Developer’s Guide

AccuVote Central Count System
Image Processing DLL
Developer’s Guide

AccuVote Central Count System
Image Processing DLL

Peter G. Martin

© Copyright 2001 by Global Election Systems, Inc.
All rights reserved
Contents

1 Introduction, 1
   Summary of DLL Functions, 1
   Function Result Codes, 2
   Installation, 3

2 Data Definitions, 4
   tBallot Record, 4
   tGemsData Record, 6
   tGemsNames Record, 7
   tMarkInfo Record, 8
   tOvalRec Record, 11
   tScanInfo Record, 12
   tWriteIn Record, 14

3 Image Processing Function Calls, 16
   ReadBallotMarks Function, 16
   RepairBallot Function, 18
   ReadBallotOvals Function, 20
   DupBallot Function, 21
   ReadWriteIns Function, 23

4 Data Conversion Functions, 27
   GemsDateToYYMMDD Function, 27
   GemsMaskToOvals Function, 28
   GemsNamesToWriteins Function, 29
   GemsVotesToOvals Function, 30
   GemsWIMaskToWriteins Function, 31
   OvalsToGemsMask Function, 32
   OvalsToGemsVotes Function, 33
   WriteinsToGemsWIMask Function, 33
   YYMMDDToGemsDate function, 34

5 Miscellaneous Functions, 36
   GetBallotMask Function, 36
   GetDLLVersion Function, 38
   GetWriteInMask Function, 39
   ShowTiffFile Function, 40
6 Ballot Processing Techniques, 43
   Image Processing Environment, 43
   Image Processing Algorithms, 47
   Reading Timing Marks, 51
   Reading Votes in Ovals, 51
   Reading Write-In Votes, 52

APPENDIX
A Timing Mark Coding, 53
   Front Marks, 53
   Back Marks, 53
   Ballot Validation, 54
B GEMS Protocol, 55
   Protocol Overview, 55
   Message Definitions, 56
   Compression Algorithms, 58
C Complete Data Definitions, 61
   Data Declarations, 61
D Complete Function Calls, 67
E Calling DLLs From Visual Basic, 70
   Numeric Data Types, 70
   Strings, 71
   Arrays, 71
   User-Defined Record Types, 71
F Sample VB Application, 73
G Glossary, 77
Introduction

This guide provides technical information for using the Image Processing DLL as part of end-user Windows applications for the AccuVote Central Count System.

The DLL was developed using Borland Delphi 3 and the Object Pascal language, but the DLL functions can be called from any of the popular Windows development environments, including Visual Basic and Visual C++. Examples are provided for calling the DLL functions from Delphi and VB.

Note: All VB code assumes that the first element of any array structure is numbered 0 (zero).

Summary of DLL Functions

The DLL provides all of the image processing capability needed to support the new AccuVote Central Count System product. That is, functions for:

- Analyzing the geometry of scanned ballot images
- Reading and decoding the timing marks around ballot edges
- Providing on-screen tools for "repairing" ballots whose timing marks cannot be read automatically
- Reading voter marks in ballot "ovals", to classify them as a NO, YES or UNDEFINED votes
- Providing on-screen tools for "duping" ballots with problems such as UNDEFINED votes
- Providing on-screen tools for reading and entering candidate names for write-in votes, when tabulation of these votes is required.

Additional functions are provided for converting data between different formats, and to assist with product testing.
Chapter 1: Introduction

DLL functions are conveniently divided into three groups:

- Image processing functions
- Data conversion functions
- Miscellaneous testing functions.

Function Result Codes

With few exceptions, DLL functions return a result code where 0 represents a successful operation, and other values represent various error conditions.

Unless otherwise stated, the digits of result codes can be interpreted as follows:

- The least-significant digit is normally 0. In some cases, it is used to provide additional detail about the error condition. For example, a 1 might indicate that the error occurred on the face-down or back side of the ballot, as opposed to the face-up or front side.
- The second digit indicates the error type; for example, the image file was unreadable.
- Any remaining digits indicate where the error occurred. See list below.

Table 1. Error sources identified by 3rd and 4th digits of result codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Source Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Error arises with multiple sources</td>
</tr>
<tr>
<td>1</td>
<td>ShowTiffFile function</td>
</tr>
<tr>
<td>2</td>
<td>ReadBallotMarks function (ScanInfo operations)</td>
</tr>
<tr>
<td>3</td>
<td>ReadBallotMarks function (MarkInfo operations)</td>
</tr>
<tr>
<td>4</td>
<td>RepairBallot function</td>
</tr>
<tr>
<td>5</td>
<td>ReadBallotOvals function</td>
</tr>
<tr>
<td>6</td>
<td>DupBallot function</td>
</tr>
<tr>
<td>7</td>
<td>ReadWriteins function</td>
</tr>
<tr>
<td>8..9</td>
<td>Reserved for future DLL functions</td>
</tr>
<tr>
<td>10..19</td>
<td>Reserved for GEMS protocol functions</td>
</tr>
<tr>
<td>20..99</td>
<td>Reserved for calling applications</td>
</tr>
</tbody>
</table>

Some result codes are associated with more than one source. In these cases, only the two least-significant digits are used:
Listing 1. Result codes associated with multiple sources

<table>
<thead>
<tr>
<th>Result Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Successful  = 0;</td>
<td>function completed successfully</td>
</tr>
<tr>
<td>BadTIFF     = 10;</td>
<td>can’t find TIFF file</td>
</tr>
<tr>
<td>BadPage     = 20;</td>
<td>can’t read page from TIFF file</td>
</tr>
<tr>
<td>BadData     = 30;</td>
<td>data causes failure</td>
</tr>
<tr>
<td>Incomplete  = 70;</td>
<td>user saved incomplete work</td>
</tr>
<tr>
<td>UserCancel  = 80;</td>
<td>user cancelled operation</td>
</tr>
<tr>
<td>UserDiscard = 90;</td>
<td>ballot discarded by user</td>
</tr>
</tbody>
</table>

The result code **BadPage** can also have the value 21, which indicates that the problem arose with the scanned image of the face-down surface of the ballot (as opposed to the face-up surface).

**Installation**

ACCSIPDLL makes use of the following files:

**Table 2. ACCSIPDLL files**

<table>
<thead>
<tr>
<th>Filename</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCDLL.DLL</td>
<td>Main DLL file</td>
</tr>
<tr>
<td>CCHELP.TXT</td>
<td>Online help file for DLL functions</td>
</tr>
<tr>
<td>CRDE96V3.DLL</td>
<td>3rd party image processing DLL</td>
</tr>
<tr>
<td>I3SPEC32.DLL</td>
<td>3rd party image processing DLL</td>
</tr>
<tr>
<td>I3TIF32.DLL</td>
<td>3rd party image processing DLL</td>
</tr>
</tbody>
</table>

The following files are provided for informational and testing purposes only:

<table>
<thead>
<tr>
<th>Filename</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MASKS.TXT</td>
<td>Text file of GEMS masks</td>
</tr>
<tr>
<td>README.TXT</td>
<td>Release notes</td>
</tr>
<tr>
<td>CCTEST.EXE</td>
<td>Delphi app to test DLL</td>
</tr>
<tr>
<td>VB*.*</td>
<td>VB6 app to test DLL</td>
</tr>
<tr>
<td>UGCDEFS.PAS</td>
<td>Delphi data definitions for DLL</td>
</tr>
<tr>
<td>CCDLL.DPR</td>
<td>Delphi API for DLL functions</td>
</tr>
</tbody>
</table>

It is recommended that all of these files be installed in the same directory as the calling application.
This chapter describes the data structures associated with ACCSIPDLL, and their associated constant declarations.

The following data structures are explained in alphabetical order:

- tBallot
- tGemsData
- tGemsNames
- tMarkInfo
- tOvalRec
- tScanInfo
- tWriteIn

Additional data structures are used in the GEMS communications protocol. These compressed data formats are described in Appendix B.

**tBallot Record**

**tBallot** is an example of the record structure needed to collect together all of the data associated with a single ballot. **tBallot** can be implemented either:

- Directly as described, when ballot data is stored in a binary file. In this case, **tBallot** fields can be passed directly as parameters to DLL functions, without translation.
- Indirectly in an equivalent data table record, when ballot data is stored in a database. In this case, many database fields will require translation before being passed as parameters to DLL functions.
Typically the second option is preferable.

Listing 2. Delphi declaration

```delphi
const
FNLen = 200; { max file path length }
IDLlen = 4; { max batch/seqno ID lengths }
NeedsMarks = 0; { ready for marks analysis }
NeedsRepair = 1; { needs repair }
Discarded = 2; { discarded by user }
NeedsMask = 3; { ready for GEMS mask }
NeedsVotes = 4; { ready for reading ovals }
NeedsDuping = 5; { needs duping }
NeedsSubmit = 6; { ready for GEMS submission }
Accepted = 7; { votes accepted by GEMS }
Rejected = 8; { rejected by GEMS }
type
tFilePath = array [0..FNLen-1] of char;
tIDField = array [0..IDLlen-1] of char;
tBallot = record
  FPath : tFilePath; { tiff file path + name }
  Batch : tIDField; { batch # }
  SeqNo : tIDField; { sequence # }
  Status : byte; { workflow status; see constants earlier }
  Error : smallint; { latest error code; see constants earlier }
  UserWriteIn : wordbool; { write-ins processed? }
  MarkInfo : tMarkInfo; { mark decoding results }
  Mask : tGemsData; { oval positions, GEMS format }
  Votes : tGemsData; { votes, in GEMS format }
  WIMask : tGemsData; { write-in specs, in GEMS format }
  WINames : tGemsNames; { writein names }
end;

Field Descriptions

**FPath** contains the directory path and filename for the 2-page TIFF file in which the scanned ballot image is stored. Filenames use the syntax `<Batch>_<SeqNo>`.tif.

**Batch** is the number of the batch to which the physical ballot was assigned.

**SeqNo** is the sequence number (within the batch) assigned to the ballot.

**Status** indicates the current status of the ballot in the work flow. See the constant declarations for the supported statuses.
Chapter 2: Data Definitions

**Error** indicates the most recent result code returned by a DLL function.

**UserWritein** is a boolean field that indicates whether the ballot has had its write-in votes processed by the user. See the **ReadWriteins** function.

**MarkInfo** is a record structure that contains the results of analyzing the ballot’s image geometry and timing marks. See **tMarkInfo** and **tScanInfo** later in this chapter.

The **Mask** field stores the ballot’s organization of ovals in the compressed format used by the GEMS protocol.

The **Votes** field stores the ballot’s votes in the compressed format used by the GEMS protocol.

The **WIMask** field stores the dimensional data associated with the ballot’s write-in ovals, using the compressed format used by the GEMS protocol.

The **WINames** field stores the write-in candidate names (as entered during write-in processing by the user), using the compressed format used by the GEMS protocol.

**Tips and Tricks**

It is the responsibility of the calling application to update the **Status** and **Error** fields when a DLL function returns a result code. Result codes are listed in the descriptions of the DLL functions.

It is also the responsibility of the calling application to update the **UserWritein** field when write-in votes are processed by the user.

**See Also**

**tGemsData, tGemsNames, tMarkInfo** and **tScanInfo**.

**tGemsData Record**

**tGemsData** defines a record for exchanging compressed ballot data with the GEMS database, via the GEMS protocol. (The compression schemes used by the GEMS protocol are described in Appendix B.)

The same record structure is used for exchanging:

- Ballot oval masks, the layout of voting ovals on a ballot.
- Votes, the results of reading voter marks in ovals.
- Write-in masks, the details of each voting position that contains a write-in vote.
Listing 3. Delphi declaration

```
const
  GemsBytes = 255;

type
  tGemsBytes = array [0..GemsBytes-1] of byte;
  tGemsData = record
    Size : byte;
    Data : tGemsBytes;
  end;
```

Listing 4. Equivalent VB declaration

```
Const GemsBytes As Byte = 255

Private Type tGemsData
  Size As Byte
  Data(GemsBytes) As Byte
End Type
```

Field Descriptions

Size indicates how many bytes of data are contained in the Data field.

The Data field contains the compressed data in an array of bytes. The data compression algorithm is explained in Appendix B.

Tips and Tricks

ACCSIPDLL provides functions for converting between these compressed data structures and conventional arrays (see Chapter 4).

For examples using this record structure, see the GemsMaskToOvals function.

See Also

tGemsNames data structure; data conversion routines in Chapter 4.

tGemsNames Record

```
tGemsNames defines a compressed record for exchanging write-in names with the GEMS database, via the GEMS protocol. (The compression schemes used by the GEMS protocol are described in Appendix B.)

ACCSIPDLL provides functions for converting between this record structure and a more conventional array (see Chapter 4).
```
Listing 5. Delphi declaration

```
const
  GemsChars = 1000;
type
  tGemsChars = array [0..GemsChars-1] of char;
tGemsNames = record
    Size : byte;
    Data : tGemsChars;
  end;
```

Listing 6. Equivalent VB declaration

```
Private Type tGemsNames
  Size As Byte
  Data As String * 1000
End Type
```

Field Descriptions

**Size** indicates how many characters are contained in the **Data** field.

The **Data** field contains the compressed names in an array of characters.

Tips and Tricks

ACCSIPDLL provides functions for converting between this compressed data structure and conventional arrays (see Chapter 4).

For examples using this record structure, see the **WriteInsToGemsNames** function.

See Also

**tGemsData** data structure; data conversion routines in Chapter 4.

**tMarkInfo Record**

**tMarkInfo** combines the **tScanInfo** geometry of both sides of a ballot with the results of analyzing and decoding the ballot’s timing marks.

Many of the fields in **tMarkInfo** are of interest to calling applications, but they must also be stored so that they can be passed to DLL functions called later.
Listing 7. Delphi declaration

```delphi
const
MaxCols = 34;
type
tMarkChars = array [0..MaxCols-3] of { string of 32 0/1 } char;
tMarkInfo = record
  ScanInfo : array [0..1] of { geometry for each side }
  of tScanInfo;
  Flipped : wordbool; { scanned face down? }
  Inverted : wordbool; { scanned feet first? }
  Repaired : wordbool; { repaired by user? }
  Duped : wordbool; { duped by user? }
  BotMarks : array [0..1] of { bottom marks as 0/1 chars }
      tMarkChars;
  CardRotID : smallint; { card identifier }
  PrecinctID : smallint; { precinct ID # or batch # }
  ElecType : byte; { e.g. 6 => "G"eneral }
  ElecDate : smallint; { election date }
end;
```

Listing 8. Equivalent VB declaration

```vbnet
Private Type tMarkInfo
  ScanInfo(1) As tScanInfo
  Flipped As Boolean
  Inverted As Boolean
  Repaired As Boolean
  Duped As Boolean
  MarksFront As String * 32
  MarksBack As String * 32
  CardRotID As Integer
  PrecinctID As Integer
  ElecType As Byte
  ElecDate As Integer
End Type
```

Field Descriptions

The `ScanInfo` field is described in the `tScanInfo` section later.

The booleans `Flipped` and `Inverted` indicate whether the ballot was scanned face down and/or feet first, respectively.

The booleans `Repaired` and `Duped` indicate whether the ballot was repaired and/or duped by the user, at any time.

`MarksFront` and `MarksBack` store the bottom timing marks on each side of the ballot, as represented by a string of 32 "0"s or "1"s. Note that the corner timing marks never contain coded information, and are discarded. Thus, the 34 columns of marks are reduced to 32 marks of interest. Note that these two fields contain the front and back marks as indicated, regardless of the ballot's orientation during
scanning.

The remaining fields contain the result of decoding the bottom marks:

**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.

**PrecinctID** normally indicates the precinct number where the ballot was used. With absentee ballots this field has a value of zero, except in the case of batch header cards, where this field contains the new batch number.

**ElecType** contains a numeric code for the "type" of election in which the ballot was used. The letters "A" through "Z" are assigned numeric values 0 through 25. Thus a general election ("G") is assigned the value 6.

**ElecDate** contains the date of the election for which the ballot was intended. This date is represented by a 16-bit integer where the 5 least significant bits are the day-of-month (1..31), the next 4 bits are the month (1..12), and the next 7 bits are the 2-digit year (0..99). This is the same date encoding scheme as used in the GEMS protocol (see Appendix B).

**Tips and Tricks**

The **CardRotID** and **PrecinctID** fields take on special significance in starter and ender cards:

**Table 3. Use of CardRotID and PrecinctID fields in different card types**

<table>
<thead>
<tr>
<th>Card type</th>
<th>CardRotID</th>
<th>Precinct ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Polling place ballot</td>
<td>&lt;65530</td>
<td>Precinct number</td>
</tr>
<tr>
<td>User deck*</td>
<td>65530</td>
<td>Batch number</td>
</tr>
<tr>
<td>Generic deck</td>
<td>65531</td>
<td>0</td>
</tr>
<tr>
<td>Precinct header</td>
<td>65532</td>
<td>Override precinct ID</td>
</tr>
<tr>
<td>Test</td>
<td>65533</td>
<td>Precinct to test</td>
</tr>
<tr>
<td>Absentee ballot</td>
<td>65534</td>
<td>0</td>
</tr>
<tr>
<td>Ender card</td>
<td>65535</td>
<td>0</td>
</tr>
</tbody>
</table>

* Central Count batch

The **Repaired** and **Duped** fields are automatically updated after the completion of a call to the **RepairBallot** or **DupBallot** function, respectively.

The **ElecDate** field can be converted between the encoded GEMS format and a conventional YYMMDD format by calling the DLL functions **GemsDateToYYMMDD** and **YYMMDDToGemsDate**.
See Also
tScanInfo data structure.

**tOvalRec Record**

*tOvalRec* provides an array for storing uncompressed oval mask and votes data for a ballot. This data structure is not needed for routine ballot processing, but it is helpful when mask or vote data needs to be accessed for display, reporting, debugging or other purposes. In these situations, the DLL’s data conversion functions can be used to convert compressed data (in GEMS format) into a *tOvalRec*. See, for example, the **GemsMaskToOvals** function.

**Listing 9. Delphi declaration**

```delphi
const
  MaxCandidates = 500;
  { vote status }
  No = 0;        { same as in GEMS }
  Yes = 1;
  Undef = 2;     { in between white & black }

type
  tOvalRec = record
    Col : byte;    { oval’s column }
    Row : byte;    { oval’s row }
    Vote : byte;   { vote status; see const above }
  end;
  tOval = array [0..MaxCandidates-1] { all ovals on a ballot } of tOvalRec;
```

**Listing 10. Equivalent VB declaration**

```vbnet
Private Type tOval
  Col As Byte
  Row As Byte
  Vote As Byte
End Type

Dim Oval(499) As tOval
```

**Field Descriptions**

**Col** stores the column number in which the oval is located. In this context, column 0 is the left-most usable column, disregarding the left-edge column of timing marks.

**Row** stores the row number in which the oval is located. In this context, row 0 is the upper-most usable row, disregarding the top row of timing marks.
Chapter 2: Data Definitions

Vote indicates the result of reading voter marks in the oval. This field can have three states: 0 = NO vote, 1 = YES vote, and 2 = UNDEF (undefined) vote. An undefined vote is classified as too black to be a NO vote, but too white to be a YES vote. It must be resolved by user duping (see DupBallot function).

See Also
- GemsMaskToOvals, GemsVotesToOvals, OvalsToGemsMask and OvalsToGemsVotes functions.

**tScanInfo Record**

**tScanInfo** describes the record structure used to store the image geometry for one side of a ballot. Field values are established when a ballot’s two images are analyzed by the ReadBallotMarks function.

Two **tScanInfo** records are used in the **tMarkInfo** record structure: one for each side of a ballot. Additional fields in **tMarkInfo** store the results of decoding ballot timing marks.

Normally the calling application makes no use of the **tScanInfo** data. It simply stores the data so that it is available for later calls to other image processing functions.

**Listing 11. Delphi declaration**

```delphi
const
MaxRows = 69; { max rows of timing marks }
type
  tYPos = array [0..MaxRows-1] of smallint; { Y pos for all marks along an edge }
  tScanInfo = record { geometry for one side }
    ImageW : smallint; { TIFF image width, pixels }
    ImageH : smallint; { TIFF image height, pixels }
    Rows : byte; { rows, defined by ballot ht }
    Scans : smallint; { # of lines scanned (testing) }
    X1T, Y1T : smallint; { top left }
    X1B, Y1B : smallint; { bottom left }
    X2T, Y2T : smallint; { top right }
    X2B, Y2B : smallint; { bottom right }
    YPosL, YPosR : tYPos; { pos of all marks; L side; R side }
  end;
  tMarkInfo = record
    ScanInfo : array [0..1] of tScanInfo; { geometry for each side }
    ... { 0 = top, 1 = bottom }
  end;
```
Listing 12. Equivalent VB declaration and example

```vbnet
Const MaxRows As Byte = 68

Private Type tScanInfo
    ImageW As Integer
    ImageH As Integer
    Rows As Byte
    Scans As Integer
    X1T As Integer
    Y1T As Integer
    X1B As Integer
    Y1B As Integer
    X2T As Integer
    Y2T As Integer
    X2B As Integer
    Y2B As Integer
    YPosL(MaxRows) As Integer
    YPosR(MaxRows) As Integer
End Type

Private Type tMarkInfo
    ScanInfo(1) As tScanInfo
    ... End Type
End Type
```

Field Descriptions

**ImageW** and **ImageH** are the width and height of the ballot image, in pixels. Images are stored in landscape mode, so an 11" ballot has typical values of 2200 and 1700, respectively. Note that these two fields are the only data that describes the ballot from a landscape perspective; all other data assumes a portrait perspective. The **ImageW** field identifies the ballot height as 11, 14, 17 or 18".

**Rows** contains the number of rows of timing marks required on a ballot of the current size. 11, 14, 17 and 18" ballots have 41, 53, 65 and 69 rows, respectively.

**Scans** is used for testing purposes only. It reports how many columns of pixels had to be analyzed in order to determine the image geometry. (High performance depends on keeping this number as low as possible.)

The eight **X** and **Y** fields provide the (X, Y) pixel location of the center of each of the four corner timing marks. Suffixes "T" and "B" indicate the left and right edges of the ballot; and "T" and "B" indicate the top and bottom edges. Thus "X2B" is the (X) position of the lower right corner mark.

**YPosL** and **YPosR** provide the (Y) pixel locations of the centers of every mark along the left and right edges of the ballot. It is necessary to capture every mark position, because of variations in scanning speed.

In the **tMarkInfo** record structure, the **ScanInfo** field provides an array of two **tScanInfo** records, **ScanInfo[0]** and **ScanInfo[1]**. These store the geometry of the top and bottom scanned surfaces of the ballot, respectively.
Chapter 2: Data Definitions

Tips and Tricks

Normally ballots are scanned face up, head first.

If a ballot is scanned face down, the ScanInfo[0] record will contain the back side’s geometry, and ScanInfo[1] will contain the front side.

If a ballot is scanned feet first, the references to the "top" and "bottom" of the ballot are reversed. That is, "X1T" will refer to the lower right corner rather than the upper left.

See Also

tMarkInfo data structure.

tWriteIn Record

tWriteIn provides an array for storing uncompressed write-in data for a ballot. This data structure is not needed for routine ballot processing, but it is helpful when write-in data needs to be accessed for display, reporting, debugging or other purposes. In these situations, the DLL’s data conversion functions can be used to convert compressed data (in GEMS format) into a tWriteIn record. See, for example, the GemsWI MaskToWriteins function.

Listing 13. Delphi declaration

```delphi
const
WriteinLen    = 32;
MaxWriteIns   = 50;
type
tName = array [0..WriteinLen] { candidate name } of char;
tWriteInRec = record
  Index : smallint; { index into Oval array }
  X1 : byte; { left edge }
  Y1 : byte; { top edge }
  X2 : byte; { right edge }
  Y2 : byte; { bottom edge }
  Name : tName; { candidate name }
end;
tWriteIn = array [0..MaxWriteIns-1] { all writeins on a ballot } of tWriteInRec;
```

Listing 14. Equivalent VB declaration and example

```vbnet
Private Type tWriteIn
  Index As Integer
  X1 As Byte
  Y1 As Byte
  X2 As Byte
  Y2 As Byte
  Name As String(32)
End Type
```
Name As String * 32 
End Type 

Dim WriteIn(49) As tWriteIn

Field Descriptions 
**Index** indicates which of the ovals in the ballot mask the write-in data applies to. An **Index** of 0 corresponds to the first oval in the mask. 

**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. 

**Name** stores the write-in candidate's name, as entered or selected by the user during write-in processing. See the **ReadWriteIns** function. 

See Also 
**ReadWriteIns, GemsWI MaskToWriteins, WriteInsToGemsWI Mask, GemsNamesToWriteins** and **WriteInsToGemsNames** functions.
This chapter describes the DLL functions that provide image processing capabilities.

The following functions are described in the order that they are normally used in the ballot processing workflow:

- ReadBallotMarks
- RepairBallot
- ReadBallotOvals
- DupBallot
- ReadWriteIns

**Note:** In the VB examples, variables are often defined as 1-element arrays of a user-defined record, and are passed to a DLL function parameter by reference to the first element (rather than by reference to the variable itself). See example below. Simple arrays are also passed by reference to their first elements. The reasons for this can be found on page 71.

**Listing 15. VB example using 1-element VB arrays for user-defined record variables**

```vbnet
Private Declare Function ReadBallotMarks Lib "CCDLL.DLL" (  ByVal Filename As String,  ByVal MarkInfo As tMarkInfo ) As Integer

Dim MarkInfo(0) As tMarkInfo
Dim Error As Integer

Error = ReadBallotMarks("InTray\0005_3516.tif", MarkInfo(0))
```
ReadBallotMarks Function

This function analyzes image files to determine the precise geometry of the scanned ballot, based on the locations of the timing marks along the left and right edges of each side. This geometry data is essential for accurately decoding information in the bottom rows of timing marks, and for reading the votes represented by filled-in "ovals."

This function also decodes and validates the timing marks, to ensure that they are read accurately.

Decoding the timing marks provides the ballot’s election type and date, and the card number ("CardRotID") that uniquely identifies the arrangement of races and candidates on the ballot. This information is used to obtain from GEMS the "mask" that identifies the locations of ovals on the ballot. Decoding the marks also yields the precinct or batch number (see table on page 10).

Listing 16. Delphi API

```delphi
function ReadBallotMarks (Filename : pChar; { 2-page TIFF filename } var MarkInfo : tMarkInfo { results of mark analysis } ) : smallint; stdcall; { result code }
```

Listing 17. VB declaration and typical function call

```vb
Private Declare Function ReadBallotMarks Lib "CCDLL.DLL" ( ByVal Filename As String, ByRef MarkInfo As tMarkInfo ) As Integer

Dim MarkInfo(0) As tMarkInfo
Dim Error As Integer

Error = ReadBallotMarks("InTray\0005_3516.tif", MarkInfo(0))
```

Parameter Descriptions

**Filename** provides the path and filename for the 2-page TIFF file containing the scanned ballot images.

The **MarkInfo** parameter is updated with the results of this function. If the operation is unsuccessful, this parameter is returned in an initialized state.

The function returns a result code from the list below. If a result code has a least significant digit of 1, this indicates that the error occurred on the bottom scanned surface, rather than the top.
Chapter 3: Image Processing Function Calls

Listing 18. ReadBallotMarks Result codes

<table>
<thead>
<tr>
<th>Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful</td>
</tr>
<tr>
<td>10</td>
<td>BadTIFF</td>
</tr>
<tr>
<td>20</td>
<td>BadPage</td>
</tr>
<tr>
<td>30</td>
<td>BadData</td>
</tr>
<tr>
<td>220</td>
<td>BadWidth</td>
</tr>
<tr>
<td>230</td>
<td>BadHeight</td>
</tr>
<tr>
<td>240</td>
<td>BadSkew</td>
</tr>
<tr>
<td>250</td>
<td>BadMarks</td>
</tr>
<tr>
<td>260</td>
<td>BadGeomX</td>
</tr>
<tr>
<td>270</td>
<td>BadGeomY</td>
</tr>
</tbody>
</table>

RepairBallot Function

This function is used when an image is rejected by the previous function. There are many causes of rejection, including torn or defaced ballots, and documents that aren't ballots at all.

Figure 1. User dialog for repairing a ballot

The RepairBallot function displays the scanned image on the screen, and leads the user through a repair "wizard" comprising three simple questions:
Is the ballot displayed the right way up? If the user answers no, the display is inverted, and the Inverted flag in tMarkInfo is set to true.

Is the front of the ballot displayed? If the user answers no, the other side of the ballot is displayed, and the Flipped flag in tMarkInfo is set to true (unless the back of the ballot was displayed only because of the user earlier using the Turn Over button).

What is the card number (CardRotID) printed at the bottom of the ballot? The user reads the card number from a magnified display of the appropriate area of the ballot, then types it into an edit field. This information is combined with the ElecType and ElecDate parameters, to generate the MarksFront and MarksBack fields of tMarkInfo. The CardRotID, ElecType and ElecDate fields of tMarkInfo are also filled in.

With these questions answered, the system has enough information to retrieve the correct masks from GEMS, and to display the ballot in the correct orientation during duping.

**Listing 19. Delphi API**

```delphi
define RepairBallot (  
Filename : pChar; { 2-page TIFF filename }  
ElecType : byte; { election type, eg 6 = "G" }  
ElecDate : smallint; { election date, in GEMS format }  
var MarkInfo : tMarkInfo { results of IDing ballot }  
) : smallint; stdcall; { result code }
```

**Listing 20. VB declaration and example**

```vbnet
Private Declare Function RepairBallot Lib "CCDLL.DLL" (  
ByVal Filename As String,  
ByVal ElecType As Byte,  
ByVal ElecDate As Integer,  
ByRef MarkInfo As tMarkInfo  
) As Integer

Dim MarkInfo(0) As tMarkInfo  
Dim Error As Integer

Error = RepairBallot(\"InTray\0005_3516.tif\", 6, 596, MarkInfo(0))
```

**Parameter Descriptions**

**Filename** provides the path and filename for the 2-page TIFF file containing the scanned ballot images.

**ElecType**
ElecDate

The MarkInfo parameter is updated with the results of this function. If the operation is canceled by the user, this parameter is returned in an initialized state.

The function returns a result code from the list below. If a result code has a least significant digit of 1, this indicates that the error occurred on the bottom scanned surface, rather than the top.

Listing 21. RepairBallot result codes

<table>
<thead>
<tr>
<th>Result Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful; function completed successfully</td>
</tr>
<tr>
<td>10</td>
<td>BadTIFF; can't find TIFF file</td>
</tr>
<tr>
<td>20</td>
<td>BadPage; can't read page from TIFF file</td>
</tr>
<tr>
<td>80</td>
<td>UserCancel; user cancelled operation</td>
</tr>
<tr>
<td>90</td>
<td>UserDiscard; ballot discarded by user</td>
</tr>
</tbody>
</table>

Tips and Tricks

If the user saves a repaired ballot, this function also sets the UserRepair flag in the MarkInfo parameter.

The ReadBallotOvals function cannot be used to read the votes on repaired ballots, because their precise geometry is unknowable. Thus, the next steps in the workflow for repaired ballots are: (1) retrieve the ovals mask from GEMS; then (2) queue the ballot for duping.

ReadBallotOvals Function

This function uses the ballot's geometry data and GEMS mask to read the vote in each of the ovals defined in the GEMS mask. The percentage of black pixels in each oval is calculated, and used to define the vote as "No", "Yes", or "Undefined." The last of these states represents markings that are too black to be classified as NO votes, but not black enough to be classified as YES. If a ballot contains one or more Undefined votes, this function returns a specific error code to indicate that the ballot requires on-screen duping (see next function).

Listing 22. Delphi API

```delphi
function ReadBallotOvals (Filename : pChar; MaxPages : integer; var MarkInfo : tMarkInfo; var Mask : tGemsData; var Votes : tGemsData) : smallint; stdcall; { result code }
```
ReadBallotOvals Function

Listing 23. VB declaration and example

Private Declare Function ReadBallotOvals Lib "CCDLL.DLL" (  
    ByVal Filename As String,  
    ByRef MarkInfo As tMarkInfo,  
    ByRef Mask As tGemsData,  
    ByRef Votes As tGemsData  
) As Integer

[Add example]

Parameter Descriptions

Filename provides the path and filename for the 2-page TIFF file containing the scanned ballot images.

MarkInfo provides the image geometry for each side of the ballot, and the timing mark information needed by this function.

Mask provides the ovals mask for the ballot, in the GEMS protocol format.

Votes returns the results of reading the marks in each of the ballot's ovals, in the GEMS protocol format. Each vote is classified as either No, Yes or Undefined.

The function returns a result code from the list below. If a result code has a least significant digit of 1, this indicates that the error occurred on the front of the ballot, rather than the back. The Undefined error (510) indicates that the operation was completed successfully, except for the existence of Undefined votes that will require duping.

Listing 24. ReadBallotOvals result codes

<table>
<thead>
<tr>
<th>Result Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Successful</td>
<td>= 0;</td>
</tr>
<tr>
<td>BadTIFF</td>
<td>= 10;</td>
</tr>
<tr>
<td>BadPage</td>
<td>= 20;</td>
</tr>
<tr>
<td>BadData</td>
<td>= 30;</td>
</tr>
<tr>
<td>Undefined</td>
<td>= 510;</td>
</tr>
</tbody>
</table>

DupBallot Function

This function is used for on-screen duping of ballots with Undefined votes, or ballots that were repaired using the RepairBallot function.
Figure 2. User dialog for duping a ballot

The scanned image is displayed on the screen, and each vote is marked with a color code to indicate how its marks were interpreted:

- A green "O" indicates a No vote
- A green "X" indicates a Yes vote
- A red "?" indicates an Undefined vote.

A panel to the right of the main display shows how many Undefined votes remain to be duped on each side of the ballot. There is also a small Magnifier area that shows an enlarged image of the region around the mouse cursor position.

If the voter's intent cannot be determined from the displayed image, a Discard button is provided to reject the ballot from the election.

Listing 25. Delphi API

```delphi
function DupBallot (Filename: pChar; var MarkInfo: tMarkInfo; var Mask: tGemsData; var Votes: tGemsData): smallint; stdcall; // result code
```
Listing 26. VB declaration and example

Private Declare Function DupBallot Lib "CCDLL.DLL" (  
   ByVal Filename As String,  
   ByRef MarkInfo As tMarkInfo,  
   ByRef Mask As tGemsData,  
   ByRef Votes As tGemsData  
) As Integer

[Add example]

Parameter Descriptions

Filename provides the path and filename for the 2-page TIFF file containing the scanned ballot images.

MarkInfo provides the image geometry for each side of the ballot, and the timing mark information needed by this function.

Mask provides the ovals mask for the ballot, in the GEMS protocol format.

Votes returns the results of reading the marks in each of the ballot's ovals, in the GEMS protocol format. Each vote is classified as either No, Yes or Undefined.

The function returns a result code from the list below. If a result code has a least significant digit of 1, this indicates that the error occurred on the front of the ballot, rather than the back.

Listing 27. DupBallot result codes

<table>
<thead>
<tr>
<th>Result Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful = function completed successfully</td>
</tr>
<tr>
<td>10</td>
<td>BadTIFF = can’t find TIFF file</td>
</tr>
<tr>
<td>20</td>
<td>BadPage = can’t read page from TIFF file</td>
</tr>
<tr>
<td>30</td>
<td>BadData = data causes failure</td>
</tr>
<tr>
<td>70</td>
<td>NothingToDo = ballot has no undefined votes</td>
</tr>
<tr>
<td>80</td>
<td>UserCancel = user cancelled operation</td>
</tr>
<tr>
<td>90</td>
<td>UserDiscard = ballot discarded by user</td>
</tr>
</tbody>
</table>

ReadWriteIns Function

This function is used for on-screen processing of ballots with write-in votes. It is used when a jurisdiction is required to tabulate such votes.

This function provides the user with the following dialog box for entering the names of write-in candidates (or selecting their names from a list of known write-in candidates).
Figure 3. User dialog for processing write-in votes

In the top left corner of the window is an indicator of how many write-in votes were made on the current ballot, and which one is currently displayed. For example, "Write-in vote 1 of 4."

The magnifier shows any marks made by the voter in the area assigned to the current write-in vote.

The candidate’s name is typed into the Candidate Name box, or selected from the drop-down list of known candidates. Names are automatically converted to uppercase letters, so entries are case-insensitive.

The Next and Prev buttons are used to move to the next or previous write-in vote, respectively.

The Save button saves all entries by returning them in the function’s Names parameter. The Cancel button closes the window without saving any entries made.

When write-in names are saved, the file of known names is automatically updated with any new names added with the current ballot.

Listing 28. Delphi API

function ReadWriteIns (  
  var Filename : pChar; { 2-page TIFF filename }  
  var NamesFile : pChar; { list of write-in candidates }  
  var MarkInfo : tMarkInfo; { results of IDing ballot }  
  var Mask : tGemsData; { ovals mask for card number }  
  var Votes : tGemsData; { votes made on ballot }  
  var WIMask : tGemsData; { writein mask, GEMS format }  
  var Names : tGemaNames { writein names, GEMS format }  
) : smallint; stdcall; { result code }
Listing 29. Equivalent VB declaration and example

```vbnet
Private Declare Function ReadWriteIns Lib "CCDLL.DLL" ( _
    ByVal Filename As String, _
    ByVal NamesFile As String, _
    ByRef MarkInfo As tMarkInfo, _
    ByRef Mask As tGemsData, _
    ByRef Votes As tGemsData, _
    ByRef WIMask As tGemsData, _
    ByRef Names As tGemsNames _
) As Integer
```

Parameter Descriptions

- **Filename** provides the path and filename for the 2-page TIFF file containing the scanned ballot images.

- **NamesFile** provides the path and filename for the ASCII text file containing the list of known write-in candidate names. If the file does not exist, it will be automatically created during the save operation.

- **MarkInfo** provides the image geometry for each side of the ballot, and the timing mark information needed by this function.

- **Mask** provides the ovals mask for the ballot, in the GEMS protocol format.

- **Votes** provides the results of reading the marks in each of the ballot's ovals, in the GEMS protocol format. Each vote is classified as either No, Yes or Undefined.

- **WIMask** must provide the details of write-in vote locations on the ballot, as obtained from GEMS or the equivalent.

- **Names** returns the candidate name for each write-in vote made by the voter, as interpreted by the user.

The function returns a result code from the list below. The result code **NothingToDo** indicates that the function call would not affect the ballot, and that no user action was sought or needed.

Listing 30. Result codes

```vbnet
Successful = 0;  { function completed successfully }
BadTIFF   = 10;  { can't find TIFF file }
BadPage   = 20;  { can't read page from TIFF file }
BadData   = 30;  { data causes failure }
NothingToDo = 70; { no write-in votes to process }
UserCancel = 80;  { user cancelled operation }
```
Tips and Tricks
All undefined votes must be resolved before passing a ballot to this function.
To avoid file sharing issues, it is recommended that each workstation have an independent local file of known write-in candidates.

See Also
GemsWIMaskToWriteIns, WriteInsToGemsWI Mask, GemsNamesToWriteIns and WriteInsToGemsNames functions.
Data Conversion Functions

These functions convert ballot data between:

- The compressed formats used in both the GEMS protocol and the DLL function calls; and
- A conventional array structure (or other variable) that simplifies data access to individual elements.

Usage of these functions is not normally required in the basic ballot processing workflow. They can be useful tools during software development, and can also satisfy application reporting requirements.

GemsDateToYYMMDD Function

This function converts a date from the compressed format used in ballot marks and the GEMS protocol to a long integer YYMMDD format.

Listing 31. Delphi API

```delphi
function GemsDateToYYMMDD (GemsDate : smallint) : longint; stdcall; { same in YYMMDD format }
```

Listing 32. Equivalent VB declaration and example

```vbnet
Private Declare Function GemsDateToYYMMDD Lib "CCDLL.DLL" _
    (ByVal GemsDate as Integer) As Long

Dim YYMMDD As Long

YYMMDD = GemsDateToYYMMDD(Ballot.ElecDate)
```
Chapter 4: Data Conversion Functions

Tips and Tricks
The compressed format packs dates into a 16-bit integer, with 5 bits for the day-of-month (1..31), 4 bits for the month (1..12), and 7 bits for the 2-digit year (0..99). Thus November 7th 2000 (decimal 1107) is compressed into binary 0000000101100111 (decimal 359).

Starting January 1, 2064, the most significant bit of a GEMS date will be 1, resulting in a negative value if the date is interpreted as a signed 16-bit integer.

See Also
YYMMDDToGemsDate function.

GemsMaskToOvals Function

This function converts a ballot oval mask from the compressed GEMS format to a conventional array (see tOval declaration earlier).

Listing 33. Delphi API

```delphi
function GemsMaskToOvals (  
  var Mask : tGemsData;   { GEMS mask to decode into Ovals }  
  var Ovals : tOval;     { decoded results }  
) : smallint; stdcall; { result code }
```

Listing 34. VB declaration and example

```vbnet
Private Declare Function GemsMaskToOvals Lib "CCDLL.DLL"  
(ByRef Mask As tGemsData, ByRef Ovals As tOval) As Integer

Dim Mask(0) As tGemsData  
Dim Ovals(499) As tOval

GemsMaskToOvals(Mask(0), Ovals(0))  
' modify mask here  
OvalsToGemsMask(Ovals(0), Mask(0))
```

Parameter Descriptions
- **Mask** provides the ovals mask for the ballot, in the GEMS protocol format.
- **Ovals** contains the results of the data conversion.

The function returns a result code from the list below.
GemsMaskToOvals Function

Listing 35. GemsMaskToOvals result codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful = function completed successfully</td>
</tr>
<tr>
<td>30</td>
<td>BadData = data causes failure</td>
</tr>
</tbody>
</table>

See Also
OvalsToGemsMask, GemsVotesToOvals and OvalsToGemsVotes functions.

GemsNamesToWriteins Function

This function converts write-in names from the compressed GEMS format to a conventional array (see tWritein declaration earlier).

Listing 36. Delphi API

```delphi
function GemsNamesToWriteins ( var Names : tGemsNames; var Writein : tWritein ) : smallint; stdcall;
```

Listing 37. VB declaration and example

```vbnet
Private Declare Function GemsNamesToWriteins Lib "CCDLL.DLL" (ByRef Names As tGemsNames, ByRef Writein As tWritein) As Integer

GemsNamesToWritein(Ballot.Names(0), Writein(0))
' display write-in names here
```

Parameter Descriptions

- **Names** provides the write-in candidate names used on a ballot, in the GEMS protocol format.
- **Writeins** return the results of the data conversion.

The function returns a result code from the list below.

Listing 38. GemsNamesToWriteins result codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful = function completed successfully</td>
</tr>
<tr>
<td>30</td>
<td>BadData = data causes failure</td>
</tr>
</tbody>
</table>
See Also
WriteinsToGemsNames function.

**GemsVotesToOvals Function**

This function converts vote data from the compressed GEMS format to the conventional array in `tOval`.

**Listing 39. Delphi API**

```delphi
function GemsVotesToOvals (  
    var Votes : tGemsData;  { GEMS vote data }  
    var Ovals : tOval;  { decode results }  
) : smallint; stdcall; { result code }  
```

**Listing 40. VB declaration and example**

```vbnet
Private Declare Function GemsVotesToOvals Lib "CCDLL.DLL"  
    (ByRef Votes As tGemsData, ByRef Ovals As tOval) As Integer

Dim Votes As tGemsData
Dim Ovals As tOval

GemsVotesToOvals(Votes(0), Ovals(0))  
' modify oval data here  
OvalsToGemsVotes(Ovals(0), Votes(0))
```

**Parameter Descriptions**

**Votes** provides the ballot’s voting results, in the GEMS protocol format.

**Ovals** returns the results of the data conversion. The **Row** and **Col** fields in this array are untouched by this function.

The function returns a result code from the list below.

**Listing 41. Result codes**

```plaintext
Successful  = 0;  { function completed successfully }  
BadData     = 30;  { data causes failure }  
```
GemsVotesToOvals Function

See Also
OvalsToGemsVotes, OvalsToGemsMask and GemsMaskToOvals functions.

GemsWIMaskToWriteins Function

This function converts write-in mask data from the compressed GEMS protocol format to the conventional array in tWritein.

Listing 42. Delphi API

function GemsWIMaskToWriteins (  
  var WIMask : tGemsData;  { writein mask, GEMS format }  
  var Writein : tWritein  { writein data, local format }  
) : smallint; stdcall;

Listing 43. VB declaration and example

Private Declare Function GemsWIMaskToWriteins Lib "CCDLL.DLL"  
  (ByRef WIMask As tGemsData, ByRef Writein As tWritein) As Integer

GemsWIMaskToWriteins(Ballot.WIMask, Writein(0))  
' modify write-in mask here  
WriteinsToGemsWIMask(Writein(0), Ballot.WIMask(0))

Parameter Descriptions

WIMask provides the write-in mask for the ballot, in the GEMS protocol format (see page 59).

Writein returns the results of the data conversion. The Names fields in this array are initialized by this function.

The function returns a result code from the list below.

Listing 44. GemsWIMaskToWriteins result codes

Successful = 0;  { function completed successfully }  
BadData = 30;   { data causes failure }
See Also

WriteinsToGemsWI Mask, WriteinsToGemsVotes and GemsVotesToWriteins functions.

OvalsToGemsMask Function

This function converts a ballot mask from the conventional array in tOval to the compressed GEMS format.

Listing 45. Delphi API

```delphi
function OvalsToGemsMask (var Ovals : tOval; var Mask : tGemsData) : smallint; stdcall; { result code }
```

Listing 46. VB declaration and example

```vbnet
Private Declare Function OvalsToGemsMask Lib "CCDLL.DLL" (ByRef Ovals As tGemsData, ByRef Mask As tOval) As Integer

GemsMaskToOvals(Ballot.Mask, Ballot.Ovals)
    ' modify mask here
OvalsToGemsMask(Ballot.Ovals, Ballot.Mask)
```

Parameter Descriptions

- **Ovals** provides the location and voting status for each oval, in a conventional array.
- **Mask** returns the results of the conversion to the GEMS protocol format.

The function returns a result code from the list below.

Listing 47. Result codes

```plaintext
Successful = 0; { function completed successfully }
BadData = 30; { data causes failure }
```
See Also

OvalsToGemsMask, OvalsToGemsVotes and GemsVotesToOvals functions.

OvalsToGemsVotes Function

This function converts vote data from the conventional array in tOval to the compressed GEMS format.

Listing 48. Delphi API

```delphi
function OvalsToGemsVotes (var Ovals : tOval; var Votes : tGemsData) : smallint; stdcall; { result code }
```

Listing 49. VB declaration and example

```vbnet
Private Declare Function OvalsToGemsVotes Lib "CCDLL.DLL"_
    (ByRef Ovals As tGemsData, ByRef Votes As tOval) As Integer

GemsVotesToOvals(Ballot.Votes, Ballot.Ovals)
' modify oval data here
OvalsToGemsVotes(Ballot.Ovals, Ballot.Votes)
```

Parameter Descriptions

Ovals provides the location and voting status for each oval, in a conventional array.

Votes provides the results of converting votes into the GEMS protocol format.

The function returns a result code from the list below.

Listing 50. Result codes

```plaintext
Successful   = 0;  { function completed successfully }
BadData       = 30; { data causes failure }
```

See Also

GemsVotesToOvals, GemsMaskToOvals and OvalsToGemsMask functions.
Chapter 4: Data Conversion Functions

WriteinsToGemsWIMask Function

This function converts write-in mask data from the conventional array in `tWritein` to the compressed GEMS format.

**Listing 51. Delphi API**

```delphi
function WriteinsToGemsWIMask (  
  var Writein : tWritein;  
  var WIMask : tGemsData;  
) : smallint; stdcall;
```

**Listing 52. VB declaration and example**

```vbnet
Private Declare Function WriteinsToGemsWIMask Lib "CCDLL.DLL"  
  (ByRef Writein As tWritein, ByRef WIMask As tGemsData) As Integer

GemsWIMaskToWriteins(Ballot.WIMask, Writein(0))  
  ' modify write-in mask here  
WriteinsToGemsWIMask(Writein(0), Ballot.WIMask(0))
```

**Parameter Descriptions**

- **Writein** provides the write-in mask data and candidate name for each write-in vote position.
- **WIMask** returns the results of the data conversion in the GEMS protocol format (see page 59).

The function returns a result code from the list below.

**Listing 53. Result codes**

<table>
<thead>
<tr>
<th>Result</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Successful</td>
<td>0</td>
</tr>
<tr>
<td>BadData</td>
<td>30</td>
</tr>
</tbody>
</table>

**See Also**

- `WriteinsToGemsWIMask`
- `WriteInsToGemsNames`
- `GemsNamesToWriteins` functions.
YYMMDDToGemsDate function

This function converts a date from a long integer YYMMDD format to the compressed format used in ballot marks and the GEMS protocol.

Listing 54. Delphi API

function YYMMDDToGemsDate (  
    YYMMDD : longint  
    ) : smallint; stdcall; { same in marks format }

Listing 55. Equivalent VB declaration and example

Private Declare Function YYMMDDToGemsDate Lib "CCDLL.DLL" _
    (ByVal YYMMDD as long) As Integer

Dim YYMMDD As Long

YYMMDD = 001107
Ballot.ElecDate = YYMMDDToGemsDate(YYMMDD)

Parameter Descriptions

YYMMDD provides the date to be converted, in a signed 32-bit integer containing the year, month and day.

The function returns the results of the date compression into a 16-bit signed integer.

Tips and Tricks

The compressed format packs dates into a 16-bit signed integer, with 5 bits for the day-of-month (1..31), 4 bits for the month (1..12), and 7 bits for the 2-digit year (0..99). Thus November 7th 2000 (decimal 1107) is compressed into binary 0000000101100111 (decimal 359).

Starting January 1, 2064, the most significant bit of a GEMS date will be 1, resulting in a negative value if the compressed date is interpreted as a signed 16-bit integer.

See Also

GemsDateToYYMMDD function.
The ACCSIPDLL also provides a collection of functions whose primary purpose is in testing the Central Count 2 application. These functions would normally not be involved in end-user activities.

The following functions are described in alphabetical order:

- GetBallotMask
- GetDLLVersion
- GetWriteInMask
- ShowTiffFile

**GetBallotMask Function**

This function is intended for use during software development, when communications with the GEMS database is not available. The function returns the ovals mask for the specified ballot style, in exactly the same format as GEMS. In this case, ballot oval masks (and optional write-in masks) are stored in an ASCII text file.

**Listing 56. Delphi API**

```delphi
function GetBallotMask (  
  Filename : pChar;  { mask text filepath, eg MASKS.TXT }  
  CardRotID : longint;  { card rot ID # }  
  var Mask : tGemsData  { GEMS style mask }  
) : smallint; stdcall;
```
GetBallotMask Function

Listing 57. VB declaration and example

Private Declare Function GetBallotMask Lib "CCDLL.DLL" _
(ByVal Filename As String, _
ByVal CardRotID As Long, _
ByRef Mask As tGemsData) _
As Integer

with Ballot
    .FPath = "InTray\" + tTiffName.Text + ".tif"
    .Error = ReadBallotMarks(.FPath, .MarkInfo(0))
    .Error = GetBallotMask("MASKS.TXT", .MarkInfo(0).CardRotID, .Mask(0))
    .Error = ReadBallotOvals(.FPath, .MarkInfo(0), .Mask(0), .Votes(0))
End With

Parameter Definitions

Filename provides the path and filename for the ASCII text file containing ballot oval and write-in masks.

CardRotID provides the card style number for which the oval mask is required.

Mask returns the oval mask in GEMS format.

The function returns a result code from the list below.

Listing 58. Result codes


<table>
<thead>
<tr>
<th>Result Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Successful</td>
<td>= 0;</td>
</tr>
<tr>
<td>BadData</td>
<td>= 30;</td>
</tr>
<tr>
<td>BadFile</td>
<td>= 40;</td>
</tr>
</tbody>
</table>

Tips and Tricks

The mask file must be formatted according to the following syntax rules:

- The data for each card number must begin with "@<CardRotID>" on a line by itself, where <CardRotID> is the card number.
- The "@<CardRotID>" line is followed by the oval mask, consisting of one or more lines with the syntax "<Col> <Row1> <Row2> <Step>", which is interpreted as follows: There is a group of ovals in column <Col>, starting at row <Row1> and ending at row <Row2> (inclusive), with an oval in every <Step>th row. Thus "1 7 21 2" means there is an oval in every second row of column 1, from row 7 to 21 inclusive.
- The oval mask is followed by an optional write-in mask, beginning with ".-WI" on a line by itself.
- The write-in mask specifies each write-in oval, with the syntax of either: (a) "<Index> <X1> <Y1> <X2> <Y2>", or (b) "<Index> 0". <Index> provides the oval's
location in the oval mask data. For example, <index> 3 refers to the 4th oval in
the oval mask. If syntax (a) is used, $X_1$ through $Y_2$ 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. $X_1$ and $X_2$ are always interpreted as
being to the right of the adjacent oval. $Y_1$ is always above the center of the oval,
and $Y_2$ is always below it. If syntax (b) is used, this indicates that the values
$X_1$ through $Y_2$ are the same as in the preceding write-in. Line breaks can
be inserted after any desired write-in specification.

- The data for each card number must end with either: (a) "@<CardRotID>" for the
next card number; or (b) the end of the file.

**Listing 59. Example of mask file for CardRotID 32**

```
@32
1 7 21 2
1 25 32 1
12 7 12 1
12 15 17 1
12 23 31 1
23 9 13 1
23 19 33 1
33 8 9 1
33 16 17 1
33 24 25 1
33 33 34 1
44 6 7 1
44 14 15 1
44 23 24 1
44 32 33 1
55 6 7 1
-WI
7 4 4 57 8
15 4 4 57 3
21 0 24 0 30 0 31 0 32 0 33 0
37 4 4 50 3
38 0 49 0 50 0 51 0 52 0 53 0
```

**See Also**

*ReadBallotOvals* function.
GetDLLVersion Function

Listing 60. Delphi API

```delphi
function GetDLLVersion : smallint; stdcall;
```

Listing 61. VB declaration and example

```vbnet
Private Declare Function GetDLLVersion Lib "CCDLL.DLL" () As Integer

Label1.Caption = "DLL Version " & Str(GetDLLVersion)
```

Parameter Descriptions
This function has no parameters, and returns the version number of the ACCSIPDLL.

Tips and Tricks
The least significant digit of the DLL version number is incremented when changes are made that do not affect compatibility with calling applications or file formats. When changes are made that do affect compatibility, the remaining digits are incremented and the least significant digit is reset to zero.

GetWriteInMask Function

This function is intended for use during software development, when communications with the GEMS database is not available. The function returns the write-in mask for the specified ballot style, in exactly the same format as GEMS. In this case, ballot oval and write-in masks are stored in an ASCII text file.

Listing 62.

```vbnet
function GetWriteInMask (Filename : pChar; { mask file path eg MASKS.TXT } CardRotID : smallint; { card rot ID # } var WIMask : tGemsData { GEMS style writein mask } ) : smallint; stdcall;
```
Listing 63. VB declaration and example

Private Declare Function GetBallotMask Lib "CCDLL.DLL" (ByVal Filename As String, ByVal CardRotID As Integer, ByRef WIMask As tGemsData) As Integer

with Ballot
  .FPath = "InTray" + tTiffName.Text + ".tif"
  .Error = ReadBallotMarks(.FPath, .MarkInfo(0))
  .Error = GetWriteInMask("MASKS.TXT", .MarkInfo(0).CardRotID, .WIMask(0))
  .Error = ReadWriteIns(.FPath, .MarkInfo(0), .WIMask(0), .Names(0))
End With

Parameter Definitions

**Filename** provides the path and filename for the ASCII text file containing ballot oval and write-in masks.

**CardRotID** provides the card style number for which the oval mask is required.

**WIMask** returns the write-in mask in GEMS format.

The function returns a result code from the list below.

Listing 64. Result codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful</td>
</tr>
<tr>
<td>30</td>
<td>BadData</td>
</tr>
<tr>
<td>40</td>
<td>BadFile</td>
</tr>
</tbody>
</table>

Tips and Tricks

For a description of the mask file syntax, see the `GetBallotMask` function.

See Also

`ReadWriteIns` function.

**ShowTiffFile Function**

This function displays a ballot image -- or any TIFF file --- on the screen. Function parameters control what scaling rule to use, and what angle to rotate the image before display, if any. If the `EditOK` parameter is set to true, various editing controls are provided for the user.
ShowTiffFile Function

Listing 65. Delphi API

```delphi
function ShowTiffFile(  
    Filename : pChar; { TIFF filename to display }  
    Page     : byte; { first page to display, 1-based }  
    Scale    : byte; { scaling method; see above }  
    Angle    : smallint; { rotate angle {0, 90, 180, 270} }  
    EditOK   : wordbool { allow editing? }  
) : smallint; stdcall; { result code }
```

Listing 66. VB declaration

```vbnet
Private Declare Function ShowTiffFile Lib "CCDLL.DLL" (  
    ByVal Filename As String, _  
    ByVal Page As Byte, _  
    ByVal Scal As Byte, _  
    ByVal Angle As Integer, _  
    ByVal EditOK As Boolean _  
) As Integer
```

Parameter Definitions

**Filename** is the path and filename for the TIFF file to be opened.

**Page** is the page number in the TIFF file to be displayed. With single-page TIFF files, only page 1 is valid.

**Scale** indicates the scaling method to be used to display the image. Valid values are: 0 = no scaling; 1 = fit width; 2 = fit height; and 3 = scale to fit (full display, no scroll bars).

**Angle** indicates the angle through which the image should be rotated before display. The angle must be a positive multiple of 90 degrees. Positive values rotate the image clockwise.

If the boolean parameter **EditOK** is true, the following editing buttons are made available to the user:

- **Deskew** straightens images rotated through small angles
- **Split** saves a multi-page TIFF image into multiple single-page TIFF files (thus making the images compatible with a wider range of graphics editors)
- **Append** adds the image in a single-page TIFF file to the existing pages in another TIFF file; this feature is used to recombine images that were divided with the Split function, then edited.
Chapter 5: Miscellaneous Functions

Result Codes

The `ShowTiffFile` function returns one of the following error codes:

**Listing 67. Result codes**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful = 0; function completed successfully</td>
</tr>
<tr>
<td>10</td>
<td>BadTIFF = 10; can't find TIFF file</td>
</tr>
<tr>
<td>20</td>
<td>BadPage = 20; can't read page from TIFF file</td>
</tr>
<tr>
<td>110</td>
<td>BadAngle = 110; invalid rotate angle</td>
</tr>
<tr>
<td>120</td>
<td>BadScale = 120; invalid scaling method</td>
</tr>
</tbody>
</table>
Ballot Processing Techniques

This chapter provides some background of the DLL’s image processing environment and the algorithms used.

**Image Processing Environment**

Ballot are intended to be loaded into the scanner in "portrait" mode: head first and face up. However, the system will automatically recognize ballots loaded in any of the four possible portrait orientations. Although the scanner would theoretically yield higher throughput by scanning 11" ballots in landscape mode, the automatic document feeder (ADF) cannot reliably handle previously folded ballots in this orientation (the leading edge of the ballot is no longer flat).

Even though ballots are scanned in portrait mode, scanned images are rotated 90 degrees and stored in landscape mode. This is done for technical reasons.* In the normal orientation, the top of the ballot is at the left side of the image file, as shown in the examples below.

---

* Horizontal scan lines can be accessed much faster than individual pixels in bitmap images. Since most of the image processing time is consumed by locating the timing marks down the left and right edges of a ballot, it makes sense for scan lines to pass through these marks. Thus, the portrait image must be rotated 90 degrees so that these marks lie along image’s horizontal axis.
Chapter 6: Ballot Processing Techniques

Figure 4. Scanned image of the front side of a ballot.
Of course, whenever a ballot is displayed on-screen for the user, it is rotated back into portrait mode. It is also inverted if it was scanned feet first, and/or flipped over if it was scanned upside down.

Scanned ballot images are stored in 2-page 1-bit (monochrome) 200 dpi CCITT G4 Fax compressed TIFF files. 1-bit 200 dpi images provide a suitable compromise between image quality, storage requirements and performance. CCITT G4 Fax compression yields the smallest files for this type of data, without resorting to the unpredictable image degradation of lossy compression schemes.

To avoid confusion in programming and elsewhere, all references to the dimensions and other properties of ballot images are named or described from a portrait perspective. (The only exception to this rule is that the width and height of the image file refer to the actual landscape dimensions.) See the diagram on the next page.
One consequence of the image rotation is that the (0, 0) pixel location is at the top right corner of the ballot, rather than the more usual top left. Thus, the pixel loca-
tion (X, Y) refers to the X'th pixel from the right side of the ballot and the Y'th pixel from the top of the ballot.

Ballot images are approximately 1700 pixels high (8.5"), and 2200, 2800, 3400 or 3600 pixels wide (corresponding to 11, 14, 17 or 18" ballots).

**Scanning Anomalies**

When a ballot is skewed slightly, its image dimensions increase proportionately. Skewing angles of up to 1.0 degrees are fairly common, but rarely exceeded.

The image processing routines were designed to accommodate skewing of up to 2.0 degrees. This corresponds to an offset of 0.63" over the length of an 18" ballot. In these situations, the offset is three times more than the width of a timing mark (0.188"), and no single scan line can pass through all timing marks along one edge. This fact had a considerable influence on the algorithms used.

Skewing can also change dynamically as a ballot passes through the scanner. This results in the left and right edges of the image having different lengths.

Deskewing routines are available in the scanner's video card software, but these were not used for three reasons. They cannot deal with dynamic skewing, they degrade the image significantly, and they are slow. Instead, image processing algorithms were developed that recognize from the beginning that the geometry of the timing mark array is basically an irregular 4-sided perimeter with non-linear sides.

Skewing is not the only scanner anomaly that the image processing routines must tolerate. The average image resolution can vary around its nominal value of 200 dpi. Furthermore, the velocity of the ballot through the scanner is not perfectly constant. Variations can result in timing marks being located up to 6 or 8 pixels from their expected vertical positions. This makes it essential to determine the exact vertical location of every timing mark on the left and right edges.

Finally, the image processing routines had to tolerate damaged ballots, where not all timing marks can be read. For example, one objective was to read ballots where a black felt pen had been used to obliterate most of the left and right edge marks.

**Image Processing Algorithms**

Image processing is involved in three fundamental areas:

- Determining the geometry of the timing marks along the left and right edges of each side of the ballot.
- Validating and decoding the timing marks in the top and bottom rows on each side of the ballot.
- Reading the votes marked in the ballot's ovals.
Chapter 6: Ballot Processing Techniques

The first two of these are the responsibility of the ReadBallotMarks function. The last is the responsibility of ReadBallotOvals.

Timing Mark Geometry

The algorithms used here are crucial, because achieving accuracy in all other aspects of image processing depends on a precise knowledge of ballot geometry.

The geometry of an image is determined by locating the timing marks along the left and right edges. These edges are used in preference to the top and bottom edges because:

- All marks are normally present. They do not contain binary coding.
- Velocity changes only affect these edges, so it is important to locate every readable one of these marks.
- The left and right marks are normally closer to the edges than the top and bottom marks, which reduces the number of scan lines that must be analyzed.

Left and right edge timing marks are located using the following method:

- Starting from the edge of the ballot and proceeding inwards, scan lines are searched for one or more timing marks. The 200 dpi image resolution is higher than necessary in the X direction, so only every third scan line is searched.
- Searching for marks in a scan line involves sliding a “window” along the column of pixels, and watching for peaks in a “score” that indicates the degree of match to a template pattern. When a peak exceeds a predefined threshold, a new timing mark has been found (a “hit”). The current pixel location identifies the Y location of the center of the mark, and the current scan line identifies the X location of the hit. The (X, Y) location of every hit is stored until the search is completed.
- Searching continues through as many as 120 scan lines. This is the number necessary to ensure that essentially all hits will be detected on an 18” ballot skewed by 2 degrees. Searching is terminated early as soon as the expected number of hits is detected, based on the ballot size.
- As each scan line is searched, the vertical location of its first and last timing mark is stored temporarily.

When searching is completed along one edge, the collected data is used to compute the (X, Y) pixel location of every left or right edge timing mark, as follows.

- The first mark locations are analyzed. Marks that are a long way from the top or bottom of the ballot (as can happen with skewed images) are discarded. Then the median value is calculated to provide the precise Y pixel position of the top mark.
- This step is repeated to provide the Y position of the last (bottom) mark.
The next step is to compute the linear regression equation that best fits the stored hit locations, using the least squares fit method. The regression line \( Y = A + BX \) passes precisely through the middle of the marks. Solving for \( X \), given the \( Y \) positions of the top and bottom marks, yields the \( X \) positions of the same marks. At this point, the centers of the four corner marks have been precisely determined. These are the \( X_{IT} \) through \( Y_{2B} \) dimensions described elsewhere (see diagram).

The \( Y \) positions of the top and bottom marks are now used to create equally-spaced “virtual buckets” for each mark along the edge. For example, with an 11" ballot, 41 buckets will be created. Each of the stored hit locations is then analyzed to determine whether it lies close to its theoretical bucket location. If so, its \( Y \) position is added to the relevant bucket.

When all hit locations have been processed, the average \( Y \) location is calculated for each bucket. This yields the precise \( Y \) position of each mark, thus making the system immune to variations in scanner speed.

When one or more marks are eradicated or torn off a ballot, the corresponding buckets will be empty. In these cases, the missing mark locations are calculated by a process of interpolation. This ensures that ballot will be readable even when many of the left and right edge marks are unreadable.

The template used to search for pattern matches in scan lines has several design requirements:

- Maximize the discrimination of marks against other elements, including white areas, black areas, transitions from white to black areas (and vice versa), random black and white pixels, and text.

- Provide opportunities for speeding up the searching process, since this accounts for most of the total time needed for all ballot image processing.

The window used is 75 pixels wide, covering somewhat more than two timing marks. See diagram below. The middle 25 pixels are ignored, so that the pattern contains roughly equal numbers of black and white pixels. This maximizes discrimination against white areas.
The window is symmetrical, which makes it possible to greatly enhance discrimination as follows: A particular pixel location only adds to the score of the current window position if (a) it has a match and (b) its mirror-image pixel also has a match.

The pseudo-code fragment below summarizes the total searching process:

Listing 68. Searching for marks in scan lines

```plaintext
for X:= 1 to 120 do begin
  GetScanLine(X-1);
  HiScore:= 0
  for Y:= 37 to (ImageWidth - 1 - 37) do begin
    Score:= 0
    for W:= 13 to 37 do
      if ((ScanLine[Y-W] = Black) = (W in [19..31])) and
         ((ScanLine[Y+W] = Black) = (W in [19..31])) then
        inc(Score)
    if Score > CutOff then begin
      { found a better fit }  
```
Reading Timing Marks

Once the ballot’s geometry has been calculated, the location of any timing mark can be determined with great accuracy. The percentage of black pixels in a timing mark’s rectangular area can be measured and used to determine whether a mark is absent or present.

As discussed elsewhere, absent marks only occur in the bottom row of timing marks, where ballot identification fields are encoded.

The black percentage is not biased in either direction, so a value of 50% is used as the threshold to classify a mark as absent or present.

Reading Votes in Ovals

As with timing marks, the ballot geometry allows ovals to be located precisely. However, classifying a mark as voted or not voted is a lot more complex, because of the variety of marks that are made by voters, and because of the variety of writing implements used.

The ink in the writing implement largely determines the contrast level of the marks. With the exception of very pale tones, all colors register as black.

The width of the line produced by the implement, and the extent of the marks made by the voter, largely determine how many black pixels are found in an oval.

Ovals on ballots are normally printed with black or red outlines, both of which are seen as black. If there are small errors in the ballot’s geometry measurements, parts of the oval will be seen in the area examined for voter marks. Thus "unmarked" ovals may not have a black percentage somewhat higher than zero.

It was determined by experiment that unmarked ovals always have a black percentage between 0 and 11 percent. Marked ovals have a black percentage between 20 and 100, with 20 representing a small "X" drawn lightly with a fine-tipped ball-point pen. Accordingly the following thresholds are used for classifying votes:
Chapter 6: Ballot Processing Techniques

- <14% black is classified as a NO vote
- >18% black is classified as a YES vote
- 14-18% black is classified as an UNDEFINED vote, requiring duping by the user.

These thresholds were selected to minimize the risk of under- or over-votes, and at the same time minimizing the number of ballots that require duping.

### Reading Write-In Votes

Write-in votes provide an area next to the oval for the voter to hand print the name of their preferred candidate. It is not practical to automate the reading of unconstrained hand printing, so on the few occasions when write-in votes are tabulated in detail, the user is required to interpret the write-ins.

To achieve this, the DLL function `ReadWriteIns` displays a magnified view of the area assigned to each write-in vote. Displaying the correct area again depends on knowing the precise geometry of the ballot.

For each write-in vote, the user reads the name, then types it in or selects it from a list of known names.
This Appendix describes the coding scheme used in the bottom row marks appearing on each side of a ballot.

Each side has a row of 34 marks along its bottom edge. Of these, the two outer marks are disregarded, and the remaining 32 have encoded information. The coding is different on the front and back of a ballot.

In the following descriptions, bits 0 and 31 refer to the least and most significant bits, respectively. The most significant bit is at the left side of the ballot.

Front Marks

Front marks are encoded as shown in the table below.

Table 4. Front mark coding scheme

<table>
<thead>
<tr>
<th>First Bit</th>
<th>Last Bit</th>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>Mod 4 check sum bits</td>
</tr>
<tr>
<td>2</td>
<td>14</td>
<td>13</td>
<td>Batch or precinct number</td>
</tr>
<tr>
<td>15</td>
<td>27</td>
<td>13</td>
<td>Card number (CardRotID)</td>
</tr>
<tr>
<td>28</td>
<td>30</td>
<td>3</td>
<td>Sequence number (always 0)</td>
</tr>
<tr>
<td>31</td>
<td>31</td>
<td>1</td>
<td>Start bit (always 1)</td>
</tr>
</tbody>
</table>

The mod 4 check sum bits are obtained by adding the number of 1’s in bits 2 through 31, then encoding the results of a mod 4 operation in bits 0 and 1. For example, if bits 2 through 31 have 18 1’s, bits 0 and 1 will hold the value 2 (18 mod 4 = 2).
Back Marks

Back marks are encoded as show in the table below.

Table 5. Back mark coding scheme

<table>
<thead>
<tr>
<th>First Bit</th>
<th>Last Bit</th>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>5</td>
<td>Election day of month (1..31)</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>4</td>
<td>Election month (1..12)</td>
</tr>
<tr>
<td>9</td>
<td>15</td>
<td>7</td>
<td>Election year (2 digits)</td>
</tr>
<tr>
<td>16</td>
<td>20</td>
<td>5</td>
<td>Election type (see below)</td>
</tr>
<tr>
<td>21</td>
<td>31</td>
<td>11</td>
<td>Ender code (binary 01110111110)</td>
</tr>
</tbody>
</table>

Election type is a number indicating the first letter of the election type, according to its position in the alphabet relative to the letter “A”. For example, a general election (“G”) has the election type code of 6, because “G” is six letter after “A” in the alphabet.

Ballot Validation

Timing marks are used to validate ballots in several different ways:

- The top row of timing marks must be all 1’s, on both side of the ballot.
- The bottom front marks must (a) start with a 1, (b) end with a valid check sum, and (c) contain valid card and batch/precinct numbers.
- The bottom back marks must (a) start with an ender code (binary 01111011110), and (b) contain the expected election type and date.
Appendix B

GEMS Protocol

This Appendix describes the communications protocol used when an AccuVote scanner or 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 in various DLL functions.

This chapter described 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, and subject to review and change.

Protocol Overview

• 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 (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 oval mask is transmitted in a compressed form.

• When the client has read the ovals on a ballot, the vote data is sent back to Gems in a compressed form.
If write-in votes are being tabulated, similar mask and vote data are exchanged with GEMS, using special messages with compressed data.

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

**Message Definitions**

**Listing 69. Data types and message IDs**

<table>
<thead>
<tr>
<th>Data Types</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYTE</td>
<td>1 byte</td>
</tr>
<tr>
<td>WORD</td>
<td>2 byte LE unsigned int</td>
</tr>
<tr>
<td>BYTE[n]</td>
<td>Array of n bytes</td>
</tr>
<tr>
<td>String</td>
<td>NULL terminated string</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Message IDs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MID_PING</td>
<td>0</td>
</tr>
<tr>
<td>MID_PONG</td>
<td>1</td>
</tr>
<tr>
<td>MID_MASK_REQUEST</td>
<td>2</td>
</tr>
<tr>
<td>MID_MASK_REPLY</td>
<td>3</td>
</tr>
<tr>
<td>MID_BALLOT_IMAGE</td>
<td>4</td>
</tr>
<tr>
<td>MID_BALLOT_DESTINATION</td>
<td>5</td>
</tr>
<tr>
<td>MID_PROTOCOL</td>
<td>6</td>
</tr>
<tr>
<td>MID_WRITEIN_REQUEST</td>
<td>7</td>
</tr>
<tr>
<td>MID_WRITEIN_REPLY</td>
<td>8</td>
</tr>
<tr>
<td>MID_WRITEIN_IMAGE</td>
<td>9</td>
</tr>
<tr>
<td>MID_WRITEIN_DESTINATION</td>
<td>10</td>
</tr>
</tbody>
</table>

**Listing 70. Message Structures**

**Protocol Version:**

<table>
<thead>
<tr>
<th>MID_PROTOCOL</th>
<th>BYTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>WORD</td>
</tr>
</tbody>
</table>

**Ballot Mask Request:**

<table>
<thead>
<tr>
<th>MID_MASK_REQUEST</th>
<th>BYTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CardRotId</td>
<td>WORD</td>
</tr>
<tr>
<td>ElectionId</td>
<td>WORD</td>
</tr>
</tbody>
</table>

**Ballot Mask Reply:**

<table>
<thead>
<tr>
<th>MID_MASK_REPLY</th>
<th>BYTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CardLength</td>
<td>BYTE</td>
</tr>
<tr>
<td>Found</td>
<td>BYTE</td>
</tr>
<tr>
<td>MaskSize</td>
<td>WORD</td>
</tr>
<tr>
<td>Mask or ErrorText</td>
<td>BYTE[Mask Size] if Found, or BYTE[32]</td>
</tr>
</tbody>
</table>

**Ballot Image:**

<table>
<thead>
<tr>
<th>MID_BALLOT_IMAGE</th>
<th>BYTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ButtonStatus</td>
<td>BYTE</td>
</tr>
<tr>
<td>CardLength</td>
<td>BYTE</td>
</tr>
</tbody>
</table>
Message Definitions

CardRotId WORD
ReportUnitId WORD
ElectionId WORD
ImageSize WORD
Image BYTE[Image Size]

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

Ballot Writein Mask Request:
MID_WRITEIN_REQUEST BYTE
CardRotId WORD
ElectionId WORD

Ballot Writein Mask Request:
MID_WRITEIN_REPLY BYTE
CardLength BYTE
Found BYTE
MaskSize WORD
Mask or ErrorText BYTE[Mask Size] if Found, or BYTE[32]

Ballot Writein Image
MID_WRITEIN_IMAGE BYTE
CardLength BYTE
CardRotId WORD
ReportUnitId WORD
ElectionId WORD
NamesSize WORD
Names BYTE[NamesSize]

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

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 same as ElecDate described on page 10.

CardLength is the number of timing mark rows required on the ballot (41, 53, 65 or 69).

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 a 32-character error string. **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.

**NameSize** indicates the number of bytes of data that must be read from the **Names** field.

### 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 71. C++ declaration of oval mask message**

```c
UCHAR cardlength (number of timing marks)
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:

```c
UCHAR type (row (1) or column (0))
UCHAR index (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 (in ASCII format)

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 position in
column 2 rows 4 to 11 and column 4 rows 4 to 8 and rows 21 to 24.
Position 0, 0 is the top left voting position (not the timing mark or diagnostic mark).

**Vote Data**

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

**Listing 72. C++ declarations for vote data message**

```
UCHAR size  (number of bytes in 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**

**WIMask** is compressed using the following method:

For each write-in position on the ballot:

Byte 1 = index into oval array for this vote
Byte 2 = X1
Byte 3 = Y1
Byte 4 = X2
Byte 5 = Y2

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.

**Write-in Names**

The write-in **Names** field is compressed using the following method:

For each write-in vote used by the voter:

Byte 1 = index into oval array for this vote
Byte 2 = number of characters in write-in candidate's name
The remaining bytes are the name itself.
Appendix C

Complete Data Definitions

The ACCSIPDLL API consists of data declarations and function calls.

Data Declarations

The listings below detail the data declarations actually used by the Delphi DLL, and an example of the equivalent declarations needed in a VB application using the DLL.

Listing 73. Delphi data declarations

```delphi
unit uCCDefs;

{ central count DLL data structures affecting DLL API or requiring
similar structure in main application's database }

interface

const

{ DLL version }
Version = 10;

{ product name }
ProdName = 'AccuVote Central Count System';

{ ballot status in workflow }
NeedsMarks = 0; { ready for marks analysis }
NeedsRepair = 1; { needs repair }
Discarded = 2; { discarded by user }
NeedsMask = 3; { ready for GEMS mask }
NeedsVotes = 4; { ready for reading ovals }
NeedsDuping = 5; { needs duping }
NeedsSubmit = 6; { ready for GEMS submission }
Accepted = 7; { votes accepted by GEMS }
Rejected = 8; { rejected by GEMS }

{ control card IDs }
Ender = 65535; { precinct ID = 0 }
Absentee = 65534; { ditto }
Test = 65533; { precinct ID = precinct to test }
PrecinctHeader = 65532; { override precinct ID }
GenericDeck = 65531; { precinct ID = 0 }
UserDeck = 65530; { precinct ID = batch # }
```
Appendix C: Complete Data Definitions

{ vote status }
No = 0;  { same as in GEMS }
Yes = 1;
Undef = 2;  { in between white & black }

{ ballot constraints }
MaxCols = 34;  { cols of marks }
MaxRows = 69;  { max rows of marks (18") }
MaxCandidates = 500;  { max # of candidates }
MaxWriteIns = 50;  { max #$ of writeins }

{ string and byte array lengths }
FNLen = 200;  { max file path length }
IDLen = 4;  { max batch/seqno ID lengths }
WriteInLen = 32;  { max write-in name length }
GemsBytes = 255;  { max bytes in GEMS mask or image }
GemsChars = 1000;  { max chars of GEMS writein names }

{ DLL error codes; some codes use LSD for details, e.g. side }
{ see Excel spreadsheet ErrorCodes1.xls }
{ generic; apply to multiple DLL functions }
Successful = 0;
BadTIFF = 10;  { can’t find TIFF file }
BadPage = 20;  { can’t read page from TIFF file }
BadData = 30;  { data causes failure }
BadFile = 40;  { can’t find other file }
NothingToDo = 70;  { ballot doesn’t need this function }
UserCancel = 80;  { user cancelled operation }
UserDiscard = 90;  { ballot discarded by user }

{ ReadTIFFFile function errors }
BadAngle = 120;  { invalid image rotate angle }
BadScale = 130;  { invalid scaling constant }

{ ReadBallotMarks function errors }
... geometry related }
BadWidth = 220;  { invalid image width }
BadHeight = 230;  { bad image height }
BadSkew = 240;  { excessive image rotation }
BadMarks = 250;  { can’t find enough marks }
BadGeomX = 260;  { bad geometry, horizontal }
BadGeomY = 270;  { bad geometry, vertical }

... marks analysis related }
BadTopMarks = 310;  { can’t find top marks }
BadEnderMarks = 320;  { can’t find ender marks 01111011110 }
BadStartBit = 330;  { can’t find start bit }
BadCheckSum = 340;  { check sum failed }

{ RepairBallot function errors }
{ ReadBallotOvals function errors }
Undefined = 510;  { ballot contains undefined ovals }

{ DupBallot function errors }
 TBD }

{ ReadWriteIns function errors }
 TBD }

type
{ compressed data exchanged via GEMS protocol }
tGemsBytes = array [0..GemsBytes-1] of { byte array }
  byte;
tGemsData = record
  Size : byte;
  Data : tGemsBytes;
end;

{ scan analysis data for each face of ballot (Top/Bottom, NOT Front/Back); this data for each side of the ballot is generated when timing marks are analyzed, and is needed later when ovals are read and write-ins are
Data Declarations • 63

processed

tyPos = array [0..MaxRows-1] of smallint;
tScanInfo = record
  { geometry for one side }
  ImageW : smallint; { width }
  ImageH : smallint; { height }
  Rows : byte; { rows, defined by ballot ht }
  Scans : smallint; { # of lines scanned (testing) }
  { X, Y coordinates of centers of 4 corner marks, in pixels }
  X1T, Y1T : smallint; { top left }
  X1B, Y1B : smallint; { bottom left }
  X2T, Y2T : smallint; { top right }
  X2B, Y2B : smallint; { bottom right }
  YPosL, { pos of all marks; L side }
  YPosR : tyPos; { ... R side }
end;

{ ballot timing marks decoding; this data for the ballot as a whole
is generated when timing marks are analyzed, and is needed later when
ovals are read and write-ins are processed; the 4 Elec... fields
are zero on control cards }
tMarkChars = array [0..MaxCols-3] of char;
tMarkInfo = record
  ScanInfo : array [0..1] of tScanInfo; { 0 = top, 1 = bottom }
  Flipped : wordbool; { scanned face down? }
  Inverted : wordbool; { scanned feet first? }
  (*
  Repaired : wordbool; { repaired by user? }
  Duped : wordbool; { duped by user? }
  *)
  BotMarks : array [0..1] of tMarkChars; { 0 = front, 1 = back }
  CardRotID : smallint; { card identifier }
  PrecinctID : smallint; { precinct ID # or batch # }
  ElecType : byte; { e.g. 6 => "G*eneral" }
  ElecDate : smallint; { election date }
end;

{ data for ovals; Row and Col come from GEMS; Vote comes from reading
ovals and duping; note Col uses the GEMS standard of using cols 0..31
on the front, and 32..63 on the back; in the context of timing marks,
these correspond to 0..33 on both sides }
tOvalRec = record
  Col : byte; { oval’s column }
  Row : byte; { oval’s row }
  Vote : byte; { vote status; see const above }
end;
tOval = array [0..MaxCandidates-1] { all ovals on a ballot }
of tOvalRec;

{ data for write-ins; Row, Col, X1..Y2 come from GEMS, see above for
Col numbering; X1..Y2 are relative to the center of the associated
oval; Name comes from OCR and user editing, and is returned to GEMS }
tName = array [0..WriteinLen] of char;
tWriteInRec = record
  Index : smallint; { index into Oval array }
  X1 : smallint; { left edge }
  Y1 : smallint; { top edge }
  X2 : smallint; { right edge }
64 • Appendix C: Complete Data Definitions

Y2 : smallint; { bottom edge }
Name : tName; { candidate name }
end;
tWriteIn = array [0..MaxWriteIns-1] { all writeins on a ballot }
of tWriteInRec;

{ compressed writein names data exchanged via GEMS protocol; each
name is a null-terminated string }
tGemsChars = array [0..GemsChars-1] of { char array }
char;
tGemsNames = record
Size : byte;
Data : tGemsChars;
end;

{ complete ballot record }
tFilePath = array [0..FNLen-1] of { file path + name }
char;
tIDField = array [0..IDLen-1] of { ballot batch/seq no }
char;
tBallot = record
FPath : tFilePath; { tiff file path + name }
{ unique ballot ID, combined for primary key }
Batch : tIDField; { batch # }
SeqNo : tIDField; { sequence # }
{ ballot status }
Status : byte; { workflow status; see
constants earlier }
Error : smallint; { latest error code; see
constants earlier }
Repaired : wordbool; { repaired by user? }
Duped : wordbool; { duped by user? }
UserWriteIn : wordbool; { write-ins processed? }
{ data from timing mark analysis }
MarkInfo : tMarkInfo; { mark decoding results }
{ oval data from GEMS and from reading and
duping ovals }
Mask : tGemsData; { oval positions, GEMS format }
Votes : tGemsData; { votes, in GEMS format }
{ writein mask from GEMS, and candidate names }
WIMask : tGemsData; { writein specs, in GEMS format }
WINames : tGemsNames; { writein names }
end;

implementation
end.

Listing 74. Example of equivalent VB declarations

' Note: all arrays have zero-based indexing; array size constants
' must match values in uCCDefs (minus 1)
'
' *************** CONSTANT DECLARATIONS ***************
'
ballon statuses
Const NeedsMarks = 0
Const NeedsRepair = 1
Const Discarded = 2
Const NeedsMask = 3
Const NeedsVotes = 4
Const NeedsDuping = 5
Const NeedsSubmit = 6
Const Accepted = 7
Const Rejected = 8

' max rows of timing marks - 1
Const MaxRows = 68

' max bytes in GEMS data array - 1
Const GemsBytes = 254

' max ovals in a ballot - 1
Const MaxCandidates = 499

' max write-in voting position on a ballot - 1
Const MaxWriteIns = 49

' ******************* TYPE DECLARATIONS *******************

' byte array for exchanging mask/vote data with GEMS
Private Type tGemsData
    Size As Byte
    Data(GemsBytes) As Byte
End Type

' char array for exchanging write-in names with GEMS
Private Type tGemsNames
    Size As Byte
    Data As String * 1000
End Type

' geometry of one side of a ballot
Private Type tScanInfo
    ImageW As Integer
    ImageH As Integer
    Rows As Byte
    Scans As Integer
    X1T As Integer
    Y1T As Integer
    X1B As Integer
    Y1B As Integer
    X2T As Integer
    Y2T As Integer
    X2B As Integer
    Y2B As Integer
    YPosL(MaxRows) As Integer
    YPosR(MaxRows) As Integer
End Type

' results of timing mark analysis
Private Type tMarkInfo
    ScanInfo(1) As tScanInfo
    Flipped As Boolean
    Inverted As Boolean
    MarksFront As String * 32
    MarksBack As String * 32
    CardRotID As Integer
    PrecinctID As Integer
    ElecType As Byte
    ElecDate As Integer
End Type
Appendix C: Complete Data Definitions

' for testing, provides easily viewed array of masks and votes
Private Type tOval
    Col As Byte
    Row As Byte
    Vote As Byte
End Type

' for testing, provides easily viewed array of write-in votes
Private Type tWriteIn
    Index As Integer
    X1 As Integer
    Y1 As Integer
    X2 As Integer
    Y2 As Integer
    Name As String * 32
End Type

' example of ballot record encapsulating all ballot info
Private Type tBallot
    FPath As String * 200
    Batch As String * 4
    SeqNo As String * 4
    Status As Byte
    Error As Integer
    Repaired As Boolean
    Duped As Boolean
    UserWriteIn As Boolean
    MarkInfo As tMarkInfo
    Mask As tGemsData
    Votes As tGemsData
    WIMask As tGemsData
    WINames As tGemsNames
End Type

' ******************* VARIABLE DECLARATIONS *******************
Dim Status As Byte
Dim Error As Integer
Dim Repaired As Boolean
Dim Duped As Boolean
Dim UserWriteIn As Boolean
Dim MarkInfo(0) As tMarkInfo
Dim Mask(0) As tGemsData
Dim Votes(0) As tGemsData
Dim Ovals(MaxCandidates) As tOval
Dim WIMask(0) As tGemsData
Dim WINames(0) As tGemsNames
Dim WriteIn(MaxWriteIns) As tWriteIn
Dim Ballot As tBallot
Apppendix D

Complete Function Calls

This appendix provides a complete listing of the Delphi API of the ACCSIPDLL, and the equivalent VB declarations.

Listing 75. Delphi DLL functions

```delphi
function GetDLLVersion : smallint; stdcall;

function GemsDateToYYMMDD (GemsDate : smallint); stdcall; { election date in marks format } { same in YYMMDD format }

function YYMMDDToGemsDate (YYMMDD : longint); stdcall; { election date in YYMMDD format } { same in marks format }

function GemsMaskToOvals (var Mask : tGemsData; var Oval : tOval); stdcall; { GEMS mask to decode into Oval } { decode results } { result code }

function OvalsToGemsMask (var Oval : tOval; var Mask : tGemsData); stdcall; { decode results } { GEMS mask to decode into Oval } { result code }

function OvalsToGemsVotes (var Ovals : tOval; var Votes : tGemsData); stdcall; { oval data to encode } { vote image data for GEMS } { result code }

function GemsVotesToOvals (var Votes : tGemsData; var Ovals : tOval); stdcall; { vote image data for GEMS } { oval data to encode } { result code }

{ ballot write-ins and names }
function GemsWIMaskToWriteins (var WIMask : tGemsData; var Writein : tWritein); stdcall; { writein mask, GEMS format } { writein data, local format }

function WriteinsToGemsNames (var Writein : tWritein);
```
function GetBallotMask {
    Filename : pChar;   { mask text filepath, eg MASKS.TXT }
    CardRotID : longint;   { card rot ID # }
    var Mask : tGemsData   { GEMS style mask }
} : smallint; stdcall;

function GetWriteinMask {
    Filename : pChar;   { mask file path eg MASKS.TXT }
    CardRotID : smallint;   { card rot ID # }
    var WIMask : tGemsData   { GEMS style writein mask }
} : smallint; stdcall;

function ShowTiffFile{
    Filename : pChar;   { TIFF filename to display }
    Page : byte;    { first page to display, 1-based }
    Scale : byte;    { scaling method; see above }
    Angle : smallint;   { rotate angle [0, 90, 180, 270] }
    EditOK : wordbool   { allow editing? }
} : smallint; stdcall;

function ReadBallotMarks {
    Filename : pChar;   { 2-page TIFF filename }
    var MarkInfo : tMarkInfo   { results of mark analysis }
} : smallint; stdcall;

function RepairBallot{
    Filename : pChar;   { 2-page TIFF filename }
    ElecType : byte;   { election type, eg G6 = "G" }
    ElecDate : smallint;   { election date, in GEMS format }
    var MarkInfo : tMarkInfo   { results of IDing ballot }
} : smallint; stdcall;

function ReadBallotOvals {
    Filename : pChar;   { 2-page TIFF filename }
    var MarkInfo : tMarkInfo;   { results of IDing ballot }
    var Mask : tGemsData;   { GEMS mask of voting positions }
    var Votes : tGemsData   { image of votes for GEMS }
} : smallint; stdcall;

function DupBallot {
    Filename : pChar;   { 2-page TIFF filename }
    var MarkInfo : tMarkInfo;   { results of IDing ballot }
    var Mask : tGemsData;   { GEMS mask of voting positions }
    var Votes : tGemsData   { image of votes for GEMS }
} : smallint; stdcall;

function ReadWriteIns {
    Filename : pChar;   { 2-page TIFF filename }
    NamesFile : pChar;   { list of write-in candidates }
    var MarkInfo : tMarkInfo;   { results of IDing ballot }
    var Mask : tGemsData;   { ovals mask for card number }
    var Votes : tGemsData;   { votes made on ballot }
    var WIMask : tGemsData;   { writein mask, GEMS format }
    var Names : tGemsNames   { writein names, GEMS format }
} : smallint; stdcall;
Appendix D: Complete Function Calls

Listing 76. Equivalent VB declarations

' ******************** CCDLL FUNCTION DECLARATIONS ******************
'
' declarations for all CCDLL functions; see Delphi CCDLL.DPR file;
' GetBallotMask and GetWriteinMask are for testing only, with San Luis card rot ID 32;
' normally use GEMS database instead

Private Declare Function GetDLLVersion Lib "CCDLL.DLL" () As Integer
Private Declare Function GemsDateToYYMMDD Lib "CCDLL.DLL" (ByVal GemsDate As Integer) As Long
Private Declare Function YYMMDDToGemsDate Lib "CCDLL.DLL" (ByVal YYMMDD As Long) As Integer
Private Declare Function GemsMaskToOvals Lib "CCDLL.DLL" (ByRef Mask As tGemsData, ByRef Ovals As tOval) As Integer
Private Declare Function OvalsToGemsMask Lib "CCDLL.DLL" (ByRef Ovals As tGemsData, ByRef Mask As tOval) As Integer
Private Declare Function OvalsToGemsVotes Lib "CCDLL.DLL" (ByRef Ovals As tGemsData, ByRef Votes As tOval) As Integer
Private Declare Function GemsVotesToOvals Lib "CCDLL.DLL" (ByRef Votes As tGemsData, ByRef Ovals As tOval) As Integer
Private Declare Function GemsWIMaskToWriteIns Lib "CCDLL.DLL" (ByRef WIMask As tGemsData, ByRef WriteIn As tWriteIn) As Integer
Private Declare Function WriteInsToGemsNames Lib "CCDLL.DLL" (ByRef WriteIn As tWriteIn, ByRef Names As tGemsNames) As Integer
Private Declare Function GemsNamesToWriteIns Lib "CCDLL.DLL" (ByRef Names As tGemsNames, ByRef WriteIn As tWriteIn) As Integer
Private Declare Function GetBallotMask Lib "CCDLL.DLL" (ByVal Filename As String, ByVal CardRotID As Long, ByRef Mask As tGemsData) As Integer
Private Declare Function GetWriteInMask Lib "CCDLL.DLL" (ByVal Filename As String, ByVal CardRotID As Long, ByRef WIMask As tGemsData) As Integer
Private Declare Function ShowTiffFile Lib "CCDLL.DLL" (ByVal Filename As String, ByVal Page As Byte, ByVal Scal As Byte, ByVal Angle As Integer, ByVal EditOK As Boolean) As Integer
Private Declare Function ReadBallotMarks Lib "CCDLL.DLL" (ByVal Filename As String, ByRef MarkInfo As tMarkInfo) As Integer
Private Declare Function RepairBallot Lib "CCDLL.DLL" (ByVal Filename As String, ByVal CardRotID As Integer, ByRef MarkInfo As tMarkInfo) As Integer
Private Declare Function ReadBallotOvals Lib "CCDLL.DLL" (ByVal Filename As String, ByRef MarkInfo As tMarkInfo, ByRef Mask As tGemsData, ByRef Votes As tGemsData) As Integer
Private Declare Function DupBallot Lib "CCDLL.DLL" (ByVal Filename As String, ByVal MarkInfo As tMarkInfo, ByVal Mask As tGemsData, ByVal Votes As tGemsData) As Integer
Private Declare Function ReadWriteIns Lib "CCDLL.DLL" (ByVal Filename As String, ByVal NamesFile As String, ByVal MarkInfo As tMarkInfo, ByVal Mask As tGemsData, ByVal Votes As tGemsData, ByVal Names As tGemsNames) As Integer
To maximize the compatibility of a DLL with calling applications developed with different programming languages and tools, function parameters and return values should use only widely supported data types. The safest approach is to use only data types used in the Win32 API, since all languages support these.

Some languages create obstacles that hinder interfacing to even well-designed DLLs. Visual Basic is a good example of this. For example, VB has an unusual approach to storing strings, arrays and user-defined data types in memory. There is overhead associated with these data types, and pointers to these types refer to the overhead area, not to the underlying data.

The design of ACCSIPDLL presents special challenges, because complex user-defined data types must be passed to and from the DLL functions.

These considerations led to adopting the following rules in designing the DLL API, and in calling the functions from VB.

**Note:** These descriptions assume that all arrays have their first elements numbered zero.

### Numeric Data Types

The table below lists the numeric data types common to Delphi and VB. These can be passed by value or by reference to DLL parameters.
Table 6. Equivalent data types in Delphi, VB

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Delphi</th>
<th>VB</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-byte unsigned integer</td>
<td>byte</td>
<td>byte</td>
</tr>
<tr>
<td>2-byte boolean</td>
<td>wordbool</td>
<td>boolean</td>
</tr>
<tr>
<td>2-byte signed integer</td>
<td>smallint</td>
<td>integer</td>
</tr>
<tr>
<td>4-byte signed integer</td>
<td>longint</td>
<td>long</td>
</tr>
<tr>
<td>4-byte real number</td>
<td>single</td>
<td>single</td>
</tr>
<tr>
<td>8-byte real number</td>
<td>double</td>
<td>double</td>
</tr>
<tr>
<td>8-byte currency</td>
<td>currency</td>
<td>currency</td>
</tr>
</tbody>
</table>

### Strings

String parameters are passed by value, using the Delphi null-terminated string type `pChar` and VB’s compatible variable length `String`.

### Arrays

Simple one-dimensional arrays are passed by reference. In Delphi and VB, they are declared in the normal way. However, VB must pass a reference to the first array element, not to the array itself. Thus:

Delphi’s `X : array [0..99] of byte`

is equivalent to

VB’s `X(99)` as `Byte`, which is passed by reference to `X(0)`.

### User-Defined Record Types

Several rules apply to complex user-defined types (records):

- In Delphi, a string field is declared as a fixed-length array of char. In VB, a string field is declared as a fixed length string. Thus Delphi’s `Field : array [0..9] of char` is equivalent to VB’s `Field As String * 10`.

- In VB, a single instance of a record structure must be declared as a single-element array. This is passed to the DLL by reference to element zero, not to the array itself.

- In VB, a multi-element array of records is also passed by reference to the first element of the array, not to the array itself. Of course, any single element of the
array can be passed by reference to the particular element.

Many examples of these rules can be found in the VB code samples elsewhere in this Guide.
Using the four primary DLL functions in a Central Count application will involve a workflow similar to that represented by the pseudocode below. (Calls to DLL functions are indicated in **boldface**.)

**Listing 77.**

```vbnet
ReadBallotMarks
if Successful then begin
    GetBallotMaskFromGEMS
    ReadBallotOvals
    if not Successful then DupBallot
end
else begin
    RepairBallot
    GetBallotMaskFromGEMS
    DupBallot
end
SendVotesToGEMS
if Required then begin
    GetWriteinMaskFromGems
    ReadWriteIns
    SendWriteInsToGems
end
```

This chapter presents a sample VB program that exercises most of the DLL functions. The program can be found in the files VB**.

The sample VB program has the single form shown below.
Choosing the Show Tiff Image button uses the DLL's ShowTiffFile function to display the selected TIFF file on-screen.

Choosing the Test All button exercises most of the primary DLL functions. The selected ballot will be displayed for repair or duping, if appropriate.

The complete listing of this program appears below. The program uses the declarations described earlier.

Listing 78.

```vbnet
' ******************* LOW LEVEL ROUTINES *******************
Private Sub AddStatus(S As String)
    ' add line S to text control Text1
    With Text1
        .Text = .Text + S + Chr(13) + Chr(10)
    End With
End Sub

' ******************* FORM EVENT HANDLERS *******************
Private Sub cbTestAll_Click()
    ' test all (most) DLL functions
    Dim FName As String
    '
        ' initialize variables
        Status = NeedsMarks
        Error = Successful
        Repaired = False
        Duped = False
        UserWriteIn = False
        ' build TIFF filename
        FName = "InTray\+ tTiffName.Text + ".TIF"
        ' initialize status display
        Text1.Text = 
        AddStatus "Processing file"
        AddStatus FName
        ' get geometry & read timing marks
        AddStatus "Reading ballot marks"
        Error = ReadBallotMarks(FName, MarkInfo(0))
        AddStatus "Result code = " & Str(Error)
        ' update status
        If Error = 0 Then
```
Status = NeedsMask
ElseIf Error < 200 Then
  Status = Rejected
Else
  Status = NeedsRepair
End If

' need to repair ballot?
If Status = NeedsRepair Then
  AddStatus "Repairing ballot"
  ' note 6 = elec type "G", 359 = gems elec date for 001107
  Error = RepairBallot(FName, 6, 359, MarkInfo(0))
  AddStatus "Result code = " & Str(Error)
  ' update status
  If Error = 0 Then
    Repaired = True
    Status = NeedsMask
  ElseIf Error = 90 Then
    Status = Discarded
  Else
    Status = Rejected
  End If
End If

' need to get oval mask?
If Status = NeedsMask Then
  ' get GEMS mask from MASKS.TXT file
  AddStatus "Getting ballot mask " & Str(MarkInfo(0).CardRotID)
  Error = GetBallotMask("MASKS.TXT", MarkInfo(0).CardRotID, Mask(0))
  AddStatus "Result code = " & Str(Error)
  ' make mask available for easy viewing in Ovals
  Error = GemsMaskToOvals(Mask(0), Ovals(0))
  ' update status
  If Error = 0 Then
    If Status = Repaired Then
      Status = NeedsDuping
    Else
      Status = NeedsVotes
    End If
  Else
    Status = Rejected
  End If
End If

' read ballot ovals?
If Status = NeedsVotes Then
  AddStatus "Reading ballot ovals"
  Error = ReadBallotOvals(FName, MarkInfo(0), Mask(0), Votes(0))
  ' make votes available for easy viewing in Ovals
  Error = GemsVotesToOvals(Votes(0), Ovals(0))
  AddStatus "Result code = " & Str(Error)
  ' update status
  If Error = 0 Then
    Status = Accepted
  ElseIf Error = 510 Then
    Status = NeedsDuping
  Else
    Status = Rejected
  End If
End If

' ballot needs duping?
If Status = NeedsDuping Then
Appendix F: Sample VB Application

AddStatus "Duping ballot"
Error = DupBallot(FName, MarkInfo(0), Mask(0), Votes(0))
' make votes available for easy viewing in Ovals
Error = GemsVotesToOvals(Votes(0), Ovals(0))
AddStatus "Result code = " & Str(Error)
' update status
If Error = 0 Then
    Status = Accepted
    Duped = True
ElseIf Error = 90 Then
    Status = Discarded
Else
    Status = Rejected
End If
End If

' get write-in mask?
If Status = Accepted Then
AddStatus "Getting write-in mask"
Error = GetWriteInMask("MASKS.TXT", MarkInfo(0).CardRotID, WIMask(0))
AddStatus "Result code = " & Str(Error)
' update status
If Error = 0 Then
    Status = Accepted
Else
    Status = Rejected
End If
End If

' process write-ins?
If Status = Accepted Then
AddStatus "Processing write-ins"
Error = ReadWriteIns(FName, "VB_NAMES.TXT", MarkInfo(0), Mask(0),
    Votes(0), WIMask(0), WINames(0))
AddStatus "Result code = " & Str(Error)
' update status
If Error = 0 Then
    Status = Accepted
    UserWriteIn = True
Else
    Status = Rejected
End If
End If
End Sub

Private Sub cShowTiff_Click()
Dim FName As String
' build TIFF filename
FName = "InTray\tTiffName.Text + ".TIF"
' display DLL version
lVersion.Caption = "DLL Ver " & Str(GetDLLVersion)
' display TIFF file page #1, no scaling, upright (90 deg), no editing
Error = ShowTiffFile(FName, 1, 0, 90, True)
End Sub

Private Sub Form_Load()
' set current drive to D:
ChDrive "D"
' display DLL version
lVersion.Caption = "DLL Ver " & Str(GetDLLVersion)
End Sub
Appendix G

Glossary

This Glossary is provided to reduce confusion over terms that have historically been used to mean two different things, such as “Image” and “Side”. It is also intended as an aid to readers unfamiliar with the jargon of election systems.

**Back**  The side of a ballot where voting information ends.

**Ballot**  A printed duplex voting form, where the voter fills in ovals to indicate voting intent.

**Batch**  A virtual set of ballots, usually corresponding to one or more physical boxes of allots, and identified by starter and ender cards.

**Bottom**  The face-down surface of a sheet during scanning, not necessarily the front or back.

**Card**  A card is a ballot or other sheet with timing marks. See table on page 10.

**Card number (CardRotID)**  The number that uniquely identifies the arrangement of ovals on a ballot.

**Column**  Each timing mark on the top and bottom edges of a ballot identifies a vertical column of locations where ovals might be printed. Column 0 is the first usable column, excluding the left column of timing marks. There are 34 columns on all ballots (including the left and right columns of timing marks).

**Duping**  The process of determining the voters intent by visually examining questionable votes on a ballot. AccuVote Central Count System provides on-screen duping instead of earlier methods where duping required marking up a clean ballot with the same card number.

**Edge**  The left, right, top or bottom perimeter of a ballot, containing a row of timing marks.

**Front**  The side of a ballot where voting information begins.

**GEMS**  The election database that provides oval and write-in masks for each ballot (based on its card number), and which stores and tabulates the voting results.

**Image**  The digital picture of a ballot, stored in a disk file and/or displayed on-
screen.

**Mark**  See Timing Mark.

**Mask**  The data providing the (Row, Col) position of each oval on a ballot, based on its card number. Masks are retrieved from the GEMS database, in a compressed format to minimize transmission time.

**Orientation**  The manner in which a ballot is fed into the scanner, by default front side up and head first.

**Oval**  The area, often outlined in red, marked by the voter to indicate a Yes vote for a particular candidate or measure. Each oval is precisely located at the intersection of its row and column locations.

**Page**  The image of one side of a ballot. A ballot’s images are stored in a 2-page disk file, with the first page containing the top of the scanned sheet (not necessarily the front).

**Pixel**  The basic element of a ballot image, representing the presence or absence of one black dot.

**Repair**  The process of visually examining an unreadable ballot, to determine its orientation and card number.

**Row**  Each timing mark on the left and right edges of a ballot identifies a row of locations where ovals might be printed. Row 0 is the first usable row, excluding the top row of timing marks. There are 41, 53, 65 and 69 rows of timing marks on 11, 14, 17 and 18” ballots respectively (including the top and bottom rows of timing marks).

**Scan Line**  One column of pixels, extending from the top to the bottom of a ballot image. Image processing involves the detailed analysis of scan lines.

**Side**  The front or back of a ballot.

**TIFF**  A standard for storing images in disk files. Many different TIFF formats exist. AccuVote Central Count System stores each ballot in a 2-page 1-bit (monochrome) 200 dpi CCITT G4 Fax compressed TIFF file. This format provides a suitable trade-off between image quality, and storage requirements and image processing performance.

**Timing Mark**  Timing marks are printed around all four edges of both sides of a ballot. Each mark is a rectangle 3/16” wide by 1/16” high.

**Top**  The face-up surface of a sheet during scanning, not necessarily the front or back.

**Undefined Vote**  See Vote below.

**Vote**  A vote is the result of examining the marks in an oval, to classify it as a No, Yes, or Undefined vote. Undefined votes are too black to be No votes, but not black enough to be Yes votes. They are resolved by duping.

**Write-in**  An oval where the voter hand prints the name of the candidate of choice,
instead of selecting one of the official candidates.
A
absentee ballot, 10
acceleration, 47
algorithms, 43
API, 61, 67
array, 70, 71
arrays, 1, 16
automatic document feeder, 43

B
back marks, 54
ballot
absentee, 10
damaged, 47, 49
diagram, 45
dimensions, 45
display, 40
geometry, 17, 20, 48
image, 44, 45
polling place, 10
torn, 49
validation, 54
batch, 55
number, 53
starter card, 10
buffer
mask, 55

C
candidate list, 24
card
ender, 10
number, 53
card number, 19
CardRotID, 10, 19, 53
card types, 10
check sum, 53
coding
timing mark, 53
colors
vote, 51
compression
GEMS, 58
image, 45
vote, 59
conversion
data, 27
date, 27, 35
oval mask, 28, 32
votes, 30, 33
write-in mask, 31, 34
write-in names, 29

D
damaged ballot, 47
database
ballots, 4
data conversion, 27
data definitions, 4, 61
data type
arrays, 71
numeric, 70
strings, 71
user-defined, 71
date
conversion, 27, 35
Delphi, 1
diagram
ballot, 45
dimensions
ballot, 45
display
ballot, 40
DLL
calling from VB, 70
version, 39
DLL API, 67
document feeder, 43
DupBallot, 21
Duped, 10
duping, 20, 21, 52

E
ElecDate, 10
election
date, 54
type, 54
ElecType, 10
encoding
timing mark, 53

derender card, 10, 55
code, 54
error
   updating, 6
error sources, 2

F
files, 3
   front
      mark, 55
   front marks, 53
   function calls, 67
   functions
      miscellaneous, 36

G
GEMS
   compression, 58
date, 35
   errors, 58
   mask, 32
   messages, 56
   protocol, 55
   votes, 30
      write-in mask, 34
      write-in names, 29
GemsDateToYYMMDD, 27
GEMS mask, 28
GemsMaskToOvals, 28
GemsNamesToWriteIns, 29
GemsVotesToOvals, 30
GemsWIMaskToWriteIns, 31
geometry, 17, 20, 48, 51
GetBallotMask, 36
GetDLLVersion, 39
GetWriteInMask, 39
glossary, 77

I
image
   size, 13
image file
   format, 45
image processing, 16, 43, 47
ink, 51
installation, 3

L
   least squares fit, 49
   linear regression, 49
   line width, 51

M
magnifier, 24, 52
mark, 48
   back, 54
   decoding, 17
   encoding, 19, 53
   front, 54, 55
   locating, 48
   missing, 49
   reading, 51
marks
   encoding, 9
   mask
      conversion, 28
      oval, 11, 36, 55, 58
      write-in, 31, 39, 59
mask file
   format, 37
median, 48
message
   GEMS protocol, 56
   MID_BALLOT_DESTINATION, 57
   MID_BALLOT_IMAGE, 56
   MID_MASK_REPLY, 56
   MID_MASK_REQUEST, 56
   MID_PROTOCOL, 56
   MID_WRITEIN_DESTINATION, 57
   MID_WRITEIN_IMAGE, 57
   MID_WRITEIN_REPLY, 57
   MID_WRITEIN_REQUEST, 57

N
name
   write-in, 59
numeric data type, 70

O
Object Pascal, 1
   orientation, 45
   scanning, 43
origin
   image file, 46
oval, 28, 30, 32, 33
   mask, 11, 36, 55, 58
Index

outline, 51
oval mask
  conversion, 28, 32
OvalsToGemsMask, 32
OvalsToGemsVotes, 33
over-vote, 52

P
percent black, 51
ping, 55
port number, 55
precinct
  header, 10
  number, 53
PrecinctID, 10
protocol
  GEMS, 55

R
ReadBallotMarks, 17
ReadBallotOvals, 20
reading
  write-in vote, 52
ReadWriteIn, 23
record structures, 71
RepairBallot, 18
Repaired, 10
result codes, 2
  interpretation, 2
  sources, 2
row, 57
rows, 13

S
sample application, 73
scan line, 43, 48
scan lines, 13
scanning
  orientation, 43
Scans, 13
sequence number, 53
ShowTiffFile, 40
skew, 47
start bit, 53
start card, 55
starter card, 10
status
  updating, 6
strings, 71

T
tBallot, 4
TCP/IP, 55
techniques, 43
template, 48, 49
tGemsData, 6
tGemsNames, 7
threshold
  mark, 51
  vote, 51
TIFF file
display, 40
TIFF format, 45
timing mark, 48
  back, 54
  encoding, 53
  front, 53
  locating, 48
  missing, 49
  reading, 51
tMarkInfo, 8
torn ballot, 49
tOval, 28
tOvalRec, 11
tScanInfo, 12
tWriteIn, 14

U
undefined vote, 20, 22
under-vote, 52
user deck, 10
user-defined data type, 71
UserWriteIn, 6

V
validation
  ballot, 54
VB
  calling DLLs, 70
  sample application, 73
velocity, 48
  changes, 47
version
  DLL, 39
vote, 33
  classification, 12, 20, 51
  compression, 59
  conversion, 33
  display, 22
threshold, 51
undefined, 22
write-in, 23
voters
conversion, 30

W
write-in
  dimensions, 15
  mask, 31, 34, 39, 59
  names, 15, 29, 59
  reading, 52
  vote, 31
write-in mask
  conversion, 31
write-in names
  conversion, 29
WriteInsToGemsWIMask, 34
write-in vote, 23
writing implement, 51

Y
YPosL, 13
YPosR, 13
YYMDDToGemsDate, 35