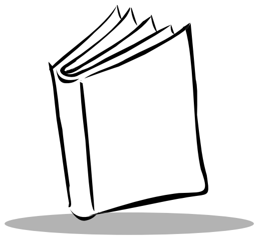
MSR 3000

System Software Manual

MSR 3000 System Software Manual

72-38411-01
Revision .1 — July 1999

Symbol Technologies, Inc. One Symbol Plaza, Holtsville N.Y. 11742



About This Guide

Introduction

The MSR 3000 System Software Manual provides information for use in developing applications to enable magnetic stripe reading on the Symbol Technologies SPT 1740 Terminal.

The MSR 3000 is an external Magnetic Card Reader for the Symbol SPT 1740 Terminal. The Software Development System enables application software developers to easily use and control all the basic and advanced functions of the MSR 3000.

The Software Development System consists of two parts:

Configurator:	Provides an easy-to-use graphical user interface (GUI) for selecting features and setting up the MSR 3000. It runs under Windows 95, 98 and Windows NT 4.0 or above.
MSR Manager Shared Library	A "C" library of all the functions to use and control the MSR 3000. The parameters of the library functions can be generated by the MSR 3000 Configurator, or defined directly by the developer.

This document describes both of these tools in detail.

This document assumes that you are familiar with the CodeWarrior™ for Palm OS development environment.

Chapter Descriptions

Chapter 1, *Using The MSR Manager Shared Library*, describes the use of the API, which enables applications on the SPT 1740 to control and receive data from the MSR 3000.

Chapter 2, *MSR Commands*, describes each API command in detail.

Chapter 3, *MSR 3000 Configurator*, describes the use of the Configurator, a windows-based tool which enables the developer to easily set up the MSR for use with the API.

Chapter 4, *Using the Configurator to Set the MSR 3000*, describes how to use the Configurator tool to generate a header file and combine the header file with the application running on the SPT 1700 Series terminal.

Chapter 5, *A Simple Application Program Sample*, provides an easy to follow application program to use as a reference in your application development.

Appendix A, *Data Editing Overview*, describes the data editing feature of the API, which allows you to edit the data which has been read from a magnetic card before sending it to the application.

Appendix B, *Common Magnetic Card Encoding Formats* provides a listing of the commonly used magnetic card formats for use in your application development.

MSR 3000 Features

All features of the MSR 3000 can be configured with the configurator tool or directly via the shared library interface. The Configurator creates a CodeWarrior include file for the selected MSR 3000 settings, and the application developer can then use the include file and call MsrSendSet() function to set up MSR 3000. See Chapter 3, *MSR 3000 Configurator* for details.

Buffer Mode

Two Buffer Modes are supported on the MSR 3000:

♦ Unbuffered Mode	MSR 3000 sends data as soon as the data is available. When using the unbuffered mode, the application program needs to be ready to receive data.
♦ Buffered Mode	The application program first sends an “Arm to Read” command to enable the magstripe reading. The user swipes a card, the decoded data is stored in the MSR 3000 data buffer and the MSR 3000 is disarmed. The application program then sends a “Get Track**x**” command to retrieve the data from the buffer. All setting functions names MsrSetXxx disarm the current read. An application must issue an “Arm to Read again before swiping a card.

If an application is designed to control the card swipe and card data read by itself, then the application developer should use buffered mode. Otherwise, unbuffered mode is recommended.

Terminator/Pre-amble/Post-amble

If Data Edit is disabled, simple message formatting can be done using the Terminator, the Pre-amble and the Post-amble features. This allows a user-definable character string(s) to be added to the data returned by the MSR 3000. Pre-amble is added to the beginning of the data, post-amble is added to the end of data and terminator, and the terminator is added to the end of data. A formatted message block would have the following arrangement:

{Pre-amble}{Message Data}{Terminator}{Post-amble}

LRC Character

LRC is a check character following the end sentinel in an magnetic stripe card. This option allows the MSR 3000 to be set to either send or not send the LRC character.

Track Selection

There is a maximum of three tracks on a magnetic stripe card which contain encoded data. This feature allows you to specify which track(s) to read.

Track Separator

This option allows the application to select the character used to separate data decoded by multiple track magnetic stripe readers. The Track Separator can be any ASCII character.

Data Edit

The MSR 3000 has very strong Data Editing features. The basic concept is to extract only the data fields (such as Name, Account Number. EXP Date, Address, Age, etc.) required by the application program. MSR 3000 then sends these fields in the order specified by the application program. The application developer does not need to have any knowledge of the specific magstripe format. The application developer can use the configurator tool to make their selections. The Configurator creates a header file for all the data edit commands. These features make the high-level application software development a lot easier. See Appendix A, *Data Editing Overview* for detail.

Special Magnetic Card Format Support

To support special magnetic card formats, the MSR 3000 provides two unique features; the Generic Decoder and the Raw Decoder.

Generic Decoder

The Generic Decoder supports a flexible magnetic card format structure. The Generic Decoder can handle special requirements encoded by the ISO standard 5 or 7 bit data formats and is more efficient than the raw data decoder. which should support most other special requirements.

The developer can define the following parameters for each track in the generic decoder:

- ♦ Bit Format: 5 bits with parity or 7 bits with parity
- ♦ Start Sentinel
- ♦ End Sentinel
- ♦ Special reserved characters. Using this feature the developer can redefine the character for any position in the ISO 7 bits or 5 bits character set table. The maximum number of reserved characters is six.

Raw Data Decoder

The Raw Data Decoder sends magnetic data in raw data format so the application program can perform complicated decoding. With this feature, raw data, which are bit level data in the card, can be sent to the application program for further processing. Two ASCII characters represent each raw data byte: the first ASCII character is for the high digit of the hex code, and the second ASCII character is for the low digit of the hex code. For example, the two ASCII characters “4” and “1” represent raw data 41h (01000001).

Track selection is invalid for the raw data decoder; that is, all encoded data of three tracks in the card is sent.

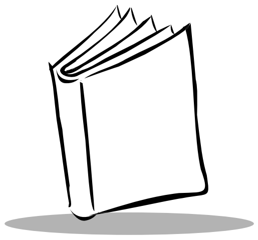
Track identification is sent before data message for each track, when decoder mode is Raw Data Decoder. Track 1 identification is hex 01, track 2 identification is hex 02 and track 3 identification is hex 03.

Library Globals

```
// maximum characters in a card

#define         MAX_CARD_DATA           400

// maximum characters for pre-amble and post-amble

#define         MAX_PRE_POST_SIZE       10

// maximum added field number

#define         MAX_AFLD_NUM            6

// maximum added field length

#define         MAX_AFLD_LEN            6

// maximum data edit send command number

#define         MAX_SCMD_NUM            4

// maximum length in a data edit send command

#define         MAX_SCMD_LEN            40

// maximum characters in whole data edit send command

#define         MAX_SCMD_CHAR           110

// maximum flexible field number

#define         MAX_FFLD_NUM            16

// maximum length in a flexible field setting command

#define         MAX_FFLD_LEN            20

// maximum characters in whole flexible field setting command

#define         MAX_FFLD_CHAR           60

// maximum reserved character to define

#define         MAX_RES_CHAR_NUM        6

// maximum track number
```

```
#define             MAX_TRACK_NUM           3

// characters for track format

#define             TRACK_FORMAT_LEN        5




// structure of reserved character

typedef             struct  ReservedChar {

        Byte                format;

        char                SR_Bits;

        char                SR_Chars;

        } ReservedChar;


Typedef         struct  MSR_Setting {

       Byte             Buffer_mode;

       Byte             Terminator;

       char             Preamble[MAX_PRE_POST_SIZE+1];

       char             Postamble[MAX_PRE_POST_SIZE+1];

       Byte             Track_selection;

       Byte             Track_separator;

       Byte             LRC_setting;

       Byte             Data_edit_setting;

       Byte             Decoder_mode;

       Byte             Track_format[MAX_TRACK_NUM][TRACK_FORMAT_LEN];

ReservedChar            Reserved_chars[MAX_RES_CHAR_NUM];
```

```
        char            Added_field[MAX_AFLD_NUM][MAX_AFLD_LEN+1];

        char            Send_cmd[MAX_SCMD_NUM][ MAX_SCMD_LEN];

        char            Flexible_field[MAX_FFLD_NUM][MAX_FFLD_LEN]

} MSR_Setting;


typedef   MSRSetting*   MSRSetting_Ptr;

typedef   char *        MSRCardInfo_Ptr;

typedef   ReservedChar *ReservedChar_Ptr;
```



Chapter 1
Using The MSR Manager Shared Library

Using the API

The MSR Manager shared library API allows SPT 1740 application programs to control and receive data from the MSR 3000 Magnetic Stripe Reader.

A typical application program uses the MSR Manager shared library to do the following:

- Open the MSR
- Set the MSR
- Handle MSR data or error messages received from the MSR 3000 MSR.
- Close the MSR.

See Chapter 5, *A Simple Application Program Sample* for a detailed walk-through of a simple MSR application program.

Using the MSR Demo Application Program

MSR Demo is a demo application program included with the MSR Manager shared library. It includes all of the API, and demonstrates:

- Using the API to set and get MSR 3000 parameters
- Handling MSR data
- Handling errors.

This demo application program also uses the SPT 1740 graphic interface to display and change MSR 3000 settings.

Chapter 2
MSR Commands

Introduction

The MSR Manager API provides commands to manipulate the MSR 3000.

Return Codes

The MSR commands may return one of the following status codes.

Status Code	Meaning	Suggested Action
MsrMgrNormal	Normal	
MsrMgrErrGlobal	Global parameter error, library global variable error on SPT1740.	Reset SPT 1740.
MsrMgrErrParam	Invalid parameter in function.	Check the parameters.
MsrMgrErrNotOpen	Shared library is not open.	Open the library before invoke any function call.
MsrMgrErrStillOpen	From MSRLibClose() if the library is still open by others.	
MsrMgrErrMemory	SPT 1740 memory error occurred.	Reset SPT 1740.

Status Code	**Meaning**	**Suggested Action**
MsrMgrErrSize	Card information from MSR 3000 MSR overflow.	Check the card and application. Information in a card should not exceed MAX_CARD_DATA (400 (105 for track1, 64 for track2 and 109 for track3) characters, and application should read card information after receiving a MsrDataReadyEvt.
MsrMgrErrNAK	Firmware NAK answer, MSR 3000 reports wrong command was received	check the command or function.
MsrMgrErrTimeout	Waiting timeout.	check the connection between SPT 1740 and MSR 3000.
MsrMgrErrROM	MSR 3000 ROM check error.	replace MSR 3000 unit.
MsrMgrErrRAM	MSR 3000 RAM check error.	reset MSR 3000.
MsrMgrErrEEPROM	MSR 3000 EEPROM check error.	reset MSR 3000.
MsrMgrErrRes	Error response from MSR 3000.	reset MSR 3000.
MsrMgrErrChecksum	Check sum error.	Reset MSR 3000.
MsrMgrBadRead	Read was failed on selected tracks and buffered mode only.	Swipe the card again and check card format.
MsrMgrLowBattery	Battery voltage is too low to enable MSR 3000.	Recharge SPT 1740 battery.
MsrMgrNoData	No data for selected tracks on buffered mode.	
serErrBadPort	Cradle port does not exist.	Check Palm and its OS.
serErrTimeOut	Unable to send or receive data within the specified timeout period.	Check the connection between the SPT 1740 and MSR 3000.
serErrAlreadyOpen	SPT 1740 Cradle port already has an installed foreground owner.	Check the application.
memErrNotEnoughSpace	No enough memory available on the SPT 1740.	Reset SPT1740.

MSR 3000 Command Descriptions

MSR Event

MsrDataReadyEvt – this event will be received by an application to indicate that there is data ready to receive from the MSR 3000.

Note: *This command is available in unbuffered mode only.*

MsrOpen

Purpose

Load and initialize the MSR 3000 Manager Library, and return the versions of the shared library and the MSR 3000 attached.

Prototype

```
Err MsrOpen (UInt refNum, unsigned long *msrVerP, unsigned long *libVerP)
```

Parameters

refNum	library reference number from SysLibLoad or SysLibFind
MsrVerP	pointer to a MSR 3000 version number
LibVerP	pointer to a shared library version number

Return

MsrMgrNormal	open successful

If an error occurs, the return status is one of the following:

MsrMgrErrGlobal	global parameter error
MsrMgrErrMemory	memory error occurred
MsrMgrErrNAK	firmware NAK answer
MsrMgrLowBattery	battery voltage too low to enable MSR 3000
MsrMgrErrRes	error response from MSR 3000 MSR
serErrTimeOut	handshake timeout
serErrBadPort	port does not exist
serErrAlreadyOpen	port was open
MemErrNotEnoughSpace	insufficient memory

Comments

This is first library function to be called by the application program. This function creates and initializes library globals, and enable power to the MSR. Default serial port settings are set to MSR. It also tests communication with the MSR and gets the version number of the MSR 3000.

This function takes approximately 1 second.

As this function controls power to the MSR 3000, care should be taken with its use. To conserve batteries, we recommend opening the MSR only when serviced, and closing it when not needed. This must balanced with the 1 second it takes to open the MSR 3000.

Version number follows the system versioning scheme; 0xMMmfsbbb, where MM is major version, m is minor version, f is bug fix, s is stage: 3-release,2-beta,1-alpha,0-development, bbb is build number for non-releases.

e.g. V1.12b3 would be 0x01122003, V2.00a2 would be 0x02001002 and V1.01 would be 0x01013000.

Example

```
error = MsrOpen(GMsrMgrLibRefNum, &versionNo, &libversion);
```

See Also

MsrClose

MsrClose

Purpose

Close the MSR Manager Library, and free resources.

Prototype

```
Err MsrClose (UInt refNum)
```

Parameters

refNum	library reference number

Return

MsrMgrNormal	close successful

If an error occurs, the return status is one of the following:

MsrMgrErrGlobal	global parameter error
MsrMgrErrMemory	memory error occurred
SerErrBadPort	this port does not exist

Comments

Frees the library globals and removes power to MSR.

Example

```
error = MsrClose(GMsrMgrLibRefNum);
```

See Also

MsrOpen

MsrSetDefault

Purpose

Set MSR 3000 with default settings.

Prototype

```
Err MsrSetDefault (UInt refNum)
```

Parameters

refNum	library reference number from SysLibLoad or SysLibFind

Return

MsrMgrNormal	setting successful.

If an error occurs, the return status is one of the following:

MsrMgrErrNAK	firmware NAK answer.
MsrMgrErrRes	error response from MSR 3000.
serErrTimeOut	handshake timeout.
serErrBadPort	this port does not exist.

Comments

Sets MSR with following default values:

Buffer Mode	Unbuffered
Terminator	CR/LF
Preamble	None
Postamble	None
Track Selection	All Three Tracks
Track Separator	CR
LRC	Do Not Send LRC
Data Edit Setting	Disabled

Example

```
error = MsrSetDefault(GMsrMgrLibRefNum);
```

See Also

MsrClose

MsrGetSetting

Purpose

Get MSR current settings from MSR 3000.

Prototype

```
Err MsrGetSetting (UInt refNum, MSR_Setting *userMSRP)
```

Parameters

refNum	library reference number
userMSRP	pointer to a variable for user MSR setting

Return

MsrMgrNormal	get current setting successful.

If an error occurs, the return status is one of the following:

MsrMgrErrNAK	firmware NAK answer.
MsrMgrErrRes	error response from MSR 3000.
serErrTimeOut	handshake timeout.
serErrBadPort	this port does not exist.

Comments

Gets MSR current settings from MSR 3000.

Example

```
error = MsrGetSetting( GMsrMgrLibRefNum, &appSetting);
```

See Also

MsrSendSetting

MsrSendSetting

Purpose

Send user settings to the MSR 3000.

Prototype

```
Err MsrSendSetting (UInt refNum, MSR_Setting userMSRP)
```

Parameters

refNum	library reference number
userMSR	variable for user MSR setting

Return

MsrMgrNormal	send user setting successful.

If an error occurs, the return status is one of the following:

MsrMgrErrNAK	firmware NAK answer.
MsrMgrErrRes	error response from MSR 3000.
serErrTimeOut	handshake timeout.
serErrBadPort	this port does not exist.

Comments

Sends user setting to MSR 3000.

Example

```
error = MsrSendSetting(GMsrMgrLibRefNum, user_MsrSetting);
```

See Also

MsrGetSetting

MsrGetVersion

Purpose

Get MSR 3000 and software library version.

Prototype

```
Err MsrGetVersion (UInt refNum, unsigned long *msrVerP, unsigned long *libVerP)
```

Parameters

refNum	library reference number
msrVerP	pointer to MSR 3000 version
libVerP	pointer to MSR shared software library

Return

MsrMgrNormal	get status successful.

If an error occurs, the return status is one of the following:

MsrMgrErrNAK	firmware NAK answer.
MsrMgrErrRes	error response from MSR 3000.
serErrTimeOut	handshake timeout.
serErrBadPort	this port does not exist.

Comments

Gets versions of the MSR 3000 and MSR shared software library. Version number follows the system versioning scheme. 0xMMmfsbbb, where MM is major version, m is minor version, f is bug fix, s is stage: 3-release,2-beta,1-alpha,0-development, bbb is build number for non-releases.

e.g. V1.12b3 would be 0x01122003, V2.00a2 would be 0x02001002 and V1.01 would be 0x01013000.

Example

```
error = MsrGetVersions(GmsrMgrLibRefNum, &msrVer, &libVer);
```

See Also

MsrOpen

MsrGetStatus

MsrGetStatus

Purpose

Get status of last magnetic stripe read.

Prototype

```
        Err MsrGetStatus (UInt refNum, BytePtr statusP)
```

Parameters

refNum	library reference number
StatusP	pointer to MSR status

Return

MsrMgrNormal	get status successful.

If an error occurs, the return status is one of the following:

MsrMgrErrNAK	firmware NAK answer.
MsrMgrErrRes	error response from MSR 3000.
serErrTimeOut	handshake timeout.
serErrBadPort	this port does not exist.

Comments

Gets MSR status of last swipe. Status byte format is as the following:

Bit5	1: track3 decoding was successful
	0: error
Bit4	1: track2 decoding was successful
	0: error
Bit3	1: track1 decoding was successful
	0: error
Bit2	1: track3 has sampling data
	0: error
Bit1	1: track2 has sampling data

	0: error
Bit0	1: track3 has sampling data
	0: error

Flag for decoding is effective only when corresponding sampling was successful.

Example

```
error = MsrGetStatus(GmsrMgrLibRefNum, &status);
```

See Also

MsrSelfDiagnose

MsrSelfDiagnose

Purpose

Initiate MSR 3000 self test and return results.

Prototype

```
Err MsrSelfDiagnose (UInt refNum)
```

Parameters

refNum	library reference number

Return

MsrMgrNormal	self diagnose successful.

If an error occurs, the return status is one of the following:

MsrMgrErrNAK	firmware NAK answer.
MsrMgrErrRes	error response from MSR 3000.
MsrMgrErrROM	ROM error.
MsrMgrErrRAM	RAM error.
MsrMgrErrEEPROM	EEPROM error.
serErrTimeOut	handshake timeout.
serErrBadPort	this port does not exist.

Comments

Starts MSR self-diagnostic test and returns check result. It takes approximately 2 seconds.

Example

```
error = MsrSelfDiagnose(GMsrMgrLibRefNum);
```

See Also

MsrGetStatus

MsrSetBufferMode

Purpose

Set buffer mode of MSR 3000.

Prototype

```
Err MsrSetBufferMode (UInt refNum, Byte mode)
```

Parameters

refNum	library reference number
mode	buffer Mode to set MSR

Return

MsrMgrNormal	set buffer mode successful.

If an error occurs, the return status is one of the following:

MsrMgrErrParam	not allowed mode value.
MsrMgrErrNAK	firmware NAK answer.
MsrMgrErrRes	error response from MSR 3000.
serErrTimeOut	handshake timeout.
serErrBadPort	this port does not exist.

Comments

Set Unbuffered Mode or Buffered Mode. Mode should be one of the following:

- MsrBufferedMode
- MsrUnbufferedMode

Example

```
error = MsrSetBufferMode(GmsrMgrLibRefNum, MsrUnbufferedMode);
```

See Also

MsrGetSetting

MsrArmtoRead

Purpose

Enable MSR to be ready for a card swipe in buffered mode.

Prototype

```
Err MsrArmtoRead (UInt refNum)
```

Parameters

refNum	library reference number

Return

MsrMgrNormal	arm to read successful.

If an error occurs, the return status is one of the following:

MsrMgrErrParam	mode neither Unbuffered nor Buffered mode.
MsrMgrErrNAK	firmware NAK answer.
MsrMgrErrRes	error response from MSR 3000.
serErrTimeOut	handshake timeout.
serErrBadPort	this port does not exist.

Comments

Clears the buffer in the MSR and enables the MSR to be ready for a card swipe. This function is for the Buffered Mode only.

Example

```
error = MsrArmtoRead(GmsrMgrLibRefNum);
```

See Also

MsrGetSetting

MsrSetTerminator

Purpose

Set terminator setting to MSR 3000.

Prototype

```
Err MsrSetTerminator (UInt refNum, Byte setting)
```

Parameters

refNum	library reference number
setting	terminator to set MSR

Return

MsrMgrNormal	set terminator successful.

If an error occurs, the return status is one of the following:

MsrMgrErrParam	not allowed setting value.
MsrMgrErrNAK	firmware NAK answer.
MsrMgrErrRes	error response from MSR 3000.
serErrTimeOut	handshake timeout.
serErrBadPort	this port does not exist.

Comments

The Terminator is a designated character which comes at the end of the last track of data, to separate card reads. This function set CR/LF, CR, LF or None as terminator.

The setting should be one of the following:

- MsrTerminatorCRLF
- MsrTerminatorCR
- MsrTerminatorLF
- MsrTerminatorNone

Example

```
error = MsrSetTerminator(GmsrMgrLibRefNum, MsrTerminatorCRLF);
```

See Also

MsrGetSetting

MsrSetPreamble

Purpose

Set a preamble to MSR 3000.

Prototype

```
Err MsrSetPreamble (UInt refNum, char *setting)
```

Parameters

refNum	library reference number
setting	preamble string to set MSR 3000 MSR.

Return

MsrMgrNormal	set preamble successful.

If an error occurs, the return status is one of the following:

MsrMgrErrParam	string too long.
MsrMgrErrNAK	firmware NAK answer.
MsrMgrErrRes	error response from MSR 3000.
serErrTimeOut	handshake timeout.
serErrBadPort	this port does not exist.

Comments

Sets preamble string. String should less than MAX_PRE_POST_SIZE.

Example

```
error = MsrSetPreamble(GmsrMgrLibRefNum, “preamble\n”);
```

See Also

MsrGetSetting

MsrSetPostamble

MsrSetPostamble

Purpose

Set a postamble to MSR 3000.

Prototype

```
        Err MsrSetPostamble (UInt refNum, char *setting)
```

Parameters

refNum	library reference number
setting	postamble string to set MSR 3000 MSR.

Return

MsrMgrNormal	set postamble successful.

If an error occurs, the return status is one of the following:

MsrMgrErrParam	string too long.
MsrMgrErrNAK	firmware NAK answer.
MsrMgrErrRes	error response from MSR 3000.
serErrTimeOut	Handshake timeout.
serErrBadPort	this port does not exist.

Comments

Sets postamble string. String should less than MAX_PRE_POST_SIZE.

Example

```
        error = MsrSetPostamble(GmsrMgrLibRefNum, “postamble\n”);
```

See Also

MsrGetSetting

MsrSetTrackSelection

Purpose

Select tracks to be decoded to MSR 3000.

Prototype

```
        Err MsrSetTrackSelection (UInt refNum, Byte setting)
```

Parameters

refNum	library reference number
Setting	tracks information to get from MSR

Return

MsrMgrNormal	set track selection successful.

If an error occurs, the return status is one of the following:

MsrMgrErrParam	not allowed mode value.
MsrMgrErrNAK	firmware NAK answer.
MsrMgrErrRes	error response from MSR 3000.
serErrTimeOut	Handshake timeout.
serErrBadPort	this port does not exist.

Comments

Sets tracks to be decoded to the MSR 3000. Setting should be one of the following:

MsrAnyTrack

MsrTrack1Only

MsrTrack2Only

MsrTrack3Only

MsrTrack1Track2

MsrTrack1Track3

MsrTrack2Track3

MsrAllThreeTracks

Example

```
error = MsrSetTrackSelection(GmsrMgrLibRefNum, MsrAnyTrack);
```

See Also

MsrGetSetting

MsrSetTrackSeparator

Purpose

Send track separator to MSR 3000.

Prototype

```
Err MsrSetTrackSeparator (UInt refNum, Byte setting)
```

Parameters

refNum	library reference number
Setting	separator for tracks to set MSR

Return

MsrMgrNormal	set track separator successful.

If an error occurs, the return status is one of the following:

MsrMgrErrNAK	firmware NAK answer.
MsrMgrErrRes	error response from MSR 3000.
serErrTimeOut	handshake timeout.
serErrBadPort	this port does not exist.

Comments

Track separator is a designated character which separates data tracks.

Example

```
error = MsrSetTrackSeparator(GmsrMgrLibRefNum, ‘\n’);
```

See Also

MsrGetSetting

MsrSetLRC

Purpose

Send LRC mode to MSR 3000.

Prototype

```
        Err MsrSetLRC(UInt refNum,  Byte setting)
```

Parameters

refNum	library reference number
setting	LRC mode to set MSR

Return

MsrMgrNormal	set LRC successful.

If an error occurs, the return status is one of the following:

MsrMgrErrParam	added string too long.
MsrMgrErrNAK	firmware NAK answer.
MsrMgrErrRes	error response from MSR 3000.
serErrTimeOut	handshake timeout.
serErrBadPort	this port does not exist.

Comments

LRC is a check character following the end sentinel. The setting should be one of the following:

MsrSendLRC

MsrNoLRC

Example

```
        error = MsrSetLRC(GmsrMgrLibRefNum, MsrNoLRC);
```

See Also

MsrGetSetting

MsrSetDataEdit

Purpose

Send data edit mode.

Prototype

```
Err MsrSetDataEdit (UInt refNum, Byte mode)
```

Parameters

refNum	library reference number
mode	Data edit mode to set MSR

Return

MsrMgrNormal	set data edit successful.

If an error occurs, the return status is one of the following:

MsrMgrErrParam	not allowed mode value.
MsrMgrErrNAK	firmware NAK answer.
MsrMgrErrRes	error response from MSR 3000.
serErrTimeOut	handshake timeout.
serErrBadPort	this port does not exist.

Comments

Sets data edit mode. Mode should be one of the following:

MsrDisbaleDataEdit

MsrDataEditMatch

MsrDataEditUnmatch

Example

```
error = MsrSetDataEdit(GmsrMgrLibRefNum, MsrDisableDataEdit);
```

See Also

MsrGetSetting

MsrSetAddedField

Purpose

Set added fields to MSR 3000.

Prototype

```
Err MsrSetAddedField (UInt refNum,

char setting[MAX_AFLD_NUM] [MAX_AFLD_LEN+1])
```

Parameters

refNum	library reference number
setting	added fields strings to set MSR 3000 MSR.

Return

MsrMgrNormal	set added fields successful.

If an error occurs, the return status is one of the following:

MsrMgrErrParam	string too long.
MsrMgrErrNAK	firmware NAK answer.
MsrMgrErrRes	error response from MSR 3000.
serErrTimeOut	handshake timeout.
serErrBadPort	this port does not exist.

Comments

Set added fields strings for data edit. String should less than MAX_AFLD_LEN.

Example

```
char addedField [MAX_AFLD_NUM][MAX_AFLD_LEN+1] = {

“fld1”, “fld2\n”, {0x0A, 0x00}, “”};

error = MsrSetAddedField(GmsrMgrLibRefNum, addedField);
```

See Also

MsrGetSetting

MsrSetDataEditSend

Purpose

Set data edit send commands to MSR 3000.

Prototype

```
Err MsrSetDataEditSend (UInt refNum,

 char setting[MAX_SCMD_NUM] [MAX_SCMD_LEN)
```

Parameters

refNum	library reference number
Setting	send command setting for data edit.

Return

MsrMgrNormal	set added fields successful.

If an error occurs, the return status is one of the following:

MsrMgrErrParam	wrong send command.
MsrMgrErrNAK	firmware NAK answer.
MsrMgrErrRes	error response from MSR 3000.
serErrTimeOut	handshake timeout.
serErrBadPort	this port does not exist.

Comments

Set data edit send commands.

setting is [ccsmd] [dmvsmd][aamvasmd] [flexsmd]

ccsmd = {field_len}{Hex E0}{field}[{field}] {Hex FF}...,

default is {Hex 00}{Hex FF}

dmvsmd = {field_len}{Hex E1}{field}[{field}]{Hex FF}...,

default is {Hex 00}{Hex FF}

aamvasmd = {field_len}{Hex E2}{field}[{field}]{Hex FF}...,

default is {Hex 00}{Hex FF}

flexsmd = {field_len}{Hex E3}{field}[{field}]{Hex FF}...,

default is {Hex 00}{Hex FF}

field_len is the number of bytes from {Hex Ex} to the {field} before {Hex FF}.

field is a one byte field identifier. The highest three bits are used to identify the track number, and the lowest 5 bits is a unique field number.

track7,track6,track5	Track number	
000	Added field number	
001	Track1 field number	
010	Track2 field number	
011	Track3 field number	
111	11111	(Successive field)
1xx	xxxxx	(Reserved)

The valid field number for Credit Card on track1 and track2 is 0~9, 0~6. And there is no valid field number on track3.

The valid field number for California Driver License Card on each track is 0~7, 0~7 and 0~19.

The valid field number for AAMVA Card on each track is 0~6, 0~8 and 0~17.

The valid field number for added field is 0~5.

The valid field number for flexible field is 0~15.

The successive symbol 0xFF is a '~' symbol, so user need not write successive field one by one.

Example

```
char scmd [MAX_SCMD_NUM][MAX_SCMD_LEN] = {

{0x00, 0xFF},

{0x00, 0xFF},

{0x00, 0xFF},

{0x00, 0xFF}};

error = MsrSetDataEditSend(GmsrMgrLibRefNum, scmd);
```

See Also

MsrGetSetting

MsrSetFlexibleField

Purpose

Set added fields to MSR 3000.

Prototype

```
Err MsrSetFlexibleField (UInt refNum,

char setting[MAX_FFLD_NUM] [MAX_FFLD_LEN)
```

Parameters

refNum	library reference number
setting	flexible field setting to set MSR 3000.

Return

MsrMgrNormal	set flexible fields successful.

If an error occurs, the return status is one of the following:

MsrMgrErrParam	wrong setting of flexible field.
MsrMgrErrNAK	firmware NAK answer.
MsrMgrErrRes	error response from MSR 3000.
serErrTimeOut	handshake timeout.
serErrBadPort	this port does not exist.

Comments

Sets flexible fields for data edit.

<Flexible_Field> is [length_match][string_match][search_before]

[search_between][search_after]…

length_match = {field_len} {hex F0} {track_no} {minimum length} {maximum length}

string_match = {field_len} {hex F1} {track_no}, {offset} {string_len} {string}

search_before = {field_len} {hex F2} {track_no}, {field_no} {times} {string_len} {string}

search_between = {field_len} {hex F3} {track_no} {field_no} {times1} {string1_len} {string1} {times2} {string2_len} {string2} search_after = {field_len} {hex F4} {track_no} { field_no} {times1} {offset} {length2} {length1} {string1}

track_no = High three bits of the byte, 001*****|010*****|011*****

field_no = Low five bits of the byte, ***00000~***11111

field_len is the number of bytes from {Hex Fx} to the end of command.

field is a one byte field identifier. The highest three bits are used to identify the track number, and the lowest 5 bits is a unique field number.

If FlexFld[0][0] is 0, then MsrSetFlexibleField() means clear all flexible field.

Example

```
char flexFld [MAX_FFLD_NUM][MAX_FFLD_LEN] = {

{0x00}};

error = MsrSetFlexibleField(GmsrMgrLibRefNum, flexFld);
```

See Also

MsrGetSetting

MsrGetDataBuffer

Purpose

Request card data information for the Buffered Mode.

Prototype

```
Err MsrGetDataBuffer (UInt refNum, MSRCardInfo_Ptr userCardInfoP,

 Byte get_Type)
```

Parameters

refNum	library reference number
UserCardInfoP	pointer to variable for user Card information
get_Type	Type to get data

Return

MsrMgrNormal	No bad read and at least one good read on the selected track.

If an error occurs, the return status is one of the following:

MsrMgrErrParam	wrong setting of flexible field.
MsrMgrErrNAK	firmware NAK answer.
MsrMgrErrRes	error response from MSR 3000, or call MSRGetDataBuffer without “Arm To Read” or “Have not swiped a card yet”.
MsrMgrBadRead	bad read on any one of selected track.
MsrMgrNoData	no card data at all selected tracks
serErrTimeOut	handshake timeout
serErrBadPort	cradle port does not exist

Comments

Request card data Information. This function is for the Buffered Mode only.

get_Type should be one of the following:

- ♦ MsrGetAllTracks
- ♦ MsrGetTrack1

- MsrGetTrack2
- MsrGetTrack3

Application should call MsrArmtoRead() first, before swiping a card and call MsrGetDataBuffer() to get the card information. MsrReadMsrBuffer() itself will issue a MsrArmtoRead(), wait a specified time for user to swipe a card and then get the card information. If the SPT 1740 goes to sleep before calling MsrGetDataBuffer(), the data buffer is lost.

Example

```
error = MsrGetDataBuffer(GmsrMgrLibRefNum, buff, MsrGetAllTracks);
```

See Also

MsrReadMSRBuffer

MsrReadMSRUnbuffer

MsrReadMSRBuffer

Purpose

Request Card Information until receiving data or time out.

Prototype

```
Err MsrReadMSRBuffer (UInt refNum, MSRCardInfo_Ptr userCardInfoP,

UInt waitTime)
```

Parameters

refNum	library reference number
UserCardInfoP	pointer to variable for user Card information
waitTime	timeout in 100ms units

Return

MsrMgrNormal	No bad read and at least one good read on the selected track.

If an error occurs, the return status is one of the following:

MsrMgrErrParam	wrong setting of flexible field.
MsrMgrErrNAK	firmware NAK answer.
MsrMgrErrRes	error response from MSR 3000.
MsrMgrBadRead	Bad read on any one of selected tracks.
MsrMgrNoData	No card data at all selected tracks.
serErrTimeOut	handshake timeout.
serErrBadPort	this port does not exist.

Comments

Issues “Arm to Read”, then requests card information until receiving data or time out. This function is for the Buffered Mode only.

Application should call MsrArmtoRead() first, before swipe a card and call MsrGetDataBuffer() to get the card information. MsrReadMsrBuffer() itself will issue a MsrArmtoRead(), wait a specified time for user to swipe a card and then get the card information.

Example

```
error = MsrReadMSRBuffer(GmsrMgrLibRefNum, buff, 20);

// request and read card information, timeout is 2 seconds
```

See Also

MsrGetDataBuffer

MsrReadMSRUnbuffer

MsrReadMSRUnbuffer

Purpose

Request card data Information for the unuffered mode.

Prototype

```
        Err MsrReadMSRUnbuffer (UInt refNum, MSRCardInfo_Ptr

                         userCardInfoP)
```

Parameters

refNum	library reference number
UserCardInfoP	pointer to variable for user card information

Return

MsrMgrNormal	No error occurred during get.

If an error occurs, the return status is one of the following:

MsrMgrErrParam	wrong setting of flexible field.
MsrMgrErrNAK	firmware NAK answer.
MsrMgrErrRes	error response from MSR 3000.
serErrTimeOut	handshake timeout.
serErrBadPort	this port does not exist.

Comments

Requests card data information. This function is for the unbuffered mode only.

Example

```
        error = MsrReadMSRUnuffer(GmsrMgrLibRefNum, buff);
```

See Also

MsrReadMSRBuffer

MsrReadMSRBuffer

MsrSetDecoderMode

Purpose

Set the decoder mode for the MSR 3000.

Prototype

```
Err MsrSetDecoderMode( UInt refNum, Byte mode )
```

Parameters

refNum	library reference number
mode	decoder mode

Return

MsrMgrNormal	No error occurred during setting.

If an error occurs, the return status is one of the following:

MsrMgrErrParam	wrong setting of flexible field.
MsrMgrErrNAK	firmware NAK answer.
MsrMgrErrRes	error response from MSR 3000.
serErrTimeOut	handshake timeout.
serErrBadPort	this port does not exist.

Comments

Sets the decoder mode for the MSR 3000. Mode should be one of the following:

♦ MsrNormalDecoder	Standard Decoder for ISO, DMV, AAMVA formats
♦ MsrGenericDecoder	Generic Decoder
♦ MsrRawDataDecoder	Raw Data Decoder

Example

```
error = MsrSetDecoderMode(GmsrMgrLibRefNum, MsrNormalDecoder);
```

See Also

MsrSetTrackFormat

MsrSetReservedChar

MsrSetTrackFormat

Purpose

Set the parameters for the Generic Decoder, such as the Bit Format, Start and End Sentinel.

Prototype

```
Err MsrSetTrackFormat( Uint refNum, Byte track_ID, Byte format, Byte SS_Bits,
Byte SS_ASCII, Byte ES_Bits, Byte ES_ASCII )
```

Parameters

refNum	library reference number
track_ID	Track to be set
format	Bit Format
SS_Bits	The bit pattern of Start Sentinel for track 1
SS-ASCII	The ASCII character of Start Sentinel for track 1
ES-Bits	The bit pattern of End Sentinel for track 1
ES-ASCII	The ASCII character of End Sentinel for track 1

Return

MsrMgrNormal	No error occurred during setting.

If an error occurs, the return status is one of the following:

MsrMgrErrParam	wrong setting of flexible field.
MsrMgrErrNAK	firmware NAK answer.
MsrMgrErrRes	error response from MSR 3000.
serErrTimeOut	handshake timeout.
serErrBadPort	this port does not exist.

Comments

Sets the parameters for the Generic Decoder, such as the Bit Format, Start and End Sentinel.

track_Id should be one of the following:

♦ MsrTrack1Format Track 1

♦ MsrTrack2Format	Track 2
♦ MsrTrack3Format	Track 3

The format should be one of the following:

♦ Msr5BitsFormat	5 bits with parity
♦ Msr7BitsFormat	7 bits with parity

Example

```
error = MsrSetTrackFormat(GmsrMgrLibRefNum, MsrTrack1Format, Msr7BitsFormat,
0x51, ‘%’, 0x7C, ‘?’);
```

See Also

MsrSetDecoderMode

MsrSetReservedChar

MsrSetReservedChar

Purpose

Set Special Reserved Characters for Generic Decoder only. Generic Decoder is based on ISO standard 5 or 7 bits encoded format. This command can be used to redefine the character in any position of ISO standard 5 or 7 bits coded character set.

Prototype

```
Err MsrSetReservedChar(Uint refNum, ReservedChar * setting)
```

Parameters

refNum	library reference number
setting	a set of reserved character to set

Return

MsrMgrNormal	No error occurred during setting.

If an error occurs, the return status is one of the following:

MsrMgrErrParam	wrong setting of flexible field.
MsrMgrErrNAK	firmware NAK answer.
MsrMgrErrRes	error response from MSR 3000.
serErrTimeOut	Handshake timeout.
serErrBadPort	this port does not exist.

Comments

Sets Special Reserved Characters.

Up to MAX_RES_CHAR_NUM characters can be set. Be sure to set format of last character to NULL.

The following is the structure defintion for special reserved character.

```
Typedef            struct      ReservedChar {
             Byte              format;
             char              SR_Bits;
             char              SR_Chars;
```

} ReservedCharSetting;

format:	bit format of a special reserved character
SR-Bits:	The bit pattern of a special reserved character
SR-ASCII:	The ASCII character of a special reserved character

Example

```
error = MsrSetReservedChar(GmsrMgrLibRefNum, setting);
```

See Also

MsrSetDecoderMode

MsrSetTrackFormat

Application Templates

The MSR Manager Shared Library is designed to be very easy to use. It also provides flexibility for developers who want to have maximum control. The following are some templates for different application needs.

If the developer needs an application to handle card data as soon as it arrives, then using Unbuffered mode is recommended.

Unbuffered Mode, Simple Application

The application typically follows this template:

1. MsrOpen()
2. Prompt user to swipe a card.
3. Call MsrReadMSRUnbuffer(), when a msrDataReadyKey keyDown event is received.
4. MsrClose()

If the developer needs an application to control the card swipe and card data read by itself, then using Buffered mode is recommended.

Buffered Mode, Simple Application

The application typically follows this template:

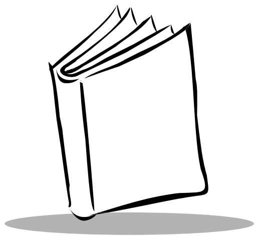
1. MsrOpen()
2. Prompt user to swipe a card.
3. MsrReadMSRBuffer(), if return code indicates No-Data, loop back to 3. or prompt user.
4. MsrClose()

Buffered Mode, Maximum Control

The application typically follows this template:

1. MsrOpen()
2. MsrArmtoRead()
3. Prompt user to swipe a card.
4. Perform other tasks until the application is ready to receive card data.

5. MsrGetDataBuffer(), if return code indicates No-Data, loop back to 5. or other process.
6. MsrClose().



Chapter 3
MSR 3000 Configurator

Introduction

The MSR 3000 Configurator Program is a Windows application program. The Configurator provides an easy to use graphic user interface for selecting features and setup the MSR 3000. The Configurator generates a header file with all the settings. The MSR Manager Shared Library supports the application program with this header file included.

File menu commands

The File menu offers the following commands:

New	Creates a new configuration file.
Open	Opens an existing configuration file.
Exit	Exits MSR 3000 Configurator.

View menu commands

The View menu offers the following commands:

Toolbar	Shows or hides the toolbar.
Status Bar	Shows or hides the status bar.

Help Menu Commands

The Help menu offers the following commands, which provide assistance with this application:

Help Topics	Displays an index of help topics for selection.
About	Displays the version number of this application.

New Command (File Menu)

Use this command to create a new file in MSR 3000 Configurator.

Use the Open command to open an existing file.

Shortcuts

Toolbar:

Keys: CTRL+N

Open Command (File Menu)

Use this command to open an existing file in a new window.

Use the New command to create a new file.

Shortcuts

Toolbar:	
Keys:	CTRL+O

File Open Dialog Box

The following options allow you to specify which file to open:

File Name:	Type or select the filename you want to open. This box lists files with the extension you select in the List Files of Type box.
List Files of Type:	Select the type of file you want to open:*.fig
Drives:	Select the drive in which MSR 3000 Configurator stores the file that you want to open.
Directories:	Select the directory in which MSR 3000 Configurator stores the file that you want to open.

Exit Command (File Menu)

Use this command to end your MSR 3000 Configurator session. The MSR 3000 Configurator prompts you to save files with unsaved changes.

Shortcuts

Mouse:	Double-click the MSR 3000 Configurator's Control menu button in the upper left-hand corner of the screen.
Keys:	ALT+F4

Toolbar Command (View Menu)

Use this command to display and hide the Toolbar, which includes buttons for some of the most common commands in MSR 3000 Configurator, such as File Open. A check mark appears next to the toolbar menu item when the Toolbar is displayed.

See *Toolbar* on page 3-7 for help on using the toolbar.

Toolbar

The toolbar is displayed across the top of the MSR 3000 Configurator's window, below the menu bar. The toolbar provides quick mouse access to many tools used in MSR 3000 Configurator.

To hide or display the Toolbar, choose Toolbar from the View menu (ALT, V, T).

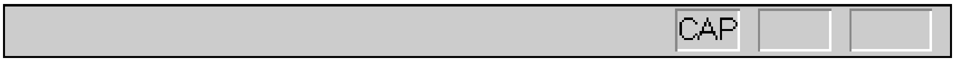
Click:	**To:**
🗋	Create a new file
📂	Open an existing file.
?	Open the Help Topics screen, which allows you to search for information about the Configurator.

Status Bar Command (View Menu)

Use this command to display and hide the Status Bar, which describes the action to be executed by the selected menu item or depressed toolbar button, and keyboard latch state. A check mark appears next to the menu item when the Status Bar is displayed.

See *Status Bar* on page 3-9 for help on using the status bar.

Status Bar



The status bar is displayed at the bottom of the MSR 3000 Configurator window. To display or hide the status bar, use the Status Bar command in the View menu.

The left area of the status bar describes actions of menu items as you use the arrow keys to navigate through menus. This area similarly shows messages that describe the actions of toolbar buttons as you press them, before releasing them. If, after viewing the description of the toolbar button command, you wish not to execute the command, then release the mouse button while the pointer is off the toolbar button.

The right areas of the status bar indicate which of the following keys are latched down:

Indicator	**Description**
CAP	The Caps Lock key is latched down.
NUM	The Num Lock key is latched down.
SCRL	The Scroll Lock key is latched down.

Help Topic Command (Help Menu)

Use this command to display the Help Index Contents screen. From the Index Contents screen, you can jump to step-by-step instructions for using MSR 3000 Configurator and various types of reference information.

Once you open Help, you can click the Contents button whenever you want to return to the Index screen.There are two other search options within the help application. You can search for information via an alphabetic index of topics by selecting the Index command, or you can enter search criteria using the Find command.

About Command (Help Menu)

Use this command to display the copyright notice and version number of your copy of MSR 3000 Configurator.

Context Help Command

Use the Context Help command to obtain help on the MSR 3000 Configurator. The Help topic for the item you selected displays.

Shortcut

Keys:	SHIFT+F1

Title Bar

"MSR 3000 Configurator" displays in the title bar.

The title bar is located along the top of a window. It contains the name of the application and document.

To move the window, drag the title bar. Note: You can also move dialog boxes by dragging their title bars.

A title bar may contain the following elements:

- Application Control-menu button
- Document Control-menu button
- Maximize button
- Minimize button
- Name of the application
- Name of the document
- Restore button

Scroll Bars

Displayed at the right and bottom edges of the document window. The scroll boxes inside the scroll bars indicate your vertical and horizontal location in the document. You can use the mouse to scroll to other parts of the document.

Restore Command (Control menu)

Use this command to return the active window to its size and position before you chose the Maximize or Minimize command.

Move Command (Control Menu)

Use this command to display a four-headed arrow so you can move the active window or dialog box with the arrow keys.

Note: *This command is unavailable if you maximize the window.*

Shortcut

Keys:	CTRL+F7

Size Command (Control Menu)

Use this command to display a four-headed arrow so you can size the active window with the arrow keys.

✥

After the pointer changes to the four-headed arrow:

1. Press one of the DIRECTION keys (left, right, up, or down arrow key) to move the pointer to the border you want to move.
2. Press a DIRECTION key to move the border.
3. Press ENTER when the window is the size you want.

Note: *This command is unavailable if you maximize the window.*

Shortcut

Mouse:	Drag the size bars at the corners or edges of the window.

Minimize Command (Control Menu)

Use this command to reduce the MSR 3000 Configurator window to an icon.

Shortcut

Mouse:	Click the minimize icon ▼ on the title bar.
Keys:	ALT+F9

Maximize Command (Control Menu)

Use this command to enlarge the active window to fill the available space.

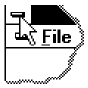
Shortcut

Mouse:	Click the maximize icon ▲ on the title bar; or double-click the title bar.
Keys:	CTRL+F10 enlarges a document window.

Close Command (Control Menu)

Use this command to close the MSR 3000 Configurator.

Double-clicking the Control-menu box is the same as choosing the Close command.



Shortcuts

Keys:	ALT+F4 closes the application window

Configurator Properties Buttons

There are four buttons which appear on each screen within the configurator.

Save Config Button

Use this button to save the active configuration file to its current name and directory. When you save a file for the first time, MSR 3000 Configurator displays the Save As dialog box so you can name your file.

File Save As dialog box

The following options allow you to specify the name and location of the file you're about to save:

File Name:	Type a new filename to save a file with a different name. A filename can contain up to eight characters and an extension of up to three characters. MSR 3000 Configurator adds the extension you specify in the Save File As Type box.
Drives:	Select the drive in which you want to store the document.
Directories:	Select the directory in which you want to store the document.

Close Button

Use this button to close the MSR 3000 Configurator Properties dialog. If you didn’t save, the current settings changed will lost.

Default All Button

Use this button to restore all configuration settings to their default values.

Help Button

Use this button to display detailed help information about the fields on the screen.

Chapter 4
Using the Configurator to Set the MSR 3000

Introduction

The Configurator is a Windows GUI utility. It helps the application developer to do MSR 3000 configuration and be aware of MSR 3000 setting structure. The Configurator can create a CodeWarrior include file for selected MSR 3000 setting, and the application developer can use the include file and call MsrSendSet() function to set up the MSR 3000.

MSR Setting Structure

```
    typedef              struct       ReservedChar {

           Byte                       format;

           char                       SR_Bits;

           char                       SR_Chars;

           } ReservedChar;




      Typedef    struct       MSR_Setting {

           Byte              Buffer_mode;

           Byte              Terminator;

           char              Preamble[MAX_PRE_POST_SIZE+1];
```

```
        char             Postamble[MAX_PRE_POST_SIZE+1];

        Byte             Track_selection;

        Byte             Track_separator;

        Byte             LRC_setting;

        Byte             Data_edit_setting;

        Byte             Decoder_mode;

        Byte             Track_format[MAX_TRACK_NUM][TRACK_FORMAT_LEN];

        ReservedChar     Reserved_chars[MAX_RES_CHAR_NUM];

        char             Added_field[MAX_AFLD_NUM][MAX_AFLD_LEN+1];

        char             Send_cmd[MAX_SCMD_NUM][ MAX_SCMD_LEN];

        char             Flexible_field[MAX_FFLD_NUM][MAX_FFLD_LEN]

    } MSR_Setting;
```

MSR Constant

```
        #define          LF                0x0A

        #define          CR                0x0D

        #define          DC1               0X11

        #define          DC3               0X13

        #define          MsrUnbufferedMode '0'

        #define          MsrBufferedMode   '1'

        #define          MsrArmtoReadMode        '0'

        #define          MsrClearBufferMode      '1'

        #define          MsrLEDOff               '0'

        #define          MsrLEDOn                '1'

        #define          MsrLEDBlink             '2'
```

```
#define         MsrTerminatorCRLF       '0'

#define         MsrTerminatorCR         '1'

#define         MsrTerminatorLF         '2'

#define         MsrTerminatorNone       '3'

#define         MsrAnyTrack             0

#define         MsrTrack1Only           1

#define         MsrTrack2Only           2

#define         MsrTrack1Track2             3

#define         MsrTrack3Only               4

#define         MsrTrack1Track3             5

#define         MsrTrack2Track3             6

#define         MsrAllThreeTracks           7

#define       MsrNoLRC                  '0'

#define       MsrSendLRC                '1'

#define       MsrDisableDataEdit        '0'

#define       MsrDataEditMatch          '1'

#define       MsrDataEditUnmatch        '3'


#define       MsrGetAllTracks           '0'

#define       MsrGetTrack1              '1'

#define       MsrGetTrack2              '2'

#define       MsrGetTrack3              '3'
```

Include File Format Created by Configurator

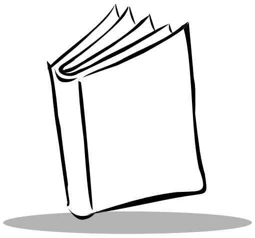
```
MSRSetting  user_MsrSetting  = {

       MsrBufferedMode,

       MsrTerminatorCRLF,

        “Preamble\n”,

       “Postamble\n”,

       MsrAnyTrack,

       CR,

       MsrSendLRC,

       MsrDisableDataEdit,

       {“addf1”,”addf2”,”addf3”, “”, “”,””},

       {{0x08, 0xE0, …},{0x10, 0xE1, …},{0X06, 0XE2, },{NULL}},

       {{0XF0, …},{0XF1, …}…}


}
```

Generating an Include File by Configurator

1. In the File Menu, Click New to display the MSR Configurator Properties screen.
2. On the MSR Configurator Properties screen, select all fields, as described in Appendix A, *Data Editing Overview*.
3. Click the “Save Config” button to display a “Save As” dialog box.
4. Change directory to where the file should be saved.
5. Type the file name in the “File Name” edit box. The default extension of the file is “.h”.
6. Press the “Save” button to save the file or the “Cancel” button exit without saving.

Setting the MSR 3000 with the Configurator

1. Create a include file by Configurator to a specified setting, actually it define a static variable user_MsrSetting.
2. Include this include file in the application program.
3. Call SysLibFind(MsrMgrLibName, &GMsrMgrLibRefNum) and/or SysLibLoad(MsrMgrLibTypeID, MsrMgrLibCreatorID, &GMsrMgrLibRefNum) to load library and get library reference number GmsrMgrLibRefNum.
4. Call MsrOpen(GMsrMgrLibRefNum, &msrVer, &libVer) to open MSR shared library.
5. Call MsrSendSetting(GMsrMgrLibRefNum, user_MsrSetting) to set MSR 3000.
6. Call MsrClose(GmsrMgrLibRefNum) to close the library.
7. Call SysLibRemove(GMsrMgrLibRefNum) to remove library from memory.



Chapter 5 A Simple Application Program Sample

Include Files

The following two #include statements provide MSR Manager interface definitions.

- #include "MsrMgrLib.h"

PilotMain Routine

The PilotMain function is a standard Palm orginizer application. If the launch code is sysAppLaunchCmdNormalLaunch, the application program will do initialization in AppStart and an run event loop until it closes the application program. At this point, the application program will be terminated by AppStop.

```
/***********************************************************************

 *

 * FUNCTION:    PilotMain

 *

 * DESCRIPTION: This is the main entry point for the application program.

 *

 * PARAMETERS:  cmd - word value specifying the launch code.

 *             cmdPB - pointer to a structure that is associated with the launch code.

 *              launchFlags -  word value providing extra information about the launch.

 * RETURNED:    Result of launch

 *

 * REVISION HISTORY:

 *

 *

 ***********************************************************************/

static DWord PilotMain(Word cmd, Ptr cmdPBP, Word launchFlags)

{

       Err      error;

       FormPtr  frmP;


       switch (cmd)
```

```
     {
     case sysAppLaunchCmdNormalLaunch:
             error = AppStart();
             if (error) {
                  AppStop();
                  return error;
              }

                  frmP = FrmInitForm(MSRInfoForm);
                  FrmDoDialog(frmP);
                  // Delete the info form.
                   FrmDeleteForm(frmP);

                  FrmGotoForm(MainForm);

                  AppEventLoop();
                  AppStop();

          default:
                  break;

          }
     return 0;
}
```

AppStart Function

The AppStart function shows what you need to do to initialize the MSR. This function:

- Loads the MSR Manager shared library.
- Powers on the MSR.
- Initiates communication between the application program and MSR 3000.
- Allocates a global handle for card data to display.

```
/*********************************************************************
 *
 * FUNCTION:     AppStart
 *
 * DESCRIPTION:  Initialize Application program.
 *
 * PARAMETERS:   nothing
 *
 * RETURNED:     Err value 0 if nothing went wrong
 *
 * REVISION HISTORY:
 *
 *

 *********************************************************************/

  static Err AppStart(void)

  {

    Err                 error;

    VoidHand            gH;

    MSRLibGlobals_Ptr   gP;

    SysLibTblEntryPtr   libEntryP;

    unsigned long       versionNo;


    // Load the MSR library

    error = SysLibFind(MsrMgrLibName, &GMsrMgrLibRefNum);
if ( error) {
       // to load MSR library
       error = SysLibLoad(MsrMgrLibTypeID, MsrMgrLibCreatorID, &GMsrMgrLibRefNum);
          }

          if ( error == MsrMgrNormal) {
               error = MsrOpen(GMsrMgrLibRefNum, &versionNo);
               if (error != MsrMgrNormal) {
                   FormPtrfrmP;

                   frmP = FrmInitForm(NoMSRForm);
                   FrmDoDialog(frmP);
                   // Delete the info form.
                   FrmDeleteForm(frmP);
                   //MsrClose(GMsrMgrLibRefNum);
                   return (error);
               }
          };


          // Allocate a new memory chunk that will contain received string.
          newHandle = MemHandleNew(MAX_CARD_DATA);
          return (MsrMgrNormal);
  }
```

MainFormHandleEvent Function

PilotMain calls EventLoop after calling AppStart. The MainFormHandleEvent function is an event handler for main form. It deals with following events.

- FrmLoadEvent: Initialize and load Main Form
- FrmOpenEvent: Draw the main Form
- FrmCloseEvent: Close the main Form
- MenuEvent: Open the About Form
- KeyDownEvent with MsrDataReadyEve: call RS232_Receive to read card data and display it.

```
/***********************************************************************
 *
 * FUNCTION:    MainFormHandleEvent
 *
 * DESCRIPTION: This routine is the event handler for the
 *              "MainForm" of this application program.
 *
 * PARAMETERS:  eventP  - a pointer to an EventType structure
 *
 * RETURNED:    true if the event has handle and should not be passed
 *              to a higher level handler.
 *
 * REVISION HISTORY:
 *
 *
 ***********************************************************************/

static Boolean MainFormHandleEvent(EventPtr eventP)

{

    Boolean handled = false;

    FormPtr frmP;



    Err         error;

      FieldPtr   fld;
```

```
switch (eventP->eType)

          {

          case menuEvent:

                switch (eventP->data.menu.itemID)

                {

                            case MainOptionsAboutMSRSample:

                                    // Load the info form, then display it.

                                    MenuEraseStatus (CurrentMenu);

                                    // Clear the Main form.

                                    frmP = FrmInitForm(MSRInfoForm);

                                    FrmDoDialog(frmP);

                                    // Delete the info form.

                                    FrmDeleteForm(frmP);

                                    handled = true;

                                    break;

                }


                break;


          case frmLoadEvent:

               FrmInitForm(MainForm);

               handled = true;

               break;
```

```
case frmOpenEvent:

    frmP = FrmGetActiveForm();

    FrmDrawForm(frmP);

    handled = true;

    break;


case frmCloseEvent:

     frmP = FrmGetActiveForm();

     fld = GetObjectPtr(MainCardInfoField);

     FldSetTextHandle(fld, NULL);

     frmP = FrmGetActiveForm();

     FrmEraseForm(frmP);

     FrmDeleteForm(frmP);

     handled = true;

     break;


case ctlSelectEvent:  // A control button was pressed and released.

    break;


case keyDownEvent:


   if ((eventP->data.keyDown.chr==msrDataReadyKey))  {

          handled=true;

          error = MSR_Receive();
          }
          break;
       default:
          break;
       }

    return handled;
}
```

```
/*********************************************************************
 *
 * FUNCTION:    MSR_Receive
 *
 * DESCRIPTION: This routine is check RS232 Port
 *              and display received data on DataInfo field
 *
 * PARAMETERS:  frmP  - a pointer to Main Form
 *
 * RETURNED:    true if no error occured.
 *
 * REVISION HISTORY:
 *
 *
 *********************************************************************/

static UInt MSR_Receive()

{

       FieldPtr                     fld;

       Err                          error;

       unsigned short               numReceived;

       CharPtr                      newText;



 // get Field pointer

       fld = GetObjectPtr(MainCardInfoField);

// RS232 receive



// get a card data time out is 500 ms

       error = MsrReadMSRUnbuffer( GMsrMgrLibRefNum, buff);

       if (error)

          return (error);

       numReceived = StrLen(buff);
if ( numReceived ){

   // Lock down the handle and get a pointer to the memory chunk.

   newText = MemHandleLock(newHandle);

   // clear screen

   *newText = NULL;

   FldSetTextHandle(fld, newHandle);

   FldDrawField(fld);

   // Copy the data from the RS232 to the new memory chunk.

   if (numReceived && (numReceived<MAX_CARD_DATA)) {

         StrCopy(newText, buff);

   }

   // Unlock the new memory chunk.

   MemHandleUnlock(newHandle);

   // Set the field's text to the data in the new memory chunk.

   FldSetTextHandle(fld, newHandle);

   FldDrawField(fld);

}

return (MsrMgrNormal);


}
```

AppStop Function

The AppStop function is called at the end of PilotMain. It

- Closes the MSR Manager shared library
- Powers off the MSR
- Frees the global handle for card data to display

```
/***********************************************************************
 *
 * FUNCTION:    AppStop
 *
 * DESCRIPTION: Save the current state of the application program.
 *
 * PARAMETERS:  nothing
 *
 * RETURNED:    nothing
 *
 * REVISION HISTORY:
 *
 *
 ***********************************************************************/

static void AppStop(void)

{

       Err  error;


       // close MSR manager library

       error = MsrClose(GMsrMgrLibRefNum);


       // Uninstall the MSR Manager library

       if ( !GMsrMgrLibWasPreLoaded && GMsrMgrLibRefNum != sysInvalidRefNum )

          {

          error = SysLibRemove(GMsrMgrLibRefNum);
```

```
            ErrFatalDisplayIf(error, "error uninstalling MSR Manager library.");

            GMsrMgrLibRefNum = sysInvalidRefNum;

            }


         // Free a memory chunk

         if (newHandle)

            MemHandleFree(newHandle);


}
```

Appendix A
Data Editing Overview

Introduction

The data editing feature allows you to edit the data that has been read from a magnetically encoded card before data is transmitted to the application software. The application software translates the information into the exact format expected by the application software.

Functions

The following functions can be performed on the data input record:

Rearrange the Data

The fields, within a track, created by established standards, can be transmitted to the application software in any order desired, regardless of the order in which they occurred in the card track.

Insert Character Strings into the Output Data Record

Character Strings can be defined and inserted at any place in the data output record.

Duplicate Fields

Fields, within a track, can be transmitted to the application software as many times as desired and in any order.

Select Output Fields

Fields, within a track, can be selected for output or not selected for output.

Output Method

[Matched or Unmatched] If matched, MSR 3000 does not send data that does not match the data edit formula. Otherwise, all data that does not match the data edit formula is sent.

Fields

The data editing concept is based upon fields. The standard field location, length, and content are determined by Standards. The field standards for ISO Credit Cards, California driver's licenses, and AAMVA driver's licenses are listed in Appendix A. By separating the input data record into small blocks(fields), each block can then be treated separately. Additional fields can also be added to the record in any position, allowing specific functions, such as carriage returns. The fields are identified by field number starting with the character 'a' up to 'z', in the order they were created, starting with the predetermined fields in the standard and adding any newly created fields, allowing as many as 26 fields per track to be defined. These fields are then sent to the application software in the order in which the user specifies.

For example, if the input data record is in the Credit Card Format for Track2:

```
                            ;1234567890123456=9912xxxxxxx?c

Field id        a[               b           ]c[ d][  e  ]fg
```

and your application software is expecting the data to be in the following format:

```
9912<CR>

1234567890123456<CR>
```

then we must break the input data record into fields, select only the desired fields, reverse the order they are sent to the application software, create a new field<CR> and insert it after each field.

We do this by using the defined fields and adding a new field:

```
Field b = 1234567890123456

Field d = 9912

Added Field a = <CR>

And sending {Field d}{Added Field a}{Field b}{Added Field a}
```

Formulas

A set of instructions to edit data is referred to as a data editing “formula”. The MSR 3000 supports four types of formulas:

- ♦ Credit Card
- ♦ California driver’s license
- ♦ AAMVA driver’s license
- ♦ Customized Format.

The user can either define four types of formula or only one formula each time. At the same time, the MSR 3000 can only keep one Credit Card, California driver’s licenses, AAMVA driver’s licenses and customized format.

If the input data matches the format (Credit Card, Driver’s License, etc.) of the formula, then it will apply the data editing functions and output the reformatted data to the application software. If the input data does not match the criteria spelled out in the first formula, then the match criteria of the second formula is applied. This process continues for each of the successive formulas until a match is found. If no match is found to any of the formulas programmed into the MSR 3000, then either nothing or the unedited data record is transmitted to the application software, according to the Data Edit Mode setting.

Added Field

An output field is created containing the character string. Up to 6 fields can be defined. The maximum characters of each field is 6.

Search Method

When working with a customer-defined format which is not Credit Card, DMV or AAMVA format, the MSR 3000 supports the following five methods:

Length Match:	The card data on the specified track has to meet the minimum and maximum length requirements.
String Match:	The card data on the specified track has to include the specified string in the specified position.
Search Before:	Generate a new field, which contains the whole message data before matching the specified string for specified times on the specified track.
Search Between:	Generate a new field, which contains the whole message data between the matches of the two specified strings for specified times on the specified track.

Search After:	Generate a new field, which contains the message data with the specified length at a specified offset after matching the specified string for a specified times on the specified track.

Operation

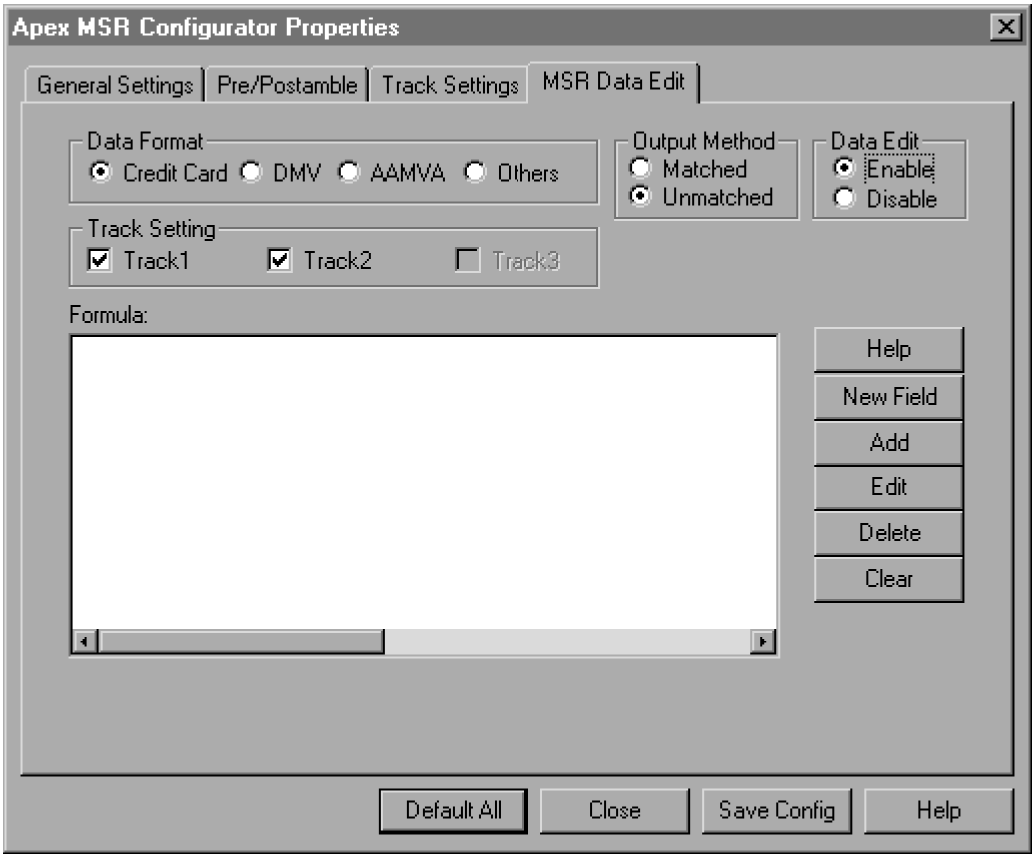
The data editing of the magnetic stripe is based on fields divided for a format. By separating the input data into fields, each field can be transmitted separately. Additional fields can also be added, allowing specific functions, such as carriage returns to be inserted at any place in data transmitted to the application.

The following includes the all operations performed on the formula:

Add Formula - To add a formula:

1. Set the data edit to ‘enable’.
2. Select one of the data formats, such as ‘Credit Card’.
3. Select the tracks that you want to transmit the data to the application.

Default is track1, track2 and track3 selected.


Apex MSR Configurator Properties

General Settings | Pre/Postamble | Track Settings | MSR Data Edit

Data Format: [✓ Credit Card] DMV AAMVA Others

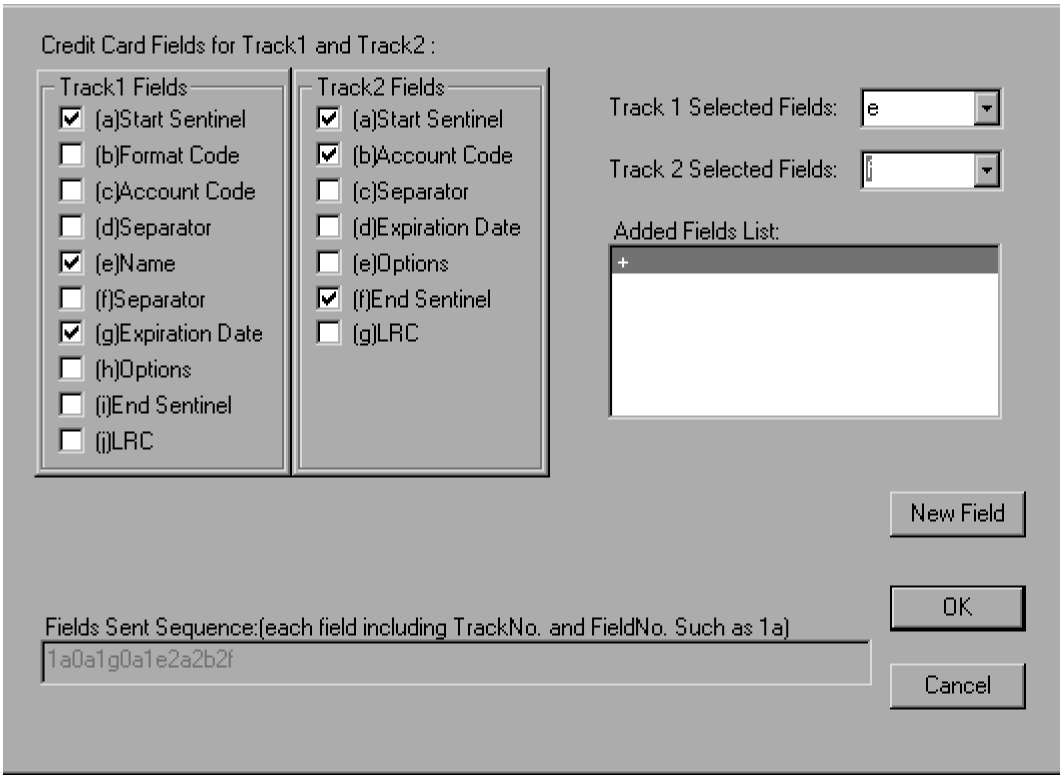
Output Method: Matched [✓ Unmatched]

Data Edit: [✓ Enable] Disable

Track Setting: [✓ Track1] [✓ Track2] Track3

Formula:

4. Press the Add button to display the dialogue box with the selected data format fields. For example, the Credit Card Format.


Credit Card Fields for Track1 and Track2 :

Track1 Fields
- [x] (a)Start Sentinel
- [ ] (b)Format Code
- [ ] (c)Account Code
- [ ] (d)Separator
- [x] (e)Name
- [ ] (f)Separator
- [x] (g)Expiration Date
- [ ] (h)Options
- [ ] (i)End Sentinel
- [ ] (j)LRC

Track2 Fields
- [x] (a)Start Sentinel
- [x] (b)Account Code
- [ ] (c)Separator
- [ ] (d)Expiration Date
- [ ] (e)Options
- [x] (f)End Sentinel
- [ ] (g)LRC

Track 1 Selected Fields: e

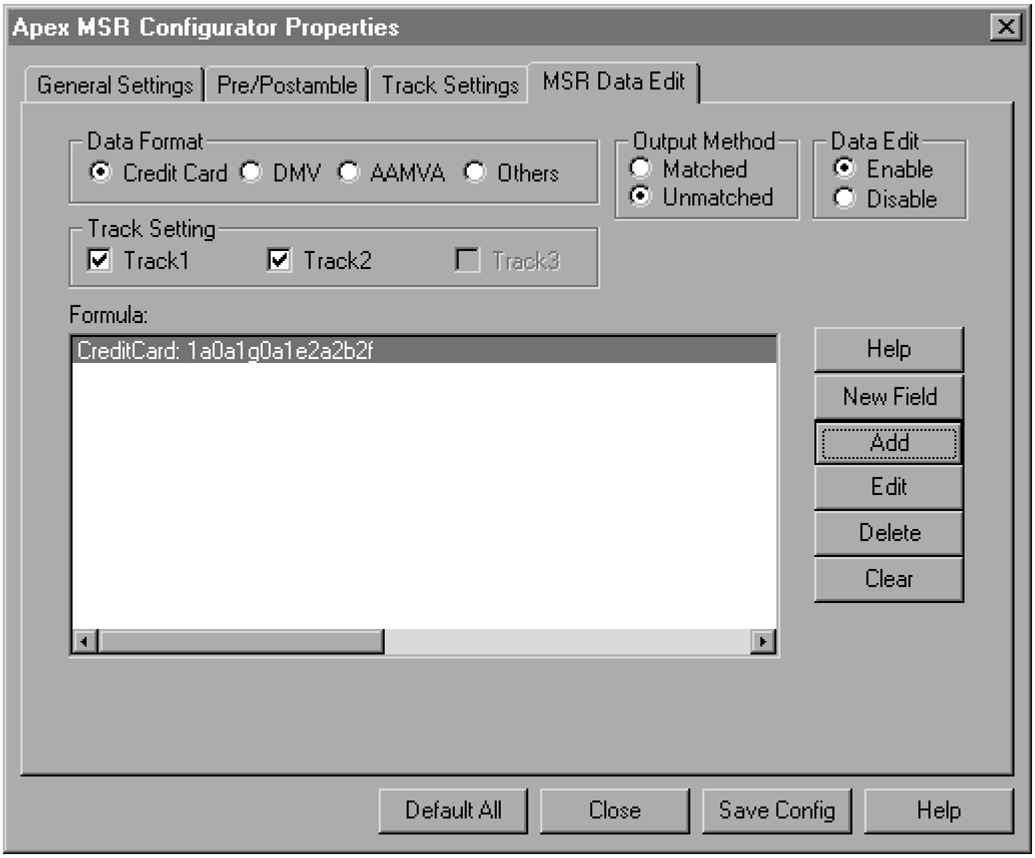
Track 2 Selected Fields:

Added Fields List:

Fields Sent Sequence:(each field including TrackNo. and FieldNo. Such as 1a)

1a0a1g0a1e2a2b2f

5. In this dialog box , check the fields you want transmitted to the application in track1, track2 and track3. The fields selected display in the Track Selected Fields for each track.
6. If you click the new field button, you can input characters in new filed dialog box to get the new field, you can put this field in any place of output data.
7. In the Track–Selected-Field pull-down list, which lists the all fields selected in this track, select the fields to generate fields sequence which are then displayed in the Fields Sent Sequence edit box. The fields will be sent to the application according to this sequence.
8. Press OK to save the formula, or Cancel button to exit without saving.



Data Format: [✓ Credit Card] DMV AAMVA Others

Output Method: Matched [✓ Unmatched]

Data Edit: [✓ Enable] Disable

Track Setting: [✓ Track1] [✓ Track2] Track3

Formula:

CreditCard: 1a0a1g0a1e2a2b2f

Example

```
Data Edit : Enable; Data Format: Credit Card;

Track Selection: track1 and track2;

Input Data :

              %B4000111222333^SMITH/JOHN R.DR^930458799999?C

   Field No.  Field Name      Field Value

          (a)     Start Sentinel    %
```

```
              (b)     Format Code      B

              (c)     Account Code     4000111222333

              (d)     Separator        ^

              (e)     Cardholder Name  Smith/John R. DR

              (f)     Separator      ^

              (g)     Expiration Date  9304

              (h)     Optional Data    58799999

              (i)     End Sentinel     ?

              (j)     LRC              C

     ;4000111222333=93045879999900000000000000?C

              (a)     Start Sentinel   ;

              (b)     Account Code     4000111222333

              (c)     Separator        =

              (d)     Expiration Date  9304

              (e)     optional Data    5879999900000000000000

              (f)     End Sentinel     ?

              (g)     LRC              C
```

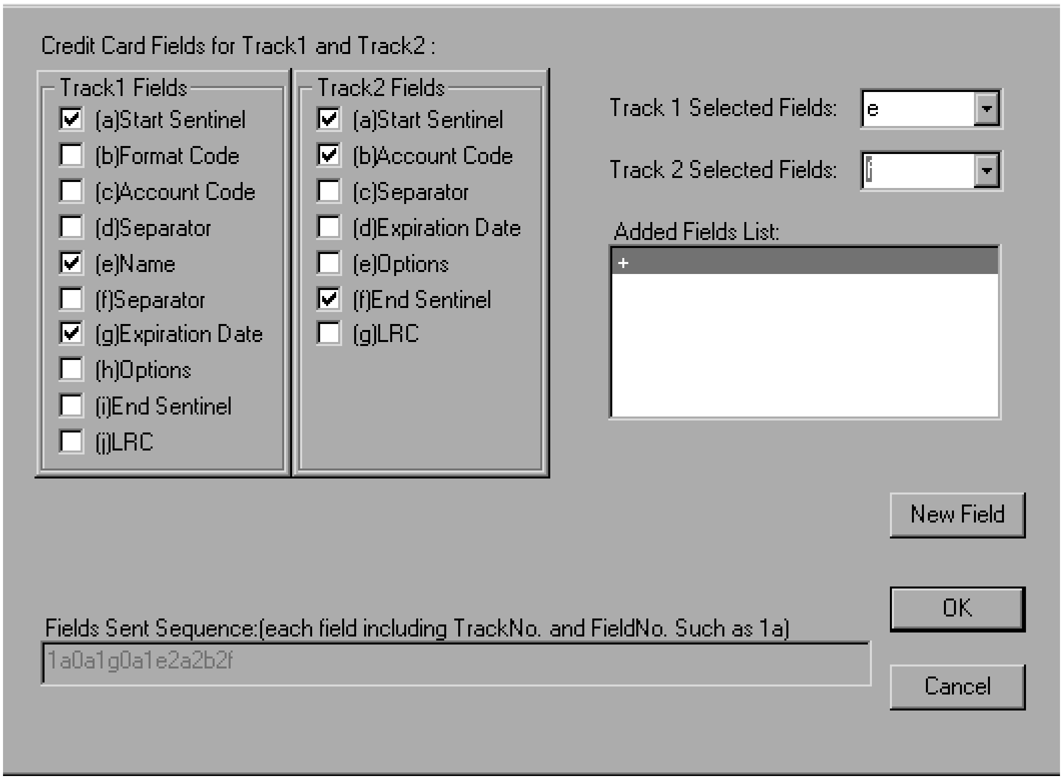
If you want to send the start sentinel, cardholder name and expiration date in track1; and the start sentinel, account code and end sentinel in track2, you should select fields (a), (e), (g) in track1 and field (a), (b), (f) in track2. If you need a new field with value '+' in track1, press the new field button and key in '+' in edit box. Now you have field (a),(e),(g) in track1 and (a),(b),(f) in track2. The added field is '+' .

```
Track1:

      Field (a) = %

      Field (e) = Smith/John R. DR

      Field (g) = 9304
```

```
Track2:

                        Field (a) = ;

                        Field (b) = 4000111222333

                        Field (f) = ?

Added Field: '+'

Track1 fields selected are  (a)(e)(g);

Track2 fields selected are (a)(b)(f);

If you want to send the expiration date before cardholder name and add a new
field to use as a separator, go to the combo box and generate the sequence
(1a)(0a)(1g)(0a)(1e), so you get the formula in the list box:

                        Credit Card: (1a)(0a)(1g)(0a)(1e)(2a)(2b)(2f)

The output data should be:

                        Track1:

                             %+9304+Smith/John R. DR

                        Track2:

                             ;4000111222333?

Track1 and track2 output data will be transmitted to application.
```

Edit Formula - To edit the formula selected in the formula list box:

1. Enable data editing.
2. Press the Edit button, which shows the dialogue box with data format of the selected formula in the formula list box.
3. In this dialog box , select the fields to be transmitted to the application in track1, track2 and track3. The selected fields are displayed in the Track Selected Fields for each track.
4. If you click the new field button, you can input characters in new field dialog box to get the new field, you can put this field in any place of output data.



Credit Card Fields for Track1 and Track2 :

Track1 Fields
- ✓ (a)Start Sentinel
- ☐ (b)Format Code
- ☐ (c)Account Code
- ☐ (d)Separator
- ✓ (e)Name
- ☐ (f)Separator
- ✓ (g)Expiration Date
- ☐ (h)Options
- ☐ (i)End Sentinel
- ☐ (j)LRC

Track2 Fields
- ✓ (a)Start Sentinel
- ✓ (b)Account Code
- ☐ (c)Separator
- ☐ (d)Expiration Date
- ☐ (e)Options
- ✓ (f)End Sentinel
- ☐ (g)LRC

Track 1 Selected Fields: e

Track 2 Selected Fields: f

Added Fields List:
+

Fields Sent Sequence:(each field including TrackNo. and FieldNo. Such as 1a)
1a0a1g0a1e2a2b2f

5. In the Track–Selected-Field pull-down list, which lists the all fields selected in this track, select the fields to generate fields sequence which are then displayed in the Fields Sent Sequence edit box. The fields will be sent to the application according to this sequence.
6. Press OK to save the formula, or Cancel button to exit without saving.

Example

```
Enable data editing. If the formula “Credit Card :
(1a)(0a)(1g)(0a)(1e)(2a)(2b)(2f)”  is selected, Press the Edit button the Credit
Card format dialog box will show. (a)(e)(g) in track1 and (a)(b)(f) in track2
were selected.

Track1 fields selected are (a)(e)(g);
```

```
Track2 fields selected are (a)(b)(f);

If you want send cardholder name before expiration date ,select the (a)(e)(g)
sequentially in the track1 sequence sent combo box.

Track1 sequence sent is changed to (a)(e)(g);

Press the OK button you will get the new formula in the list box of the MSR Edit

dialog-box.

                              Credit Card : (1a)(1e)(1g)(2a)(2b)(2f)

The output is

                              Track1:

                                   %Smith/John R. DR9304

                              Track2:

                                   ;4000111222333?
```

Delete Formula - To Delete a formula:

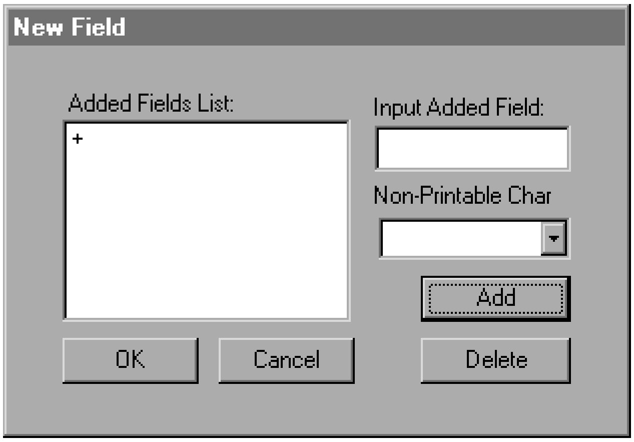
1. Enable data editing.
2. Select a formula you want to delete, then press the Delete button. This formula is deleted from the formula list box.

Clear Formulas - To Delete all formulas:

1. Enable data editing.
2. Press the Clear button. All formulas are deleted from the formula list box.

New Field - To add new fields:

1. Click the New Field button.
2. Type in the string in the Input Added Field, or, if you want to input a non-printable character, you can use the non-printable character pull-down list to select the non-printable character.
3. Press Add button to append the new field to the added fields list. Press Delete button to delete the selected field from the added fields list.
4. Press OK to accept, or press Cancel to exit without saving changes.

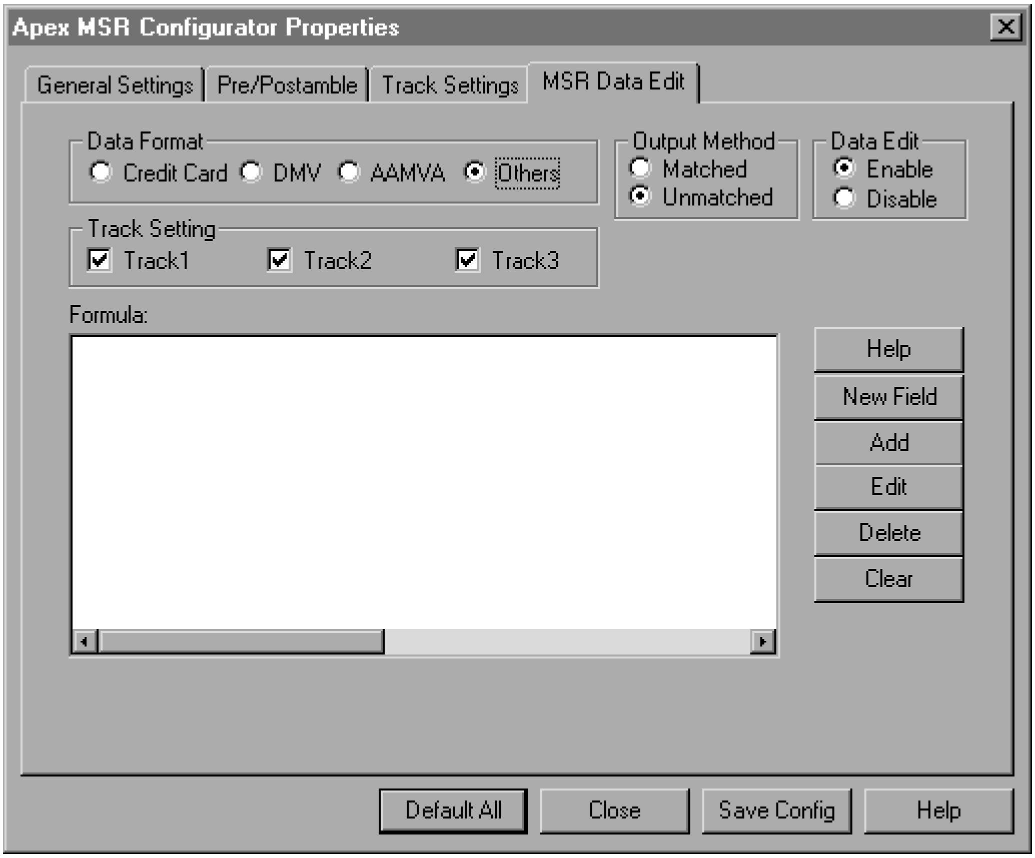


Note: *The maximum number of new fields is 6, the maximum number of characters for each field is 6.*

Customized Format

To create a customized format:

1. On the MSR Configurator Properties screen, select the Others option in the Data Format Group.



Apex MSR Configurator Properties

General Settings | Pre/Postamble | Track Settings | MSR Data Edit

Data Format: Credit Card, DMV, AAMVA, [✓ Others]

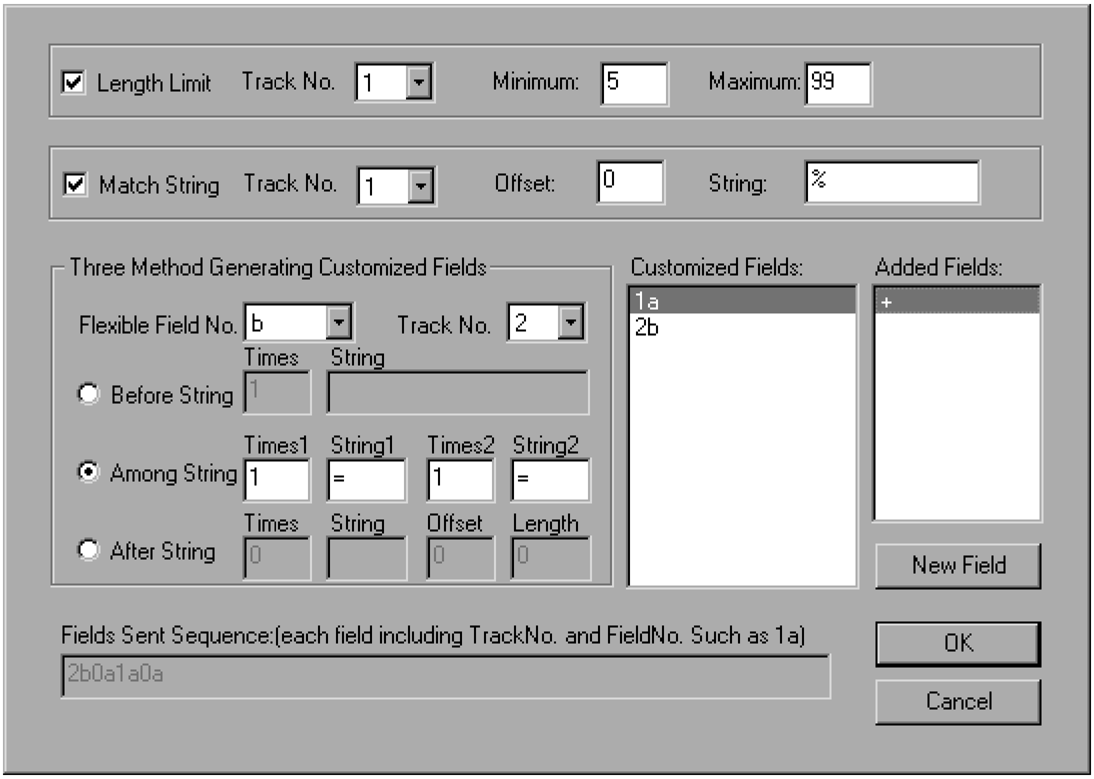
Output Method: Matched, [✓ Unmatched]

Data Edit: [✓ Enable], Disable

Track Setting: [✓ Track1] [✓ Track2] [✓ Track3]

Formula:

2. Click the Add Button to display the customized field defined dialog box.



[✓] Length Limit Track No. 1 Minimum: 5 Maximum: 99

[✓] Match String Track No. 1 Offset: 0 String: %

Three Method Generating Customized Fields

Flexible Field No. b Track No. 2

Before String Times 1 String

[● Among String] Times1 1 String1 = Times2 1 String2 =

After String Times 0 String Offset 0 Length 0

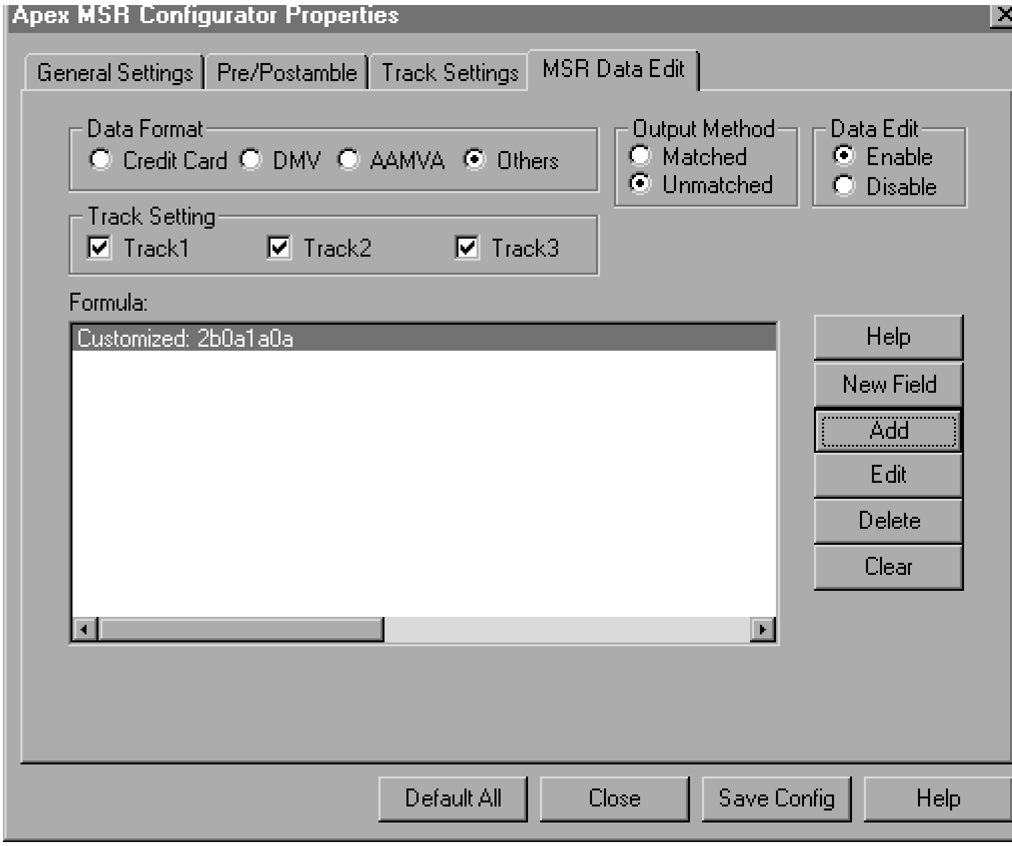
Customized Fields: 1a 2b

Added Fields: +

Fields Sent Sequence:(each field including TrackNo. and FieldNo. Such as 1a) 2b0a1a0a

3. Define the length limit and match string for different tracks, if desired.
4. Select a value from the Flexible Field No pull-down menu**.**
5. Select a value from the Track No. pull-down menu.
6. Select one of the three search methods.
7. Select from customized fields list and added fields list to generate the fields sent sequence.
8. Press OK to accept the new formula for the customized format(others), Press Cancel to exit without saving changes.

The MSR Configurator Properties screen redisplays, with the new custom formula listed in the formula list box.



Data Format: ○ Credit Card ○ DMV ○ AAMVA ● Others

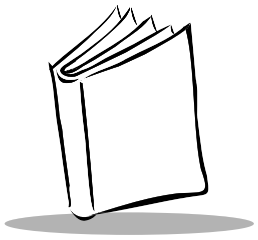
Output Method: ○ Matched ● Unmatched

Data Edit: ● Enable ○ Disable

Track Setting: ☑ Track1 ☑ Track2 ☑ Track3

Formula:

Customized: 2b0a1a0a



Appendix B
Common Magnetic Card Encoding Formats

Credit Card Format

Track1 Format:

Field No.	**Element/Definition**	**Size**
A	Start Sentinel Always "%"	1
B	Format Code Always "B"	1
C	Account Code 13 or 16 characters	13 or16
D	Separator Always "^"	1
E	Cardholder Name	variable
F	Separator Always "^"	1

G	Expiration Date 4 Digits, YYMM Format	4
H	Optional Discretionary Data	variable
I	End Sentinel Always "?"	1
J	LRC	1

Track2 Format:

Field No.	**Element/Definition**	**Size**
A	Start Sentinel Always ";"	1
B	Account Code 13 or 16 characters	13or16
C	Separator Always "="	1
D	Expiration Date 4 Digits, YYMM Format	4
E	Optional Discretionary Data	variable
F	End Sentinel Always "?"	1
G	LRC	1

California Driver's License Format(DMV)

Track1 Format:

Field No.	Element/Definition	Size
A	Start Sentinel 05	1

B	Format Type
	C=Commercial
	S=Salesperson
	D=Driver
	I=Identification
	R=Senior Citizen
		1

C	Name Line1	29
D	Name Line2	29
E	Address Line	29
F	City	13
G	End Sentinel(1fh)	1
H	LRC	1

Track2 Format:

Field No.	Element/Definition	Size
A	Start Sentinel 05	1
B	IDENTIFICATION NUMBER	
	• ANSI User ID	6
	• DL/ID Alpha translated	2
	• 7 position DL/ID number	7
	• Check digit	1
C	Field Separator	1
D	Expiration Date	4
E	Field Separator	1
F	Discretionary Data Field contains eight position Birth Date	8
G	End Sentinel (1 fh)	1
H	LRC	1

Track3 Format:

Field No.	Element/Definition	Size
A	Start Sentinel 05	1
B	Class	4
C	Endorsements	4
D	State Code	2
E	Zip Code	9
F	Sex	1
G	Hair	3
H	Eyes	3
I	Height	3
J	Weight	3
K	Restrictions	10
L	Issue Date	8
M	Office	3
N	Employee ID	2
O	LRE ID	2
P	Fee Due Year	4
Q	Address Line2	28
R	Reserved Space	11
S	End Sentinel	1
T	LRC	1

Driver's License Format Recommended by AAMVA

Track1 Format:

Field No.	Element/Definition	Size
A	Start Sentinel this character must be used at the beginning of the track.	1
B	State or Province(addressee) (Mailing or Residential code) this field will use the ANSI D-20 standard	2
C	City this field should be truncated with a field separator^ if less than13 characters long. If a field separator ^ is used, the "NAME" field follows immediately. EXAMPLE: Bear^	13
D	Name this field should be truncated with a field separator^ if less than 35 characters long. If a field separator is used, the "ADDRESS" field follows immediately. The $ symbol is to be used as a delimiter, between names. EXAMPLE: Roe$Cheryal$A^	35

	EXAMPLE using "city and Name" Bear^Roe$Cheryal$A^ at this point a total of 19 bytes have been used, allowing the remainder to be used for the address.	
E	Address this field has a minimum length of 29 which can be exceeded when utilizing the space from either the city and/or name field. The $ symbol can be used as a delimiter of multiple address lines. EXAMPLE using the City, Name, Address Combination: Bear^Roe$Cheryal $A^123 Something St^	29
F	End Sentinel this character must be the next to the last character of the track.	1
G	LRC this character must be used at the end of the track.	1

Track2 Format:

Field No.	Element/Definition	Size
A	Start Sentinel this character must be used at the beginning of the track.	1
B	ANSI User Code this field is assigned by ANSI for the utilization of Track2	1
C	ANSI User ID this field is the assigned identification number from ISO(International Standards Organization).	5
D	Jurisdiction ID/DL Number this field is used to represent the ID/DL number assigned by each jurisdiction. If less than 13 bytes are used the field is truncated by a field separator character (binary bit string 1101). If 13 bytes are used the field separator character MUST appear in the 14th position. Overflow can be accommodated in the field number 7. It is essential that all users refer to the	14

	Guidelines which follow.	
E	Expiration Date this field will be represented in the following format YYMM. This meets the ISO standard	4
F	Birthdate this field will be represented in the ANSI D-20 Standard YYYYMMDD.	8
G	Remainder of Jurisdictional ID/DL # this field is used to handle the overflow from the jurisdiction ID/DL field. Refer to the Guidelines.	5
H	End Sentinel this character must be the next to the last character of the track.	1
I	LRC this character must be used at the end of the track.	1

Track3 Format:

Field No.	Element/Definition	Size
A	Start Sentinel this character must be used at the beginning of the track.	1
B	Template Version # 1 byte table, value 01-63, this field will be used to store the magnetic stripe version being used. It will be necessary to register the use with AAMVA. Refer to the Guidelines.	1
C	Security Version #. 1 byte table, value 00-63, this field will be used to store the magnetic security version used. 00 represents security is not used. Refer to the Guidelines.	1
D	Postal Code this field will be used to store an 11 position Zip Code or the Canadian postal code. 11 Alphanumeric digits will soon be required to meet postal standards growth. Left justified with spaces filled. Use no hyphens.	11
E	Class	2

	this field will be alphanumeric and will represent the type of license. Use ANSI D-20 codes as modified for CDLIS.	
F	Restrictions this field will be alphanumeric and will use the ANSI D20 standard.	10
G	Endorsements this field will be alphanumeric and will use the ANSI D-20 standard.	4
H	Sex represent as alphanumeric	1
I	Height represented as numeric. See ANSI D-20	3
J	Weight represented as numeric. See ANSI D-20	3
K	Hair Color represented as alphanumeric. See ANSI D-20	3
L	Eye Color represented as alphanumeric. See ANSI D-20	3
M	ID# this field can be utilized by each jurisdiction as needed, but if used it will be necessary	10

	to register the use with AAMVA	
N	Reserved Space this field can be utilized by each jurisdiction as needed, but if used it will be necessary to register the use with AAMVA	16
O	Error Correction this field can be utilized by jurisdiction, but is not a mandatory field. Refer to the Guidelines, Appendix "D" for specific use.	6
P	Security this field for use of each jurisdiction. Refer to the Guidelines.	5
Q	End Sentinel this character must be the next to the last character for the track.	1
R	LRC this character must be used at the end of the track.	1