that must be read from the **Buffer** field.

DataSize indicates the number of bytes of data that must be read from the **Data** field.

The messages associated with write-ins are identical to the equivalent messages for ovals, except for the contents and encoding of the compressed data fields.

Compression Algorithms

This section describes the algorithms used to compress data into the formats used by the GEMS protocol.

Oval Masks

Ballot oval masks are compressed using the following method:

Listing 77. *C++ declaration of oval mask data*

```
UCHAR Size              (number of bytes in remainder of message)
UCHAR Data[Size]        (the ENCODE_MASK_ITEMS for each column or row)

where ENCODE_MASK_ITEM is as follows:

UCHAR Type              (by row (1) or by column (0))
UCHAR Index             (zero-based column or row being defined)
UCHAR Length            (number of bytes in the mask data)
UCHAR Data[Length]      (bit mask where each bit refers to a column
                         or row within the row or column, where 1
                         indicates a used voting position and 0
                         indicates an unused voting position.  Any
                         voting position not defined in the mask is
                         unused.)
```

For example the data may be

41, 12,

0, 2, 2, 0x0f, 0xf0

0, 16, 4, 0x0f, 0x80, 0x03, 0xc0

This would describe a ballot that is 41 timing marks long and has voting positions in column 2 rows 4 to 11, and in column 4 rows 4 to 8 and rows 21 to 24.

Vote Data

Vote data (obtained from reading ovals) are compressed using the following method:

Listing 78. *C++ declarations for vote data*

```
UCHAR Size              (number of bytes in remainder of message)
UCHAR Data[Size]        (data bytes)

where data is the compressed vote data.  The data is two bits per
oval, in the same order as the ballot mask. A value of 0x0 indicates
a NO vote, 0x01 a YES vote, and 0x02 an undefined mark (one that
the system cannot be sure is voted nor unvoted).
```

Write-in Masks

As proposed, write-in mask data is compressed using the following method:

Listing 79. *C++ declaration of write-in mask data*

```
UCHAR Size              (number of bytes in remainder of message)
UCHAR Data[Size]        (the ENCODE_WRITEIN_MASK_ITEM for all write-in
                         voting positions)

where ENCODE_WRITEIN_MASK_ITEM comprises the data for all write-in
voting positions, concatenated together in a single data stream.
```

```
Each write-in voting position has the following bytes of encoded data:

UCHAR Index           (zero-based index into oval mask array)
UCHAR X1              (left side of write-in area, mm)
UCHAR Y1              (top side of write-in area, mm)
UCHAR X2              (right side of write-in area, mm)
UCHAR Y2              (bottom side of write-in area, mm)
```

X1 through **Y2** are dimensions describing the rectangle that encloses the write-in area provided for the voter. These dimensions are in millimeters, and are relative to the center of the oval. Note that all of these values must be positive numbers. **X1** and **X2** are always interpreted as being to the *right* of the adjacent oval. **Y1** is always *above* the center of the oval, and **Y2** is always *below* it.

Note that if the X1..Y2 values for a write-in voting position are the same as for the previous position, X1 can be returned as zero, and the remaining 3 values omitted.

For example, if the 13th, 28th and 41st voting positions are write-ins, with the 13th and 28th having the same dimensions, the data stream might be:

```
(12)(12)(3)(54)(5)(2)(27)(0)(40)(3)(60)(5)(2)
```

where decimal byte values are in parentheses.

Write-in Names

Candidate names for write-ins used by a voter are compressed using the following method:

Listing 80. *C++ declarations for write-in names data*

```
UCHAR DataSize        (number of bytes in remainder of message)
UCHAR Data[DataSize]  (data bytes)

where Data provides the voting position and candidate name for all of the
write-in voting positions actually used by the voter, concatenated into
a single data stream. Unused write-in voting positions are not included
in the data. Each write-in vote has the following bytes of encoded data:

UCHAR Index           (zero-based index into oval array for this vote)
UCHAR Chars           (number of characters in Name)
UCHAR Name[Chars]     (the write-in candidate's name)
```

For example, a vote for John Smith in the 13th voting position and for Jane Doe in the 28th voting position would result in the data stream:

```
(22)(12)(10)John Smith(27)(8)Jane Doe
```

where decimal byte values are in parentheses.