MOBILE DATA

EDACS CommServ Programmers Guide

CONFIDENTIAL

MS-DOS/ Terminal Interface
for
EDACS
Radio Networks

by

ERICSSON

TRADEMARKS

PREFACE

The EGE Mobile Data Application Programming Interface (MDAPI) is intended for application programmers developing software products that use the RF data features of the Enhanced Digital Access Communications System (EDACS) trunked radio system. Users of this document should be familiar with computer programming and program design concepts.

Hardware and Software Requirements

A software development system requires the following hardware and system software to utilize the tools described in this document:

- An IBM compatible personal computer
- A 720 KB or 1.44 MB 3.5 inch diskette drive
- A hard disk drive
- One or two serial ports (COM1 and/or COM2)
- MS-DOS or PC-DOS 3.0 or higher version
- An ANSI C compiler

Document Organization

This document is divided into ten parts as follows:

- Preface
- Table of Contents
- Chapter 1: Introduction
- Chapter 2: Communications Server (CommServ)
- Chapter 3: Mobile Data Application Library (MDALib)
- Appendix A: Return Levels and Return Codes
- Appendix B: MDALib to CommServ Call Relationships
- Appendix C: Coding Examples
- Appendix D: Return Code to MDALib Relationships
- Index

Chapter 1 provides a general description of MDAPI and its two components, CommServ and MDALib.

Chapter 2 covers the functional operation of CommServ, including program loading and the low level application interface. The six commands used by an application to control the operation of CommServ are described.

Chapter 3 provides an overview of MDALib, and a detailed description of each function included in the Library. The reader should be familiar with the information presented in Chapter 2 before using this chapter.

Appendix A contains the complete list of return levels and return codes, with explanations, comments and suggested actions for handling errors.

Appendix B contains tables showing the relationship between MDALib function calls and the underlying CommServ commands.

Appendix C contains coding examples using MDALib function calls.

Appendix D has a table showing MDALib functions associated with specific return codes.

Glossary of Terms

API

Application Programming Interface - A specification (usually for a specific programming language) of the interface between an application program and a system control program.

CommServ

Communications Server - a low level serial communications program supporting RF data transmission on an EDACS radio system.

DCB

Device Control Block - A data structure containing information used to control a hardware device.

DCE

Data Communications Equipment - Typically a modem, in this case the DCE is the EDACS Radio Data Interface (RDI).

DTE

Data Terminal Equipment - The computer system (Mobile Data Terminal).

MDALib

Mobile Data Application Function Library - A library of software functions callable from C language programs. These functions comprise the application programming interface to CommServ.

MDT

Mobile Data Terminal - A portable computer capable of transmitting data messages via an RF (radio) communications link.

RDI

Radio Data Interface - The EGE proprietary hardware and associated message protocol that allows the connection of an RS-232 port to an EDACS data radio. The RDI may be a separate component, or may be integrated into the radio itself.

TSR

Terminate and Stay Resident - A technique for terminating a DOS program without removing it from system memory. The program is dormant while other programs are executed, but remains in memory and may be activated at any time by a hardware or software interrupt. The term "TSR" often refers to the dormant program itself.

UART

Universal Asynchronous Receiver Transmitter - A hardware device capable of receiving serial data, transposing it to parallel and transmitting it. It is also capable of receiving parallel data and transmitting it as a serial bit stream.

COMPATIBILITY

Because of the compatibility at the application interface level, it is possible to easily port Mobile Data application programs developed for the IBM RF Communications Manager to operate on an EDACS system. EGE wishes to acknowledge the contribution of IBM in the conception of this product and their cooperation in its development.

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

This page intentionally left blank.

CHAPTER 1

INTRODUCTION

1.0 Introduction

The MDAPI software product described in this document is designed to support development of application programs utilizing RF data communications on the EDACS radio system. All hardware and message protocol considerations are hidden from the application developer. MDAPI can be used for program development on IBM PCs and compatible systems and requires the MS-DOS or PC-DOS operating system as a development and runtime environment. MDAPI provides "low-level" interfacing to standard serial asynchronous communication port hardware. Two serial ports are supported using individual hardware interrupts.

MDAPI may be the only communications software required by an application, or it may serve as the bottom layer of a multi-layered communications package. MDAPI is simple, robust, and flexible and provides extensive error reporting. All calls return a severity level and a specific return code. MDAPI consists of two distinct components; CommServ, the Communications Server, and MDALib, the Mobile Data Application Functions Library. These components are described below.

1.1 Communications Server (CommServ)

CommServ is a DOS TSR program, it is not installed as a device driver nor DOS application program. Its function is to process commands from an application program, interfacing directly with the serial port hardware and handling all the details of message framing, acknowledgement, asynchronous character reception, and other standard repetitive tasks. The low level interface to CommServ consists of a Device Control Block, a data structure set up by the application program that controls the behavior of CommServ, and a software interrupt, used by the application to pass control to CommServ.

The basic communication services performed by CommServ are (1) configuring the hardware, (2) establishing a communication link, (3) reading and/or writing messages over the link, and finally (4) restoring the port hardware to its initial state. In addition, CommServ provides reports on port status and message statistics. CommServ uses the following six commands to provide these services:

- **INITIALIZE -** This command is used to initialize the port hardware and software and to set up an optional data tracing facility.
- **OPEN -** This command is used to establish a circuit (connection) to the RDI. OPEN also provides subcommands to enable data tracing and keyboard abort of timed operations.
- **WRITE -** This command is used to send data messages.
- **READ -** This command is used to receive data messages.
- **STATUS -** This command is used to get port status and message statistics reports. It is also used to reset status (clear errors) and reset statistics counters to zero.
- **CLOSE -** This command is used to close the port and restore the original state of the port hardware. It is also used to disable data tracing and keyboard abort.

1.2 Mobile Data Application Functions Library (MDALib)

The second component of this software is a Library of functions that allow an application programmer to utilize the low level interface to CommServ without having to directly manipulate the DCB or issue the software interrupt. By linking with MDALib, C language application programs can perform all communication functions with simple calls to open a port, read and write message data through the port, or obtain port status or message statistics. MDALib contains one direct call to CommServ that takes a pointer to a DCB as an argument. This call allows programmers to accomplish special CommServ functions via the low level interface.

Most of the MDALib functions are "pass-through". The library function sets up the DCB, issues the software interrupt, and then lets CommServ do the work. If an error occurs, CommServ generates the error code and returns it to the library function via the DCB. In a few cases, a library function may do more. For example, the trace functions in MDALib may allocate trace buffer memory in the application's memory space before calling CommServ to initialize and enable data tracing.

The following diagram shows the relationship between user applications, MDALib, CommServ, and the system hardware.

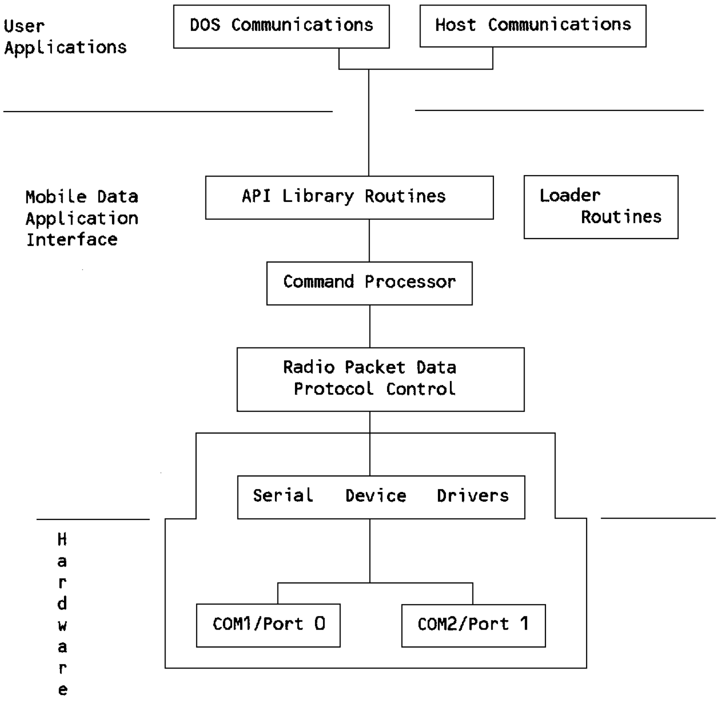

Figure 1. MDAPI Interconnections and Program/Functional Flow

1.6 Storage Considerations.

CommServ requires approximately 32K bytes of system storage. The buffer pool allocation, 20K bytes default, is in addition to the storage required for the operating code. CommServ is designed so that it can be loaded from a batch file or from the DOS prompt.

1.7 Performance Considerations

Asynchronous communications is a character oriented protocol, therefore, receiving data is the highest priority activity. When designing an application program, the number of ports actively receiving, each port's baud rate (9600 baud), and processor speed must be considered. Applications must be designed to allow sufficient processor time to service all serial ports that are active without a loss of characters. Particular attention must be given to the use of the system timer interrupts. The timer interrupt has a higher priority than the communications interrupts on an IBM PC. If the application 'hooks' the timer interrupt, it must make sure that the interrupt routine completes its functions in less than 1 character time, (1 millisecond at 9600 baud) or received characters may be lost.

1.8 Restrictions

CommServ retains exclusive control of all hardware and software interrupt vectors after the application opens the Port until the application closes it. Chaining of interrupts should not be performed. CommServ only attaches itself to a communications port vector when the specific port is initialized by the application. If an application requires the port for another communication package, the application must close the desired port before another program can use it.

CommServ can remain resident in storage while other programs are running and service interrupts for data received on ports remaining active in CommServ.

CHAPTER 2

Communication Server Program (CommServe)

2.1 CommServ Loading and Initial Setup

When CommServ is loaded into memory it reads the command line parameters identified below to allocate the buffer pool; adjust the default parameters based on the communication ports identified by the BIOS power-up-self-test (POST); and copy the default port parameters into the port working parameter areas. The default parameters for each port are maintained in an unchangeable area of CommServ.

CommServ may be loaded into memory via a batch file or from a program. The command line for loading is:

```
COMMSERV /Vnn /Knn /Q /R /I

/Vnn - Vector Number nn = 60 - 67 (in hex)
/Knn - Buffer Pool   nn =  5 - 50 (Decimal blocks of 1K
                bytes)
/Q   - Quiet mode, does not display copyright notice
/R   - Remove CommServ from memory
/I   - Load at specified vector even though vector
                is already used.

NOTE: Command line parameters are not case sensitive!
```

Figure 2. CommServ Command Line Parameters

The Vector Number parameter, /Vnn, defines the user selected Software interrupt vector. CommServ reads this parameter and installs the user interface Software Interrupt Service Routine (ISR) at the chosen vector address. If the command line parameter is not within the 60(hex) - 67(hex) limits CommServ returns a DOS ERRORLEVEL return code of 2. The default software interrupt vector number is 60(hex) if the /Vnn is omitted.

If the specified load vector is already in use, and the /I parameter is not used, CommServ returns a DOS ERRORLEVEL code of 6.

If the /I parameter is used, CommServ loads itself at the specified vector, whether that vector is in use or not. This could result in unpredictable behavior, and is not recommended.

The Buffer Pool parameter, /Knn, indicates to CommServ the total buffer space to be allocated in 1024 byte increments. The default size of the buffer pool is 20480 bytes (20K). Once CommServ is loaded, it must remain fixed in size. CommServ uses buffer pool space for a Rx buffer and a Tx buffer for each port initialized using the INITIALIZE command and subcommand 05. If the user does not specify a large enough buffer pool (default or otherwise), the appropriate return level and return code is returned in the initialization DCB. The user can either reload CommServ with an increased value of /Knn buffer pool allocation on the command line or specify smaller Rx and/or Tx buffer sizes in the initialization parameter structure. The default buffer pool is large enough to initialize any two ports with the default Rx and Tx buffer sizes.

When the user closes a port, it's buffer pool space is freed. Normal buffer pool allocation is used so that several scattered small buffer blocks that have been freed, may not be combined to make one large Rx or Tx buffer unless there is enough contiguous free buffer pool space. An error is returned if the command line parameter is outside the specified limits. If the command line parameter is not within the 5K - 50K limit, CommServ returns a DOS ERRORLEVEL return code of 1.

The parameter "/Q" informs CommServ not to display a banner message at load time.

The /R parameter causes CommServ to close all open ports and remove itself from memory. Unless the /Q parameter is also used, a prompt appears:

```
CommServ will be removed from memory. Continue? (Y/N):
```

informing the user that CommServ is to be removed from memory and asking if the user wants to proceed.

The following DOS ERRORLEVEL codes are returned by CommServ if a problem is detected during the loading phase of the TSR when loaded using a *.BAT file or from a program.

- ERRORLEVEL 0 = CommServ loaded without error.
- ERRORLEVEL 1 = CommServ /Knn command line parameter is out of range.

- ERRORLEVEL 2 = CommServ /Vnn command line parameter is out of range.
- ERRORLEVEL 3 = CommServ command line parameter is not recognized.
- ERRORLEVEL 4 = CommServ Terminate and Stay Resident (TSR) is already installed.
- ERRORLEVEL 5 = CommServ TSR failed.
- ERRORLEVEL 6 = /Vnn vector in use.

The following appears on the display when CommServ is loaded from DOS prompt command line:

- CommServ loaded without error and without the /Q option:

```
******            CommServ Version a.aa            ******
Memory Resident Communications Server for RF Mobile Data
Copyright (c) 1993 Ericsson GE Mobile Communications, Inc.
```

a.aa = CommServ Manager Version (ASCII Characters).

- CommServ /Knn command line parameter is out of range:

```
CommServ Load Error! Invalid Buffer Pool Size Parameter /k3000
Valid range is /K5 - /K50
```

CommServ has detected an invalid Buffer Pool Command line parameter. The valid range is displayed on the error line. The user must reload using this command line parameter set to a valid range.

- CommServ /Vnn command line parameter is out of range.

```
CommServ Load Error! Invalid Interrupt Vector Number Parameter /v20
Valid range is /V60 - /V67
```

CommServ has detected that the /V parameter, Select load vector, is not within the valid range of 60 - 67. The user must reload CommServ with a valid /Vnn parameter.

- CommServ command line parameter(s) not recognized.

```
CommServ Load Error! Unrecognized parameter /y
Valid parameters are /Vnn, /Knn, /Q, /R, /I
```

CommServ has detected an unrecognized command line parameter when loading. The user must reload CommServ with only recognized command line parameters.

- CommServ detected a currently active copy already loaded in memory:

```
CommServ is already installed.
```

An attempt was made to load CommServ with a copy of the program already in memory. CommServ does not load but returns this error.

- Another process is using the interrupt vector number:

```
The specified interrupt vector is owned by another process.
```

CommServ has detected that the vector number specified in the /Vnn parameter is already used by another process. CommServ should be loaded at a different vector, or the /I parameter should be used (not recommended).

- CommServ could not be loaded for an unknown reason:

```
CommServ Load Error!  Unknown reason for load failure.
```

CommServ has attempted to install in memory and there was a problem that could not be determined. CommServ checks its own signature in memory before requesting the DOS Terminate-and-Stay-Resident privilege. If CommServ finds that the signature is wrong it does not remain in memory. The problem cannot be determined.

If CommServ requests TSR privilege and DOS returns to CommServ for any reason, then CommServ returns this error and removes itself from memory. The reason for the DOS return cannot be determined.

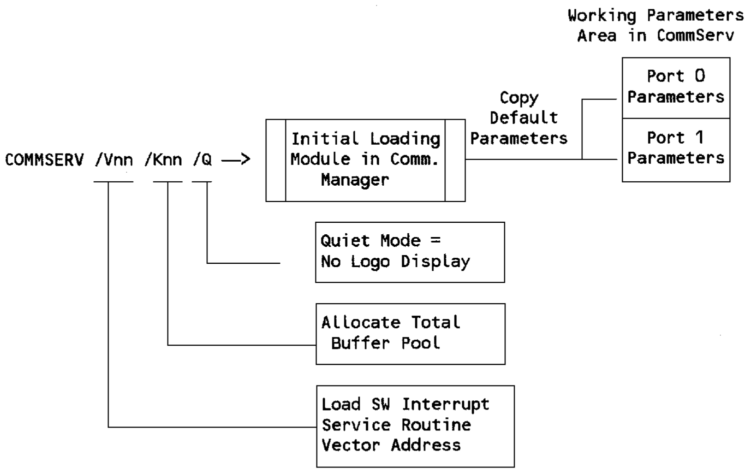
Working Parameters Area in CommServ
Port 0 Parameters
Port 1 Parameters
Copy Default Parameters
Initial Loading Module in Comm. Manager
COMMSERV /Vnn /Knn /Q —>
Quiet Mode = No Logo Display
Allocate Total Buffer Pool
Load SW Interrupt Service Routine Vector Address

Figure 3. CommServ Initial Load Operations

This page intentionally left blank.

CHAPTER 3

MDAPI Communications Library (MDALib)

3.1 Features.

Easy to use ’C’ interface to CommServ.

3.1.1 MDALib Overview

This is a ’C’ language Application Programmer Interface (API) Library to CommServ.

These calls allow the application programmer to utilize CommServ functions without knowing the details of the CommServ Control Interface or its associated structures.

Only a subset of these calls are required to use CommServ. Many calls are only needed to customize CommServ to support non-default environments or just fine-tuning of CommServ parameters.

The Library follows all of the rules defined by the CommServ Control Interface. For example, a port must be opened before it can be read from or written to, and must be initialized before it is opened.

MDALib supports multiple memory models with versions of the library for each memory model. The different versions are distinguished by a character at the end of the library file name as follows.

mdalibt.lib - tiny model
mdalibs.lib - small model
mdalibm.lib - medium model
mdalibc.lib - compact model
mdalibl.lib - large model
mdalibh.lib - huge model

3.1.2 Miscellaneous CommServ Support Calls

The Miscellaneous CommServ Support Calls are described first.

BYTE cs_get_vector_number();
BOOL cs_is_installed();

char far * cs_get_cs_version_number();
char far * cs_get_cs_version_date();
char far * cs_get_api_version_number();
char far * cs_get_api_version_date();

The above calls are not required. They are provided as utilities only.

int cs_set_open_timeout();
int cs_set_write_timeout();

There are no default timeout values for the ***cs_write*** calls. Timeouts must be set before many of the ***cs_write*** calls can be used. See specific calls to see if a timeout value is required.

int cs_enable_user_abort();
int cs_disable_user_abort();

The user abort feature allows the user to abort function calls that use the timeout value by pressing a selectable key sequence.

int cs_reset_status();

This function is used in relationship to the ***cs_get_modem_status*** and ***cs_get_line_status*** function calls.

Note: "BYTE", "BOOL" and "WORD" are not standard ’C’ types. See the MDALIB.H file for the actual type definition. There is a discussion of return codes at the end of this document.

3.1.3 CommServ Port Parameter Calls

During CommServ Port Parameter Calls the Library calls appropriate CS INITIALIZE subcommand to set the proper parameters.

int cs_set_destination_id();
int cs_set_rx_buffer_size();
int cs_set_tx_buffer_size();
int cs_set_defaults();

The following table summarizes the ports and their default settings. The above calls are necessary to change a port parameter from its default setting. The destination ID must be changed from the default setting before a port can be opened.

Port Parameter	Default (All Ports)
Parity	CS_EVEN_PARITY
Stop_Bits	CS_ONE_STOP_BIT
Word_Length	CS_EIGHT_BIT_WORD
Destination_ID	0000
Rx_Buffer_Size	6000
Tx_Buffer_Size	3000
Baud	9600

Figure 24. Port Parameter Defaults

3.1.4 CommServ Communication Calls

During CommServ Communication Calls the Library makes an appropriate call to CommServ to do the actual function requested.

int cs_init();
int cs_open_rf();

int cs_write_msg_wait_ack();
int cs_write_addr_msg_wait_ack();
int cs_write_msg_ignore_ack();
int cs_write_addr_msg_ignore_ack();
int cs_write_msg_no_wait();
int cs_write_addr_msg_no_wait();
int cs_write_msg_ack_status();
int cs_cancel_write_msg_no_wait();

```
int cs_read_addr_msg ();
int cs_read_msg ();
int cs_close ();
int cs_remove ();

int cs_interrupt();
```

Note: The ***cs_read_addr_msg*** and ***cs_read_msg*** function calls use **buffer_size**, but all the other calls use **buffer_count.** The **buffer_size** variable gives the size of the buffer or memory area, but **buffer_count** refers to the actual number of bytes being stored at that memory location. **buffer_size** is always larger than (or equal) to **buffer_count** when referring to the same memory area. In this call the application doesn't know exactly how large the incoming message is, so it must provide a larger than necessary buffer space to hold it.

These are the calls that do the real work of communication. The application programmer must use these calls in a specific order.

Once the port has been initialized with the ***cs_init*** function call, the port can be opened with the ***cs_open_rf*** function call. Remember to set the **open_timeout** with the ***cs_set_open_timeout*** call if a timeout longer than the default is required.

Notes:

After a connection is made with the other side, the ***cs_read*** and ***cs_write*** calls are used to transfer data through the communications port.

Remember to set **write_timeout** with the ***cs_set_write_timeout*** call before issuing a write command requiring a timeout.

When communications are completed the port is closed with the ***cs_close*** function call.

The following table summarizes the various read and write commands.

CommServ Library Call	Transfer Type	Timeout and Abort	Read Type	Maximum Transfer
cs_write_msg_wait_ack	Message	Yes	n/a	2000(3)
cs_write_addr_msg_wait_ack	Addr Msg	Yes	n/a	2000
cs_write_msg_ignore_ack(2)	Message	Yes	n/a	2000(3)
cs_write_addr_msg_ignore_ack(2)	Addr Msg	Yes	n/a	2000
cs_write_msg_no_wait(2)	Message	Yes(1)	n/a	2000(3)
cs_write_addr_msg_no_wait(2)	Addr Msg	Yes(1)	n/a	2000
cs_write_msg_ack_status	Message	No(1)	n/a	n/a
cs_cancel_write_msg_no_wait	Message	No	n/a	n/a
cs_read_msg(3,4)	Message	No	Complete	Rx_Buffer_Size
cs_rx_trace_read	Character	No	Partial	Rx_Trace_Size
cs_tx_trace_read	Character	No	Partial	Tx_Trace_Size
cs_read_addr_msg(3)	Addr Msg	No	Complete	512

Figure 25. Read/Write Summary

Notes:

1. The timeout from the ***cs_write_msg_no_wait***, or ***cs_write_addr_msg_no_wait*** calls is extended after CommServ returns and is used in conjunction with the ***cs_write_msg_ack_status*** call that must follow.

2. CommServ returns when all the characters have been put in the Tx buffer or when the timeout has expired. The application can determine the amount of available space in the Tx buffer by using the ***cs_get_tx_buffer_count*** call and then subtracting from the **tx_buffer_size** set with the ***cs_set_tx_buffer_size*** call (or the default). It should be noted that writes larger than the **tx_buffer_size** can be made. However, the timeout value must be large enough to allow all characters (at the port's current Baud rate) to be sent and an ACK, if expected, to be returned within the timeout or CommServ returns an error. If the timeout is not long enough, at the time the error is reported part or all of the message is normally transmitted leaving the connection in an unknown state.

3. Because of the way flow control operates in CommServ, the **rx_buffer_size** should be selected carefully. Flow control is implemented at the message level, that is, there must be room in the receive buffer for the maximum size message plus protocol overhead (a total of 531 bytes) or CommServ will indicate to the RDI that no more data is to be transmitted. To calculate the proper Rx buffer size, take the average expected message size, add 18 bytes, multiply by the number of messages the buffer must hold between reads, then add 531 bytes.

3.1.5 CommServ Trace Calls

During CommServ Trace Calls the Library and CommServ share the work. CommServ is called to setup the tracing environment and actually put the data into the trace buffers. The Library handles the memory allocation for the application and performs the reads from the trace buffers similar to the way cs_read calls read characters from the communication buffers.
Note: The Library has default trace buffers. If more than 128 bytes of trace are required, then the ***cs_alloc_trace_buffers*** function call allocates more space.

```
int cs_alloc_trace_buffers();
int cs_init_trace();
int cs_start_trace();
int cs_rx_trace_read();
int cs_tx_trace_read();
int cs_stop_trace();
int cs_free_trace_buffers();
```

These functions calls can be used to verify the data being sent and received at the CommServ transmit and receive buffers. Again, the application programmer must use these calls in a specific order.

Even before the port has been opened with the appropriate ***cs_open*** call, but after the port has been initialized by ***cs_init***, the trace facility can be utilized. First, the ***cs_alloc_trace_buffers*** function call allocates buffers to contain data gathered by CommServ during the trace following reads and writes. The ***cs_init_trace*** function call tells CommServ where to put the data.

Use the ***cs_start_trace*** function call to start tracing and the ***cs_stop_trace*** function call to stop tracing. The trace can be started and stopped independently of the data being transferred by reads and writes so that specific transfers can be traced and others ignored.

Use the ***cs_rx_trace_read*** and ***cs_tx_trace_read*** function calls to retrieve data captured during reads and writes. These calls should be made frequently enough to avoid data being overrun in the trace buffers. No error is reported if the trace buffer is overrun. The sizes of the trace buffers are set during the ***cs_alloc_trace_buffers*** function call. Both transmit and receive are traced when tracing is enabled; however, the application can ignore data received if it is only interested in what is being captured during transmission. To free up the trace buffers after they are used, use the ***cs_free_trace_buffers*** function call.

3.1.6 CommServ Status and Statistics Calls

During CommServ Status Calls the Library calls CommServ Status Command to retrieve the desired status information or message statistics.

<u>Status Calls</u>

```
BYTE cs_get_state();
BYTE cs_get_rx_msg_count();
WORD cs_get_options();
BYTE cs_get_modem_status();
BYTE cs_get_line_status();
WORD cs_get_rx_buffer_count();
WORD cs_get_tx_buffer_count();
```

<u>Statistics Calls</u>

```
WORD cs_get_rx_count();
WORD cs_get_rx_errors();
WORD cs_get_tx_count();
WORD cs_get_tx_errors();
WORD cs_get_tx_retries();
```

Status calls return specific information that may be useful to the application program. For example, if the ***cs_get_rx_msg_count*** indicates that no messages are waiting, then a ***cs_read_msg*** function call is unnecessary.

3.2 MDALib Call Format

Each call supported by the MDAPI Library is described in detail using the following format:

- **SUMMARY OF CALL**

The syntax of the call is described including the type of return information and parameters, if any.

All calls show that the file MDALIB.H should be included in the source file. This is needed for the prototype definition. For some calls the parameters are constants and they are defined in the MDALIB.H file.

- **DESCRIPTION OF FUNCTION**

The function of the call is described in detail.

- **DESCRIPTION OF PARAMETERS**

The parameters of the call are described.

On some calls a pointer is passed as a parameter so that a variable in the application area can be updated.

- **RETURN VALUES**

Many calls to the Library return an integer value. This value indicates the completion status or return level for the call. These levels range from successful to fatal with several levels in between. A global variable, **cs_rtn_level**, is set to this value. If a call was unsuccessful a global variable, **cs_rtn_code**, is set to indicate the specific error condition that generated this level. If a call was successful **cs_rtn_code** indicates success. Refer to the RETURN LEVELs and RETURN CODEs section of this document for a detailed description of the return values.

All of the ***cs_get*** calls, and the ***cs_is_installed*** call, return the desired information as the return value (the return types BOOL, BYTE, and WORD are defined in MDALIB.H). If an error occurred during the call, the return value is Hex FF or Hex FFFF depending on the size of the expected return value. If the Library returns a far pointer, (for instance, ***cs_get_cs_version_number***, or ***cs_get_cs_version_date***) or if an error occurs, the return value is a null pointer. In either case the two global values are set to the indicated error. **cs_return_level** is set to indicate the level of the error and **cs_rtn_code** again indicates the specific error condition.

Some invalid parameters passed during Port Parameter Calls (***cs_set*** calls) cannot be found until a ***cs_init*** call verifies the port parameter settings.

Most errors are generated by CommServ and not the MDAPI Library. These errors are passed back directly without the Library's intervention. When the Library generates the error codes they are returned in the same fashion as with CommServ, with return levels and return codes.

3.3 MDALib Calls

cs_alloc_trace_buffers()

Syntax of Function

```
#include "mdalib.h"
int far cs_alloc_trace_buffers (BYTE port_number,
      WORD rx_trace_size,
      WORD tx_trace_size);
```

Description of Function

This function allocates space from DOS to be used for the buffers that trace the receive and transmit lines of the port indicated.

Description of Parameters

port_number indicates the communications port for the function. Valid port values are COM1 (serial port 0) and COM2 (serial port 1). The **rx_trace_size** is the size of the receive trace buffer and must be from 2 to 65535 KB. The **tx_trace_size** is the size of the transmit trace buffer and must also be from 2 to 65535 KB. These buffers should be large enough to hold the amount of data saved between each ***cs_rx_trace_read*** or ***cs_tx_trace_read*** function call.

Return Values

The return value is zero if the operation was successful. A non-zero value is returned if the trace buffers are already allocated or if there is not enough memory available to allocate the buffers. The global variables **cs_rtn_level** and **cs_rtn_code** are set with the return level and return code. (See "Return Levels and Return Codes" in Appendix A.)

cs_cancel_write_msg_no_wait()

Syntax of Function

```
#include "mdalib.h"
int far cs_cancel_write_msg_no_wait (BYTE port_number);
```

Description of Function

This call cancels the **WAITING_FOR_ACK** state that the port was put into by ***cs_write_msg_no_wait***. The ***cs_write_msg_no_wait*** function returns to the application before the ACK/NAK arrives, therefore CommServ waits for the write_timeout period before returning an error. This function forces CommServ to stop waiting for the ACK and return to an open state.

Description of Parameters

port_number indicates the communications port for the function. Valid port values are COM1 (serial port 0) and COM2 (serial port 1).

Return Values

The return value is zero if the operation is successful. If an error occurs during execution of the function, the global variables **cs_rtn_level** and **cs_rtn_code** are set with an error level and code, and the function returns the value of **cs_rtn_level**.

cs_close()

Syntax of Function

```
#include "mdalib.h"
int far cs_close (BYTE port_number);
```

Description of Function

The port indicated is closed. All characters waiting to be transmitted in the Tx buffer and all characters waiting to be read in the Rx buffer are discarded.

Description of Parameters

port_number indicates the communications port for the function. Valid port values are COM1 (serial port 0) and COM2 (serial port 1).

Return Values

The return value is zero if the operation is successful. If an error occurs during execution of the function, the global variables **cs_rtn_level** and **cs_rtn_code** are set with an error level and code, and the function returns the value of **cs_rtn_level**.

cs_disable_user_abort()

Syntax of Function

```
#include "mdalib.h"
int far cs_disable_user_abort (void);
```

Description of Function

This function disables the key sequence that can be used to abort calls that use a timeout. It disables the feature for all ports. See **cs_enable_user_abort**.

Description of Parameters

None

Return Values

The return value is zero if the operation is successful. If an error occurs during execution of the function, the global variables **cs_rtn_level** and **cs_rtn_code** are set with an error level and code, and the function returns the value of **cs_rtn_level**.

cs_enable_user_abort()

Syntax of Function

```
#include "mdalib.h"
int far cs_enable_user_abort (void far *key_pattern,
     WORD pattern_count);
```

Description of Function

This function sets and enables the **key_pattern** that can abort calls with timeouts. The **key_pattern** is a series of characters that CommServ looks for in the keyboard buffer while waiting for a timed event to be completed. At least one of the ports must be initialized when this command is called. CommServ leaves the **key_pattern** in the keyboard buffer, so the application must deal with the abort pattern once the aborted function returns. The active **key_pattern** can be changed at any time without first disabling it. The user abort feature, once enabled, is active for all ports. You cannot have a different **key_pattern** for each port.

Description of Parameters

The **key_pattern** is a far pointer to a data buffer containing the sequence. The **pattern_count** value represents the number of bytes in the **key_pattern**, and must be from 1 to 4.

Return Values

The return value is a zero if the operation is successful. A non-zero error code is returned if the **key_pattern** pointer is null, **pattern_count** is zero or **pattern_count** exceeds four bytes. Error level and code are returned in the global variables **cs_rtn_level** and **cs_rtn_code**, and the function returns the value of **cs_rtn_level**.

cs_free_trace_buffers()

Syntax of Function

```
#include "mdalib.h"
int far cs_free_trace_buffers (BYTE port_number);
```

Description of Function

This function frees the buffers allocated in the **cs_alloc_trace_buffers** call.

Description of Parameters

port_number indicates the communications port for the function. Valid port values are COM1 (serial port 0) and COM2 (serial port 1).

Return Values

The return value is zero if the operation is successful. If an error occurs in the execution of the function, the global variables **cs_rtn_level** and **cs_rtn_code** are set with an error level and code, and the function returns the value of **cs_rtn_level**.

cs_get_api_version_date()

Syntax of Function

```
#include "mdalib.h"
char far *far cs_get_api_version_date (void);
```

Description of Function

This function returns a far pointer to an ASCII string containing the API version date.

Description of Parameters

None

Return Values

The return value is a far pointer to an ASCII string containing the API version date. If an error occurs in the execution of the function, the global variables **cs_rtn_level** and **cs_rtn_code** are set with an error level and code, and a null pointer is returned.

cs_get_api_version_number()

Syntax of Function

```
#include "mdalib.h"
char far *far cs_get_api_version_number (void);
```

Description of Function

This function returns a far pointer to an ASCII string containing the API version number.

Description of Parameters

None

Return Values

The return value is a far pointer to an ASCII string containing the API version number. If an error occurs in the execution of the function, the global variables **cs_rtn_level** and **cs_rtn_code** are set with an error level and code, and a null pointer is returned.

cs_get_cs_version_date()

Syntax of Function

```
#include "mdalib.h"
char far *far cs_get_cs_version_date (void);
```

Description of Function

This function returns a far pointer to an ASCII string containing the CommServ version date.

Description of Parameters

None

Return Values

The return value is a far pointer to an ASCII string containing the API version number. If an error occurs in the execution of the function, the global variables **cs_rtn_level** and **cs_rtn_code** are set with an error level and code, and a null pointer is returned.

cs_get_cs_version_number()

Syntax of Function

```
#include "mdalib.h"
char far *far cs_get_cs_version_number (void);
```

Description of Function

This function returns a far pointer to an ASCII string containing the CommServ version number.

Description of Parameters

None

Return Values

The return value is a far pointer to an ASCII string containing the API version number. If an error occurs in the execution of the function, the global variables **cs_rtn_level** and **cs_rtn_code** are set with an error level and code, and a null pointer is returned.

cs_get_line_status()

Syntax of Function

```
#include "mdalib.h"
BYTE far cs_get_line_status (BYTE port_number);
```

Description of Function

This function returns the contents of the accumulated line status in CommServ. See the ***cs_reset_status*** function call.

Description of Parameters

port_number indicates the communications port for the function. Valid port values are COM1 (serial port 0) and COM2 (serial port 1).

Return Values

The return value is the line status. The byte is bit oriented, definitions are as follows.

Bit 7	Line - Error in Receiver FIFO Register (16550 FIFO buffered UART)
Bit 6	Line - Transmitter Shift Register Empty
Bit 5	Line - Transmitter Holding Register Empty
Bit 4	Line - Break Interrupt
Bit 3	Line - Framing Error
Bit 2	Line - Parity Error
Bit 1	Line - Overrun Error
Bit 0	Line - Data Ready

If an error occurs in the execution of the function, the global variables **cs_rtn_level** and **cs_rtn_code** are set with an error level and code, and a value of FF(hex) is returned.

cs_get_modem_status()

Syntax of Function

```
#include "mdalib.h"
BYTE far cs_get_modem_status (BYTE port_number);
```

Description of Function

This function returns the contents of the accumulated modem status in CommServ. See the ***cs_reset_status*** call.

Description of Parameters

port_number indicates the communications port for the function. Valid port values are COM1 (serial port 0) and COM2 (serial port 1).

Return Values

The return value is the modem status. The byte is bit oriented, definitions are as follows.

Bit 7	Data Carrier Detect
Bit 6	Ring Indicator
Bit 5	Data Set Ready
Bit 4	Clear to Send
Bit 3	Delta Carrier Detect
Bit 2	Trailing Edge Ring Indicator
Bit 1	Delta Data Set Ready
Bit 0	Delta Clear to Send

If an error occurs during execution of the function the global variables **cs_rtn_level** and **cs_rtn_code** are set with an error level and code, and a value of FF(hex) is returned.

cs_get_options()

Syntax of Function

```
#include "mdalib.h"
WORD far cs_get_options (BYTE port_number);
```

Description of Function

This function returns the current state of the trace and user abort.

Description of Parameters

port_number indicates the communications port for the function. Valid port values are COM1 (serial port 0) and COM2 (serial port 1).

Return Values

The return value is the current state of the trace and user abort. The word is bit oriented, definitions are as follows.

Bit 15	Rx Trace Initialized
Bit 14	Rx Trace Started
Bit 13	Tx Trace Initialized
Bit 12	Tx Trace Started
Bit 11	User Abort Enabled
Bit 10-0	Reserved = 0

If an error occurs in the execution of the function, the global variables **cs_rtn_level** and **cs_rtn_code** are set with an error level and code, and a value of FFFF(hex) is returned.

cs_get_rx_buffer_count()

Syntax of Function

```
#include "mdalib.h"
WORD far cs_get_rx_buffer_count (BYTE port_number);
```

Description of Function

This function returns the number of characters waiting to be read from the Rx buffer. The Rx buffer count includes an overhead of 18 bytes for each message in the buffer.

Description of Parameters

port_number indicates the communications port for the function. Valid port values are COM1 (serial port 0) and COM2 (serial port 1).

Return Values

The return value is the number of characters waiting in the Rx buffer. If an error occurs in the execution of the function, the global variables **cs_rtn_level** and **cs_rtn_code** are set with an error level and code, and a value of FFFF(hex) is returned.

cs_get_rx_count()

Syntax of Function

```
#include "mdalib.h"
WORD far cs_get_rx_count (BYTE port_number);
```

Description of Function

This function returns the number of messages successfully received since opening the port, or since the last call to reset statistics [***cs_reset_msg_stats();***].

Description of Parameters

port_number indicates the communications port for the function. Valid port values are COM1 (serial port 0) and COM2 (serial port 1).

Return Values

The return value is the number of messages received. If an error occurs during execution of the function, the global variables **cs_rtn_level** and **cs_rtn_code** are set with an error level and code, and a value of FFFF(hex) is returned.

cs_get_rx_errors()

Syntax of Function

```
#include "mdalib.h"
WORD far cs_get_rx_errors (BYTE port_number);
```

Description of Function

This function returns the number of receive errors that have occurred since the port was opened, or since the last call to reset statistics [***cs_reset_msg_stats()***].

Description of Parameters

port_number indicates the communications port for the function. Valid port values are COM1 (serial port 0) and COM2 (serial port 1).

Return Values

The return value is the number of receive errors. If an error occurs during execution of the function, the global variables **cs_rtn_level** and **cs_rtn_code** are set with an error level and code, and a value of FFFF(hex) is returned.

cs_get_rx_msg_count()

Syntax of Function

```
#include "mdalib.h"
BYTE far cs_get_rx_msg_count (BYTE port_number);
```

Description of Function

This function returns the number of whole messages waiting to be read from the Rx buffer.

Description of Parameters

port_number indicates the communications port for the function. Valid port values are COM1 (serial port 0) and COM2 (serial port 1).

Return Values

The return value is the number of messages waiting in the Rx buffer. If an error occurs during execution of the function, the global variables **cs_rtn_level** and **cs_rtn_code** are set with an error level and code, and a value of FF(hex) is returned. CommServ reports up to 254 messages. If more than that exist, only 254 are indicated. The actual count is reported when the message count is below 255.

cs_get_state()

Syntax of Function

```
#include "mdalib.h"
BYTE far cs_get_state (BYTE port_number);
```

Description of Function

This function return the current port state.

Description of Parameters

port_number indicates the communications port for the function. Valid port values are COM1 (serial port 0) and COM2 (serial port 1).

Return Values

The port state values shown below are defined in MDALIB.H.

CLOSED - The port is currently closed.

INITIALIZED - The port is currently initialized.

OPENED - The port is currently open.

WAITING_FOR_ACK - The API was called with the ***cs_write_msg_no_wait*** and CommServ is waiting for an ACK.

If an error occurs during execution of the function, the global variables **cs_rtn_level** and **cs_rtn_code** are set with an error level and code, and a value of FF(hex) is returned.

cs_get_tx_buffer_count()

Syntax of Function

```
#include "mdalib.h"
WORD far cs_get_tx_buffer_count (BYTE port_number);
```

Description of Function

This function returns the number of characters waiting to be transmitted from the Tx buffer. This number includes 18 overhead bytes for each complete message.

Description of Parameters

port_number indicates the communications port for the function. Valid port values are COM1 (serial port 0) and COM2 (serial port 1).

Return Values

The return value is the number of characters waiting in the Tx buffer. If an error occurs during execution of the function, the global variables **cs_rtn_level** and **cs_rtn_code** are set with an error level and code, and a value of FFFF(hex) is returned.

cs_get_tx_count()

Syntax of Function

```
#include "mdalib.h"
WORD far cs_get_tx_count (BYTE port_number);
```

Description of Function

This function returns the number of messages successfully transmitted since the port was opened, or since the last call to reset statistics [***cs_reset_msg_stats()***].

Description of Parameters

port_number indicates the communications port for the function. Valid port values are COM1 (serial port 0) and COM2 (serial port 1).

Return Values

The return value is the accumulated number of messages transmitted. If an error occurs during execution of the function, the global variables **cs_rtn_level** and **cs_rtn_code** are set with an error level and code, and a value of FFFF(hex) is returned.

cs_get_tx_errors()

Syntax of Function

```
#include "mdalib.h"
WORD far cs_get_tx_errors (BYTE port_number);
```

Description of Function

This function returns the number of failed message transmissions that have occurred since the port was opened, or since the last call to reset statistics [***cs_reset_msg_stats()***].

Description of Parameters

port_number indicates the communications port for the function. Valid port values are COM1 (serial port 0) and COM2 (serial port 1).

Return Values

The return value is the accumulated number of transmission failures. If an error occurs during execution of the function, the global variables **cs_rtn_level** and **cs_rtn_code** are set with an error level and code, and a value of FFFF(hex) is returned.

cs_get_tx_retries()

Syntax of Function

```
#include "mdalib.h"
WORD far cs_get_tx_retries (BYTE port_number);
```

Description of Function

This function returns the number of attempts to transmit a message since the port was opened, or since the last call to reset statistics [***cs_reset_msg_stats()***]. This number is the sum of all successful and unsuccessful attempts.

Description of Parameters

port_number indicates the communications port for the function. Valid port values are COM1 (serial port 0) and COM2 (serial port 1).

Return Values

The return value is the accumulated number of transmission attempts. If an error occurs during execution of the function, the global variables **cs_rtn_level** and **cs_rtn_code** are set with an error level and code, and a value of FFFF(hex) is returned.

cs_get_vector_number()

Syntax of Function

```
#include "mdalib.h"
BYTE far cs_get_vector_number (void);
```

Description of Function

This function determines if CommServ is loaded into memory as a TSR program, and if it is, returns the interrupt vector for the program.

When CommServ is loaded into memory the interrupt vector number is specified as an argument on the DOS command line. CommServ defaults to interrupt 60(hex). Valid interrupt vectors for CommServ are 60(hex) through 67(hex).

Description of Parameters

None

Return Values

The return value is the interrupt vector for CommServ. If an error occurs during execution of the function, the global variables **cs_rtn_level** and **cs_rtn_code** are set with an error level and code, and a value of FF(hex) is returned.

cs_init()

Syntax of Function

```
#include "mdalib.h"
int far cs_init (BYTE port_number);
```

Description of Function

The port indicated is initialized with the current port parameters. Original port parameters are set with ***cs_set*** commands. ***cs_set_destination_id()*** must be called before the port can be initialized. All other port parameters have acceptable default values. A port must be initialized before it can be opened.

Description of Parameters

port_number indicates the communications port for the function. Valid port values are COM1 (serial port 0) and COM2 (serial port 1).

Return Values

The return value is a zero if the operation was successful. If an error occurs during execution of the function, the global variables

cs_rtn_level and **cs_rtn_code** are set with an error level and code, and the function returns the value of **cs_rtn_level**.

cs_init_data_groups()

Syntax of Function

```
#include "mdalib.h"
int far cs_init_data_groups (BYTE port_number,
     int far *gids,
     WORD far *actual_count);
```

Description of Function

This function is used to send a Data Group Initialization message to an EDACS mobile radio. The Data Group Initialization message programs the radio to accept group data calls directed to the specified Group IDs. The message is not transmitted over the air by the radio. If the radio has not been programmed with a Data Group Initialization message it can only receive group data messages directed to Group 0. The radio stores the data groups in non-volitle memory, so the programming only has to be done once unless a change is required.

Description of Parameters

port_number indicates the communications port for the function. Valid port values are COM1 (serial port 0) and COM2 (serial port 1).

gids is a far pointer to an array of integers that are valid Group ID numbers. The value of each integer must be in the range 0 to 2047. The array may contain up to 16 IDs terminated with a zero. Non-zero values following the first zero in the array are ignored.

Return Values

actual_count contains the number of bytes transferred to the Tx buffer, and should always be 33. The function returns a value of zero if successful. If an error occurs during execution of the function, the global variables **cs_rtn_level** and **cs_rtn_code** are set with an error level and code, and the function returns the value of **cs_rtn_level**.

cs_init_trace()

Syntax of Function

```
#include "mdalib.h"
int far cs_init_trace (BYTE port_number);
```

Description of Function

This function initializes CommServ with the location of the trace buffers. The port must be initialized using ***cs_init*** before using this call.

Description of Parameters

port_number indicates the communications port for the function. Valid port values are COM1 (serial port 0) and COM2 (serial port 1).

Return Values

The return value is a zero is the operation was successful. If an error occurs during execution of the function, the global variables **cs_rtn_level** and **cs_rtn_code** are set with an error level and code, and the function returns the value of **cs_rtn_level**.

cs_interrupt()

Syntax of Function

```
#include "mdalib.h"
int far cs_interrupt(DCB far *dcb_ptr);
```

Description of Function

This function calls CommServ with information provided in the DCB. The programmer is responsible for setting the entire DCB. See Chapter 2 Communication Server Program on page .

Description of Parameters

The **dcb_ptr** is a far pointer to a DCB set up by the programmer.

Return Values

The return value is a zero is the operation was successful. If an error occurs during execution of the function, the global variables **cs_rtn_level** and **cs_rtn_code** are set with an error level and code, and the function returns the value of **cs_rtn_level**.

cs_is_installed()

Syntax of Function

```
#include "mdalib.h"
BOOL far cs_is_installed (void);
```

Description of Function

This function determines if CommServ is loaded into memory as a TSR program.

Description of Parameters

None

Return Values

The return value is TRUE (1) if CommServ is installed. If CommServ is not installed the global variables **cs_rtn_level** is set to INVALID_CALL, **cs_rtn_code** is set to LB_NO_CS_PRESENT, and the return value is FALSE (0).

cs_open_rf()

Syntax of Function

```
#include "mdalib.h"
int far cs_open_rf (BYTE port_number);
```

Description of Function

The port indicated is opened and the radio data interface is put online. ***cs_set_destination_id()*** and ***cs_init()*** must be called before this function is called.

Description of Parameters

port_number indicates the communications port for the function. Valid port values are COM1 (serial port 0) and COM2 (serial port 1).

Return Values

The return value is a zero if the operation was successful. If an error occurs during execution of the function, the global variables **cs_rtn_level** and **cs_rtn_code** are set with an error level and code, and the function returns the value of **cs_rtn_level**.

cs_read_addr_msg()

Syntax of Function

```
#include "mdalib.h"
int far cs_read_addr_msg (BYTE port_number,
    BYTE far *type,
    WORD far *id,
    void far * buffer,
    WORD buffer_size,
    WORD far *actual_count);
```

Description of Function

A message, if available, up to the **buffer_size** number of characters is read from the RX buffer of the port indicated, and copied to the memory pointed to by buffer.

Description of Parameters

port_number indicates the communications port for the function. Valid port values are COM1 (serial port 0) and COM2 (serial port 1).

buffer is a far pointer to the buffer for receiving the data.

buffer_size is the size of the data buffer. Since the application does not know how large the message is, the buffer must be larg enough to hold the largest expected message. The message may range in size from 1 to 512 bytes.

Return Values

type indicates the ID type of the call originator. This value should always be IS_LID (2). **id** indicates the individual logical ID (LID) of the call originator. **actual_count** indicates the actual number of bytes copied from the RX buffer. The function value is zero if the operation was successful. If an error occurs during execution of the function the global variables **cs_rtn_level** and **cs_rtn_code** are set with an error level and code, and the function returns **cs_rtn_level**.

A return level of SOFT_ERROR and a return code of NO_MESSAGES indicates that the RX buffer contains no complete messages, but does not imply an error in the operation of CommServ.

cs_read_msg()

Syntax of Function

```
#include "mdalib.h"
int far cs_read_msg (BYTE port_number,
     void far *buffer,
     WORD buffer_size,
     WORD far *actual_count);
```

Description of Function

A message, if available, up to the **buffer_size** number of characters is read from the Rx buffer of the port indicated, and copied to memory pointed to by **buffer**.

Description of Parameters

port_number indicates the communications port for the function. Valid port values are COM1 (serial port 0) and COM2 (serial port 1).

buffer is a far pointer to the data buffer for receiving the data. **buffer_size** is the size of the data buffer to receive the message. Since the application does not know how large the message is, the buffer must be big enough to hold the largest expected message. The message may range in size from 1 to 512 bytes.

Return Values

actual_count indicates the actual number of bytes copied from the Rx buffer. The return value is zero if the operation was successful. If an error occurs during execution of the function, the global variables **cs_rtn_level** and **cs_rtn_code** are set with an error level and code, and the function returns the value of **cs_rtn_level**.

A return level of SOFT_ERROR and return code of NO_MESSAGES indicates that the Rx buffer was empty, but does not imply an error in the operation of CommServ.

cs_remove()

Syntax of Function

```
#include "mdalib.h"
int far cs_remove (void);
```

Description of Function

CommServ closes all open ports and removes itself from memory.

Description of Parameters

None

Return Values

The return value is a zero is the operation was successful. If an error occurs during execution of the function, the global variables **cs_rtn_level** and **cs_rtn_code** are set with an error level and code, and the function returns the value of **cs_rtn_level**.

cs_reset_msg_stats()

Syntax of Function

```
#include "mdalib.h"
int far cs_reset_msg_stats (BYTE port_number);
```

Description of Function

The statistics for the port indicated are reset to zero. See ***cs_get_rx_count(), cs_get_rx_errors(), cs_get_tx_count(), cs_get_tx_errors(), and cs_get_tx_retries()***.

Description of Parameters

port_number indicates the communications port for the function. Valid port values are COM1 (serial port 0) and COM2 (serial port 1).

Return Values

The return value is a zero if the operation was successful. If an error occurs during execution of the function, the global variables **cs_rtn_level** and **cs_rtn_code** are set with an error level and code, and the function returns the value of **cs_rtn_level**.

cs_reset_status()

Syntax of Function

```
#include "mdalib.h"
int far cs_reset_status (BYTE port_number);
```

Description of Function

The modem_status and line_status for the port indicated are reset to zero. The Rx buffer overflow flag is reset to FALSE (0).

cs_read calls return detected line errors. The status must be reset so that future ***cs_reads*** do not return the same error.

Description of Parameters

port_number indicates the communications port for the function. Valid port values are COM1 (serial port 0) and COM2 (serial port 1).

Return Values

The return value is a zero if the operation was successful. If an error occurs during execution of the function, the global variables **cs_rtn_level** and **cs_rtn_code** are set with an error level and code, and the function returns the value of **cs_rtn_level**.

cs_rx_trace_read()

Syntax of Function

```
#include "mdalib.h"
int far cs_rx_trace_read (BYTE port_number,
     void far *buffer,
     WORD buffer_count,
     WORD far *actual_count);
```

Description of Function

Characters are read from the Rx trace buffer and are copied to a memory location pointed to by **buffer**. Up to the **buffer_count** of characters are transferred. The Rx trace buffer must be read before the buffer overflows or the oldest data is lost.

Description of Parameters

port_number indicates the communications port for the function. Valid port values are COM1 (serial port 0) and COM2 (serial port 1).

buffer is a far pointer to the area to receive the Rx trace data. **buff-er_count** is the number of bytes requested from the Rx trace buffer. **buffer_count** must be in the range of 1 to **rx_trace_size**.

Return Values

The **actual_count** field indicates the actual number of bytes copied from the Rx trace buffer. The function returns a zero if the operation was successful. If an error occurs during execution of the function, the global variables **cs_rtn_level** and **cs_rtn_code** are set with an error level and code, and the function returns the value of **cs_rtn_level**.

cs_set_defaults()

Syntax of Function

```
#include "mdalib.h"
int far cs_set_defaults (BYTE port_number);
```

Description of Function

This function returns the designated port to the default (CommServ load time) port parameter setting. The port must be closed when this command is executed. After issuing this call, ***cs_set_destination_id()*** must be called before the port can be initialized or opened.

Description of Parameters

port_number indicates the communications port for the function. Valid port values are COM1 (serial port 0) and COM2 (serial port 1).

Return Values

The return value is zero if the operation was successful. If an error occurs during execution of the function, the global variables **cs_rtn_level** and **cs_rtn_code** are set with an error level and code, and the function returns the value of **cs_rtn_level**.

cs_set_destination_id()

Syntax of Function

```
#include "mdalib.h"
int far cs_set_destination_id (BYTE port_number,
     WORD id,
     BYTE type_flag);
```

Description of Function

This function changes the designated port's destination id. This is the individual or group id of the radio or host designated to receive messages by default when the ***cs_write_message*** commands are used. When the cs_write_addr_msg commands are used, this destination id is overridden.

NOTE: This function must be called before any other ***cs_set*** function, ***cs_init***, or ***cs_open_rf*** can be used.

Description of Parameters

port_number indicates the communications port for the function. Valid port values are COM1 (serial port 0) and COM2 (serial port 1).

The **id** is either a group id (GID) or an individual or host id (LID) used as the destination id for the radio data interface. The type_flag must be IS_GID if id is a group id, or IS_LID if id is an individual or host id. The valid range for group IDs is 0-2047. The valid range for individual IDs is 1-16382. Host IDs must be in the range 1-64, however, no error is detected for a host ID in the range 65-16382. CommServ defaults the id to 0000, two bytes of null.

Return Values

The return value is zero if the operation was successful. If an error occurs during execution of the function, the global variables **cs_rtn_level** and **cs_rtn_code** are set with an error level and code, and the function returns the value of **cs_rtn_level**.

cs_set_open_timeout()

Syntax of Function

```
#include "mdalib.h"
int far cs_set_open_timeout (BYTE port_number,
     WORD open_timeout);
```

Description of Function

This function presets the designated port’s **open_timeout** value for subsequent ***cs_open*** calls. If this function is not called, the default value for open timeout is 182 ticks or 10 seconds.

Description of Parameters

port_number indicates the communications port for the function. Valid port values are COM1 (serial port 0) and COM2 (serial port 1).

The **open_timeout** value specifies the maximum length of time to wait for a command to be completed, and can be from 20 to 65535 ticks. Time is in units of system timer ticks and the default value is 182 ticks (10 seconds). A value of zero disables the timeout function. With a **open_timeout** value of 0, if a ***cs_open*** cannot be completed and does not generate an error, the system may "hang," and have to be restarted.

Return Values

The return value is zero if the operation was successful. If an error occurs during execution of the function, the global variables **cs_rtn_level** and **cs_rtn_code** are set with an error level and code, and the function returns the value of **cs_rtn_level**.

cs_set_rx_buffer_size()

Syntax of Function

```
#include "mdalib.h"
int far cs_set_rx_buffer_size (BYTE port_number,
    WORD new_rx_buffer_size);
```

Description of Function

This function changes the designated port's rx_buffer_size to the desired setting. The port must be closed when this command is issued.

Description of Parameters

port_number indicates the communications port for the function. Valid port values are COM1 (serial port 0) and COM2 (serial port 1).

The **new_rx_buffer_size** is the size for the **rx_buffer** that is allocated during a ***cs_init*** call. CommServ defaults all ports to 6000. The **tx_buffer_size** and the **rx_buffer_size** must be 531 bytes or more and fit into the buffer pool. Buffer pool is set with a command line parameter when CommServ is loaded.

Return Values

The return value is zero if the operation was successful. If an error occurs during execution of the function, the global variables **cs_rtn_level** and **cs_rtn_code** are set with an error level and code, and the function returns the value of **cs_rtn_level**.

cs_set_tx_buffer_size()

Syntax of Function

```
#include "mdalib.h"
int far cs_set_tx_buffer_size (BYTE port_number,
     WORD new_tx_buffer_size);
```

Description of Function

This function changes the designated port’s **tx_buffer_size** to the desired setting. The port must be closed when this command is issued.

Description of Parameters

port_number indicates the communications port for the function. Valid port values are COM1 (serial port 0) and COM2 (serial port 1).

new_tx_buffer_size is the size for the **tx_buffer** that is allocated during a **cs_init** call. CommServ defaults all ports to 3000. The **tx_buffer_size** and the **rx_buffer_size** must be 531 bytes or more and fit into the buffer pool. Buffer pool is set with a command line parameter when CommServ is loaded.

Return Values

The return value is zero if the operation was successful. If an error occurs during execution of the function, the global variables **cs_rtn_level** and **cs_rtn_code** are set with an error level and code, and the function returns the value of **cs_rtn_level**.

cs_set_write_timeout()

Syntax of Function

```
#include "mdalib.h"
int far cs_set_write_timeout (BYTE port_number,
     WORD write_timeout);
```

Description of Function

This function presets the designated port's **write_timeout** value for subsequent ***cs_write*** calls. If this function is not called, the default value of 182 ticks, or 10 seconds, is used.

Description of Parameters

port_number indicates the communications port for the function. Valid port values are COM1 (serial port 0) and COM2 (serial port 1).

The **write_timeout** value represents the maximum length of time to wait for the command to be completed, and can be from 55 to 65535. Time is in units of system timer ticks. The minimum **write_timeout** value depends on the message length for a particular ***cs_write()*** call. A **write_timeout** value of at least 182 ticks (10 seconds) is recommended. This is the default value.

Return Values

The return value is zero if the operation was successful. If an error occurs during execution of the function, the global variables **cs_rtn_level** and **cs_rtn_code** are set with an error level and code, and the function returns the value of **cs_rtn_level**.

cs_start_trace()

Syntax of Function

```
#include "mdalib.h"
int far cs_start_trace (BYTE port_number);
```

Description of Function

This function enables the trace facility. CommServ starts tracing each ***cs_read*** and ***cs_write*** command performed on the specified port. During a ***cs_read*** or ***cs_write*** command, every character that comes in (or goes out) the indicated port is copied to the appropriate trace buffer. The port must be initialized with the ***cs_init()*** call, and the trace function must be initialized with the ***cs_init_trace()*** call, when this command is executed.

Description of Parameters

port_number indicates the communications port for the function. Valid port values are COM1 (serial port 0) and COM2 (serial port 1).

Return Values

The return value is zero if the operation was successful. If an error occurs during execution of the function, the global variables **cs_rtn_level** and **cs_rtn_code** are set with an error level and code, and the function returns the value of **cs_rtn_level**.

cs_stop_trace()

Syntax of Function

```
#include "mdalib.h"
int cs_stop_trace (BYTE port_number);
```

Description of Function

This function disables the trace facility. CommServ stops tracing ***cs_read*** and ***cs_write*** commands on the designated port.

Description of Parameters

port_number indicates the communications port for the function. Valid port values are COM1 (serial port 0) and COM2 (serial port 1).

Return Values

The return value is zero if the operation was successful. If an error occurs during execution of the function, the global variables **cs_rtn_level** and **cs_rtn_code** are set with an error level and code, and the function returns the value of **cs_rtn_level**.

cs_tx_trace_read()

Syntax of Function

```
#include "mdalib.h"
int far cs_tx_trace_read (BYTE port_number,
     void far *buffer,
     WORD buffer_count,
     WORD far *actual_count);
```

Description of Function

Characters are read from the Tx buffer and are copied to a memory location designated by **buffer**. The number of characters transferred is designated by **buffer_count**. Data must be removed from the Tx trace buffer before the buffer overflows or the oldest data is lost.

Description of Parameters

port_number indicates the communications port for the function. Valid port values are COM1 (serial port 0) and COM2 (serial port 1).

buffer is a far pointer to the data area to receive Tx trace data. **buffer_count** is the number of bytes requested from the Tx trace buffer. It must be in the range of 1 to tx_trace_size.

Return Values

The **actual_count** field indicates the actual number of bytes copied from the Tx trace buffer. **actual_count** can be less than **buffer_count** if an error occurs. The function returns a zero if the operation was successful. If an error occurs during execution of the function, the global variables **cs_rtn_level** and **cs_rtn_code** are set with an error level and code, and the function returns the value of **cs_rtn_level**.

cs_write_addr_msg_ignore_ack()

Syntax of Function

```
#include "mdalib.h"
int far cs_write_addr_msg_ignore_ack(
     BYTE port_number,
     BYTE type,
     WORD id,
     void far *buffer,
     WORD buffer_count,
     WORD far *actual_count);
```

Description of Function

A number of characters, **buffer_count**, pointed to by **buffer**, are written to the Tx buffer of the indicated port. The RDI protocol header is added to the data stream, and **type** and **id** are used to specify the message destination. If there is not enough room in the Tx buffer for the message plus overhead, the function waits for the period specified by **write_timeout**. Once the message is in the buffer, the function returns immediately. Any subsequent ACK/NAK response is ignored.

Description of Parameters

port_number indicates the communications port for the function. Valid port values are COM1 (serial port 0) and COM2 (serial port 1).

The **type** parameter indicates whether the destination id is a group id (GID) or an individual or host id (LID). This parameter must have a value of IS_GID or IS_LID.

The **id** parameter is the message destination ID. If **type** = IS_GID, the id must be from 0 to 2047. If **type** = IS_LID, the id must be from 1 to 16382. The **buffer** parameter is a far pointer to the data buffer containing the message to be transmitted.

buffer_count is the number of bytes to be transmitted, and must be from 1 to 2000 bytes.

Return Values

actual_count is the actual number of bytes copied to the Tx buffer, excluding the RDI protocol header. **actual_count** may be less than **buffer_count** if an error occurs. The return value is zero if the operation was successful. If an error occurs during execution of the function, the global variables **cs_rtn_level** and **cs_rtn_code** are set with an error level and code, and the function returns the value of **cs_rtn_level**.

cs_write_addr_msg_no_wait()

Syntax of Function

```
#include "mdalib.h"
int far cs_write_addr_msg_no_wait(
    BYTE port_number,
    BYTE type,
    WORD id,
    void far *buffer,
    WORD buffer_count,
    WORD far *actual_count);
```

Description of Function

A number of characters, **buffer_count**, pointed to by **buffer**, are written to the Tx buffer of the indicated port. The RDI protocol header is added to the data stream, and **type** and **id** are used to specify the message destination. If there is not enough room in the Tx buffer for the message plus overhead, the function waits for the period specified by **write_timeout**. Once the message is in the buffer, the function returns immediately, but CommServ remains in the WAITING_FOR_ACK state. No further write commands are accepted until the ACK/NAK response is received.

The ***cs_write_msg_ack_status*** call must be used next (or repeatedly) to determine when this call completes.
cs_write_msg_no_wait starts the timeout interval that remains active through subsequent ***cs_write_msg_ack_status*** calls. Future calls with ***cs_write_msg_ack_status*** return a busy state until this call is completed successfully, with an error, or the timeout has expired. See the ***cs_set_write_timeout*** function call.

Description of Parameters

port_number indicates the communications port for the function. Valid port values are COM1 (serial port 0) and COM2 (serial port 1).

The **type** parameter indicates whether the destination id is a group id (GID) or an individual or host id (LID). This parameter must have a value of IS_GID or IS_LID.

The **id** parameter is the message destination ID. If **type** = IS_GID, the id must be from 0 to 2047. If **type** = IS_LID, the id must be from 1 to 16382. The **buffer** parameter is a far pointer to the data buffer containing the message to be transmitted.

buffer_count is the number of bytes to be transmitted, and must be from 1 to 2000 bytes.

Return Values

actual_count is the actual number of bytes copied into the Tx buffer excluding the RDI protocol header. **actual_count** could be less than **buffer_count** if an error occurs. The return value is 1, CMD_ACCEPT, if the operation was accepted. If an error occurs during execution of the function, the global variables **cs_rtn_level** and **cs_rtn_code** are set with an error level and code, and the function returns the value of **cs_rtn_level**.

cs_write_addr_msg_wait_ack()

Syntax of Function

```
#include "mdalib.h"
int far cs_write_addr_msg_wait_ack(
     BYTE port_number,
     BYTE type,
     WORD id,
     void far *buffer,
     WORD buffer_count,
     WORD far *actual_count);
```

Description of Function

A number of characters, **buffer_count**, pointed to by **buffer**, are written to the Tx buffer of the indicated port. The RDI protocol header is added to the data stream, and **type** and **id** are used to specify the message destination. If there is not enough room in the Tx buffer for the message plus overhead, the function waits for the period specified by **write_timeout**. Once the message is in the buffer, the function waits for an ACK/NAK response before returning to the function.

Description of Parameters

port_number indicates the communications port for the function. Valid port values are COM1 (serial port 0) and COM2 (serial port 1).

The **type** parameter indicates whether the destination id is a group id (GID) or an individual or host id (LID). This parameter must have a value of IS_GID or IS_LID.

The **id** parameter is the message destination ID. If **type** = IS_GID, the id must be from 0 to 2047. If **type** = IS_LID, the id must be from 1 to 16382. The **buffer** parameter is a far pointer to the data buffer containing the message to be transmitted.

buffer_count is the number of bytes to be transmitted, and must be from 1 to 2000 bytes.

Return Values

actual_count is the actual number of bytes copied to the Tx buffer, excluding the RDI protocol header. **actual_count** may be less than buffer count if an error occurs. The return value is zero if the operation was successful. If an error occurs during execution of the function, the global variables **cs_rtn_level** and **cs_rtn_code** are set with an error level and code, and the function returns the value of **cs_rtn_level**.

cs_write_msg_ack_status()

Syntax of Function

```
#include "mdalib.h"
int far cs_write_msg_ack_status (
     BYTE port_number);
```

Description of Function

This call must be used after the ***cs_write_addr_msg_no_wait*** call to see how it concluded. Only ***cs_get*** and ***cs_read*** calls are allowed until the previous ***no_wait*** command finishes either successfully or in error.

Description of Parameters

port_number indicates the communications port for the function. Valid port values are COM1 (serial port 0) and COM2 (serial port 1).

Return Values

The return value is 2, DEV_BUSY, if the operation is still pending. The return value is 0 if the ACK was received. If an error occurs during execution of the function, the global variables **cs_rtn_level** and **cs_rtn_code** are set with an error level and code, and the function returns the value of **cs_rtn_level**.

cs_write_msg_ignore_ack()

Syntax of Function

```
#include "mdalib.h"
int far cs_write_msg_ignore_ack(
     BYTE port_number,
     void far *buffer,
     WORD buffer_count,
     WORD far *actual_count);
```

Description of Function

A number of characters, **buffer_count**, pointed to by **buffer**, are written to the Tx buffer of the indicated port. The RDI protocol header is added to the data stream, and the default destination ID is used to specify the message destination (see ***cs_set_destination_id***). If there is not enough room in the Tx buffer for the message plus overhead, the function waits for the period specified by **write_timeout**. Once the message is in the buffer, the function returns immediately. Any subsequent ACK/NAK response is ignored.

Description of Parameters

port_number indicates the communications port for the function. Valid port values are COM1 (serial port 0) and COM2 (serial port 1).

buffer is a far pointer to the data buffer to be transmitted. **buffer_count** is the number of bytes to be transmitted. It must be in the range from 1 to 2000.

Return Values

actual_count is the actual number of bytes copied into the Tx buffer excluding the RDI protocol header. **actual_count** could be less than **buffer_count** if an error occurs. The return value is zero if the operation was successful. If an error occurs during execution of the function, the global variables **cs_rtn_level** and **cs_rtn_code** are set with an error level and code, and the function returns the value of **cs_rtn_level**.

cs_write_msg_no_wait()

Syntax of Function

```
#include "mdalib.h"
int far cs_write_msg_no_wait(
     BYTE port_number,
     void far *buffer,
     WORD buffer_count,
     WORD far *actual_count);
```

Description of Function

A number of characters, **buffer_count**, pointed to by **buffer**, are written to the Tx buffer of the indicated port. The RDI protocol header is added to the data stream, and the default destination ID is used to specify the message destination (see ***cs_set_destination_id***). If there is not enough room in the Tx buffer for the message plus overhead, the function waits for the period specified by **write_timeout**. Once the message is in the buffer, the function returns immediately, but CommServ remains in the WAITING_FOR_ACK state. No further write commands are accepted until the ACK/NAK response is received.

The ***cs_write_msg_ack_status*** call must be used next (or repeatedly) to determine when this call completes. ***cs_write_msg_no_wait*** starts the timeout interval that remains active through subsequent ***cs_write_msg_ack_status*** calls. Future calls with ***cs_write_msg_ack_status*** return a busy state until this call is completely successfully, with an error, or the timeout has expired. See the ***cs_set_write_timeout*** function call.

Description of Parameters

port_number indicates the communications port for the function. Valid port values are COM1 (serial port 0) and COM2 (serial port 1).

buffer is a far pointer to the data buffer to be transmitted. **buffer_count** is the number of bytes to be transmitted. It must be in the range from 1 to 2000.

Return Values

actual_count is the actual number of bytes copied into the Tx buffer excluding the RDI protocol header. **actual_count** could be less than **buffer_count** if an error occurs. The return value is 1, CMD_ACCEPT, if the operation was accepted. If an error occurs during execution of the function, the global variables **cs_rtn_level** and **cs_rtn_code** are set with an error level and code, and the function returns the value of **cs_rtn_level**.

cs_write_msg_wait_ack()

Syntax of Function

```
#include "mdalib.h"
int far cs_write_msg_wait_ack (
     BYTE port_number,
     void far *buffer,
     WORD buffer_count,
     WORD far *actual_count);
```

Description of Function

A number of characters, **buffer_count**, pointed to by **buffer**, are written to the Tx buffer of the indicated port. The RDI protocol header is added to the data stream, with the default destination id used to specify the message destination (see ***cs_set_destination_id***). If there is not enough room in the Tx buffer for the message plus overhead, the function waits for the period specified by **write_timeout**. Once the message is in the buffer, the function returns immediately. Any subsequent ACK/NAK response is ignored.

Description of Parameters

port_number indicates the communications port for the function. Valid port values are COM1 (serial port 0) and COM2 (serial port 1).

buffer is a far pointer to the data buffer to be transmitted. The **buffer_count** is the number of bytes to be transmitted. It must be in the range from 1 to 2000.

Return Values

actual_count indicates the actual number of bytes copied to the Tx buffer excluding the RDI protocol header. **actual_count** could be less than **buffer_count** if an error occurs. The return value is zero if the operation was successful. If an error occurs during execution of the function, the global variables **cs_rtn_level** and **cs_rtn_code** are set with an error level and code, and the function returns the value of **cs_rtn_level**.

This page intentionally left blank

Appendix A

Return Levels and Return Codes

A-1 Return Levels

Level	Symbol/Meaning

0 OP_CMPLT

The command or operation completed successfully.

1 CMD_ACCEPT

The command was expected by CommServ but has not been completed. Valid only on a WRITE command with subcommand 03.

2 DEV_BUSY

The command was rejected because the device is busy with a previous operation. This includes waiting for an ACK, or an attempt to re-enter CommServ.

3 INVALID_CALL

The command received by CM from the application has invalid parameters in the Device Control Block. Some causes for this are:

- Bad value in the DCB
- Invalid command
- Null pointer
- unsupported baud rate in init structure
- invalid port

- Improper call sequence
  - Open command issued before an init command
  - Read or Write issued before an Open command

4 SOFT_ERROR

CommServ detected an error that may be corrected with a retry.

5 USER_ACTION

CommServ has reported a condition that probably requires operator action to correct.

6 HARD_ERROR

CommServ has reported a hardware malfunction.

- Communications port adapter not present
- Radio Data Interface not responding to commands

A-2 CommServ Return Codes

The Return Code is the specific exception code number used to detail the Return Level. The application programmer may use these codes to build an extensive error checking and recovery routine using the numbers as an index to, (1) an error message file for screen display of message and instructions, and/or (2) National Language Support error message displays.

A-2.1 Informational and Error Return Codes

All Return Codes presented by CommServ are positive for informational codes and negative for error codes. The following is a list of all codes returned at the completion of a command.

A-2.2 Informational Return Codes

Code	Symbol/Meaning
0	CS_SUCCESS (Return Level = 0) The command completed successfully.
1	RESERVED
2	NO_MESSAGES (Return Level = 4) There are no messages available in the receive buffer.
3	RESERVED
4	UNRECOGNIZED_MESSAGE_DISCARDED (Return Level = 4) Data was found in the receive buffer that contained an undetermined format while operating packet mode transfers. The data did not constitute a valid message packet and was therefore discarded. Message count is affected.

A-2.3 DCB Error Return Codes

(Return Level = 3)

Code	Symbol/Meaning
-1	DCB_INV_PORT The command is directed to an invalid port. The valid range is 0 or 1.
-2	DCB_INV_XFR_TYPE The selected transfer type is invalid for this command. Valid transfer types are MSG_MODE (2) and ADDR_MSG_MODE (10).
-3	DCB_INV_COMMAND The command is invalid. The valid range is 1 to 6.
-4	DCB_INV_SUB_COMMAND The subcommand selected for this command is invalid. The valid range is command dependent.
-5	DCB_INV_SUB_CMD_FOR_XFR_TYPE The subcommand selected is invalid for use with the selected data transfer type.
-6	RESERVED
-7	DCB_INV_ID_TYPE The ID type (byte 23 of DCB) specified in as ADDR_MSG_MODE transfer was invalid. Valid types are IS_GID (1) and IS_LID (2).

-8 DCB_INV_DESTINATION_ID

The destination ID (bytes 24-25 of DCB) specified in as ADDR_MSG_MODE transfer was invalid. Valid range is 0-2047 for GID's and 1-16382 for LID's.

-9 RESERVED

-10 DCB_INV_POINTER

Any NULL pointer detected by CommServ involving the DCB where a valid pointer in required.

-11 DCB_INV_BUFFER_SIZE

CommServ has detected that the buffer size parameter of the DCB,BYTES 18-19, is zero for a WRITE or READ command, or the buffer size is larger than the receive buffer allocated by CommServ at initialize time. e.g., the application allocated a receive buffer of 6000 bytes and the READ command requests 6005 bytes. The 6005 exceeds the size of the allocated receive buffer.

-12 RESERVED

-13 RESERVED

-14 RESERVED

-15 RESERVED

-16 DCB_INV_NEXT_DCB_POINTER

The Chained DCB pointer, BYTES 26-29, designated as pointing to the next DCB in the chain, when chaining is selected, is NULL.

A-2.4 INITIALIZE Structure Error Return Codes

(Return Level=3 unless otherwise noted)

Code	Symbol/Meaning
-20	IS_INV_PORT_ATTRIBUTE The PORT attribute in the INITIALIZE structure is invalid. The only valid value is 4.
-21	IS_INV_PARITY The PARITY in the INITIALIZE structure for this port is invalid. The only valid value is 2.
-22	IS_INV_STOP_BITS The number of STOP BITS in the INITIALIZE structure is invalid. The only valid value is 1.
-23	RESERVED
-24	IS_INV_WORD_LEN The character WORD LENGTH in the INITIALIZE structure is invalid. The only valid value is 3.
-25	IS_INV_PORT_ADDR The port I/O address in the INITIALIZE structure is zero. The default is zero when COM ports are not listed in the BIOS Data area and produces this error if initialized.

-26 IS_INV_INT_NUM

The hardware Interrupt Vector Number used to install the hardware Interrupt Service Routine (ISR) in low memory is out of the range supported by CommServ. The valid range is 10 to 15.

-27 IS_INV_IRQ_NUM

The selected hardware Interrupt Request Number (IRQ) is out of the range supported by CommServ. The valid range is 2 to 7.

-28 IS_INV_DESTINATION_ID

The default destination ID for messages using the MSG_MODE transfer type is invalid. The destination ID is a 3 or 4 digit hex number expressed as ASCII characters. For group ID's (GID's) the ASCII string must be "000" to "7FF" with a null in the fourth character. For individual ID's (LID's) the ASCII string must be "0001" to "3FFF".

-29 IS_Rx_Tx_MODE

The Rx/Tx Mode parameter in the port initialize structure is not set for either receive or transmit.

-30 IS_INV_Rx_BUFFER_SIZE

The selected Rx buffer size for the receive buffer is below the minimum (530) or above the maximum (51200).

-31 IS_INV_Tx_BUFFER_SIZE

The selected Tx buffer size for the transmit buffer is below the minimum (530) or above the maximum (51200).

-32 IS_NO_ROOM_IN_BUFFER_POOL

Not enough room is available in the CommServ buffer pool for the requested Rx or Tx buffer size. The user must adjust the size of the Rx or Tx buffer requested or remove and reload CommServ with a new larger/Knn buffer pool request on the command line.

-33 IS_INV_BAUD_RATE

The selected BAUD rate for this port is not in a valid range for the device in use. The only valid value is 9600.

-34 RESERVED

-35 RESERVED

-36 RESERVED

-37 IS_INV_FLOW_CONTROL

The defined FLOW CONTROL method is invalid. The only valid flow control method is RDI_FLOW_CTRL.

-38 RESERVED

-39 RESERVED

-40 RESERVED

-41 IS_SHARED_PORT_INT_IRQ_MISMATCH

There is a conflict on the port: All ports sharing the same IRQ number must have the same interrupt vector number. This error is produced when 2 or more ports are initialized with the same IRQ value and different interrupt vector numbers, or the same interrupt vector numbers associated with multiple IRQ numbers.

-42 IS_NO_UART_FOUND
(Return Level=6)

CommServ has detected there is no UART (8250 type) for the selected port or I/O address. This is a hardware problem.

-43 IS_INV_RxTx_MODE_FOR_PORT_ATTRB

The port attribute indicates that both Rx and Tx interrupt mode must be enabled for successful operation. Packet data transfer method require both Rx and Tx to be enabled.

-44 IS_INTERRUPT_PENDING
(Return Level=5)

When attempting to initialize the interrupt controller (8259) for the designated port, an active interrupt (a pending interrupt) was found. This could be caused by the asynchronous communication device not being properly closed by other communications programs or by defective hardware. Reboot the system. If this error persists, the hardware is defective or another non-communication device is sharing the same interrupt and causing this error.

A-2.5 Trace Related Error Return Codes
(Return Level=3)

<u>Code</u>	<u>Symbol/Meaning</u>
-60	TR_NO_TRACE_OPTION At least one trace option bit must be specified during initializing of trace.
-61	TR_INV_TRACE_STRUC_PTR Pointer to the user structure is NULL. Refer to the definition for initializing the trace pointers under "2.5.1 Initialize Subcommands".
-62	TR_INV_TRACE_BUFFER_SIZE The trace buffer size contained in the trace structure is less than 2 bytes. The size must be at least two bytes for CommServ to place data into the designated buffer.
-63	TR_INV_TRACE_BUFFER_PTR The pointer to the user trace buffer is NULL. Refer to the definition for initializing the trace pointers under "2.5.1 Initialize Subcommands".
-64	TR_TRACE_NOT_INIT A request to open tracing has been received by CommServ and the trace structures have not been initialized for this port using the INITIALIZE command with the appropriate subcommand.

A-2.6 CommServ Related Error Return Codes

Code Symbol/Meaning

-70 CM_OP_TIMEOUT
(Return Level = 4)

The requested operation has timed out. This error can occur if insufficient time is provided during an OPEN, or WRITE.

-71 CS_INTERNAL_ERROR
(Return Level = 6)

This is a CommServ internal error.

-72 CS_ATTEMPT_TO_REENTER
(Return Level = 2)

The application has attempted to re-enter CommServ from within CommServ. For example, if the application calls CommServ from a timer-tick interrupt in the background and calls CommServ in the foreground, this error occurs. This may also happen when operating with CommServ while using a debugger and exiting the debugger before allowing CommServ to return to the caller.

-73 CS_UNSOLICITED_INTERRUPT
(Return Level = 5)

CommServ received an interrupt for the designated port which was not recognized. The interrupt handler in CommServ can only process 8250,16450, or 16550 type UART interrupts. Therefore, all interrupts on this IRQ are blocked when an unsolicited (unknown) interrupt occurs. This return code could be caused by a defective communication adapter or a non-communication device sharing the same interrupt. Reboot the system. If this error persists, check the communication adapter and other adapters in the system that may have inadvertently been assigned the wrong IRQ number.

A-2.7 CommServ State Error Return Codes

Code	Symbol/Meaning
-80	ST_PORT_NOT_USED (Return Level = 3) The port attribute for the requested port is set to zero indicating the port in not in use. This can occur when trying to initialize a port or initialize trace when the port attribute indicates that the port is not used.
-81	ST_PORT_NOT_CLOSED (Return Level = 3) The port must be in STATE = CLOSED before issuing the command that produced this error.
-82	ST_PORT_NOT_INIT (Return Level = 3) The port must be in STATE = INITIALIZED before issuing the command that produced this error. **Note:** If the port is in the OPEN state, the Port must be CLOSED and then INITIALIZED before issuing the command that produced this error.
-83	ST_PORT_NOT_OPEN (Return Level = 3) A READ or WRITE command was issued and the port was not opened with the OPEN command.

-84 ST_PORT_BUSY
(Return Level = 2)

The selected port is in use. This is the normal response code when waiting for an ACK, or trying to re-enter.

-85 ST_PORT_NOT_WAITING_FOR_ACK
(Return Level = 3)

The command to "inquire if an ACK has been received" (write subcommand 04) was issued before the command to "transmit without waiting for the ACK" (write subcommand 03) to be executed.

-86 RESERVED

A-2.8 Radio Data Interface Error Return Codes

Code Symbol/Meaning

-120 RF_NO_MODEM_RESPONSE
(Return Level = 6)

The Radio Data Interface (RDI) does not respond to any commands. This is an indication that the power may not be ON for the RDI, the RDI is not attached, or the RDI itself is defective.

-121 RF_OUT_OF_RANGE
(Return Level = 5)

A message was transmitted by the RDI but not acknowledged by the intended receiver. There are a number of conditions that may cause this response: Physically out of range (in a tunnel, etc), defective Tx and/or Rx antennae on the mobile radio or the BASE STATION antennae, or any other circumstances that cause the transmitting mobile radio to not receive from the BASE STATION for any reason.

-122 RF_NAKed_MESSAGE
(Return Level = 4)

The message sent by the RDI was aborted by the EDACS system, and a NAK response was sent to the RDI. The message is discarded by the RDI.

-123 RESERVED

-124 RESERVED

-125 RESERVED

-126 RF_PACKET_ERROR
(Return Level = 4)

The RDI detected an error in the transmit data packet presented to it. This is an RDI generated error. The user should retry the operation.

-127 RESERVED

-128 RF_MSG_TOO_LONG
(Return Level = 3)

The maximum RDI write message size was exceeded. The current packet size limit for the RDI is 2000 bytes, or the size of the Tx buffer minus space for RDI protocol overhead, whichever is less.

-129 RESERVED

-130 RESERVED

-131 RESERVED

-132 RESERVED

-133 RESERVED

-134 RF_TIMEOUT_VALUE_TOO_SMALL
(Return Level = 3)

The timeout value contained in the DCB is less than the minimum value required to transmit the message or not zero (No timeout). The minimum timeout depends upon packet length. For a 2000 byte packet the minimum timeout is 180 timer ticks.

The recommended timeout value is 550 timer ticks or approximately 30 seconds for network operations.

A-2.9 Abort Error Return Codes

Code	Symbol/Meaning

-150 AB_ABORT_SEQ_ZERO_LEN
(Return Level = 3)

The keystroke byte count contains a zero (0) when the OPEN command subcommand 07 was issued.

-151 AB_ABORT_SEQ_TOO_LONG
(Return Level = 3)

The keystroke sequence byte count exceeds the maximum of four bytes.

-152 AB_USER_ABORT
(Return Level = 5)

This return code indicates that the current CommServ command was aborted by the user via a keyboard key sequence.

A-2.10 READ Error Return Code

Code Symbol/Meaning

-160 Rx_BUFFER_EMPTY
(Return Level = 4)

The serial receive buffer is empty and the requested byte count has not been exhausted when using packet mode transfer. The number of bytes available in the receive buffer has reached zero before the requested byte count value reaches zero. One cause of data overruns is excessive time spent in servicing higher priority interrupts than the communications port.

-161 Rx_USER_BUFFER_TOO_SMALL
(Return Level = 4)

The message in the Rx buffer is larger than then buffer supplied to receive it. This is determined by checking the available message with the buffer size in the READ command DCB.

-162 Rx_BAD_LEN_CHECK
(Return Level = 4)

This is an internal error and the application should retry the operation.

-163 Rx_OVERRUN_ERROR
(Return Level = 4)

The "RECEIVE DATA" overrun error has occurred. The data message(s) in CM receive buffer has missing character(s). It may result in buffer empty or packet errors.

Data overrun conditions can also remove a character from the UART before the next character is received. Generally, this is caused by interrupts being disabled by other interrupt service routines for too long a period of time during receipt of a message. The application may be disabling interrupts for too long a time period. **NOTE**: The cs_reset_status ( ) function call will clear this return code.

-164 Rx_PARITY_ERROR
(Return Level = 4)

There was a PARITY error detected on received character. The data in the receive buffer may have invalid characters. This is generally caused by data corruption on the circuit. The remote transmitter can send characters to the receiver with the wrong parity. **NOTE**: The ***cs_reset_status ( )*** function call will clear this return code.

-165 Rx_FRAMING_ERROR
(Return Level = 4)

A framing error occurred during receiving of data. The data in the receive buffer may be invalid. This can be caused by data degeneration due to poor data transmission quality or different Baud rates between sender and receiver. **NOTE**: The ***cs_reset_status ( )*** function call will clear this return code.

-166 RESERVED

-167 Rx_BUFFER_OVERFLOW
(Return Level = 4)

More characters were received than could fit in the receive buffer. The extra characters were lost.

A-2.11 Library Error Return Codes
(Return Level = 3)

Code Symbol/Meaning

-500 LB_NO_CS_PRESENT

CommServ could not be found in memory.

-501 RESERVED

-502 LB_INV_POINTER

The DCB pointer is NULL.

-503 LB_NO_MEMORY

There was insufficient memory available to complete this request.

-504 RESERVED

-505 RESERVED

-506 LB_TRACE_ALREADY_ALLOCATED

The current trace buffers must be freed before new buffers can be allocated.

-507 LB_TRACE_ALREADY_FREED

The trace buffers must be allocated before they can be freed.

-508 LB_TIMEOUT_VALUE_TOO_SMALL

The timeout value specified is too small. Set the timeout equal or greater than the minimum, or set the timeout to 0 = forever.

-509 LB_INV_POINTER_ACTUAL_COUNT

The pointer is NULL to the **actual_count** parameter of a ***cs_read, cs_write, cs_rx_trace_read*** or ***cs_tx_trace_read*** call.

Appendix B

MDALib to CommServ Relationships

MDALib Function Call	CommServ Call
cs_alloc_trace_buffers()	
cs_cancel_write_msg_no_wait()	Write subcommand 5 Transfer_Type = Message
cs_close()	Close subcommand 1
cs_disable_user_abort()	Close subcommand 4
cs_enable_user_abort()	Open subcommand 7
cs_free_trace_buffers()	
cs_get_api_version_date()	
cs_get_api_version_number()	
cs_get_cs_version_date()	
cs_get_cs_version_number()	
cs_get_line_status()	Status subcommand 1
cs_get_modem_id()	Status subcommand 1
cs_get_modem_status()	Status subcommand 1
cs_get_options()	Status subcommand 1
cs_get_rx_buffer_count()	Status subcommand 1
cs_get_rx_count()	Status subcommand 11
cs_get_rx_errors()	Status subcommand 11
cs_get_rx_msg_count()	Status subcommand 1
cs_get_state()	Status subcommand 1
cs_get_tx_buffer_count()	Status subcommand 1

Figure B-1. MDALib to CommServ Call Relationships (Part 1 of 3)

MDALib Function Call	**CommServ Call**
cs_get_tx_count()	Status subcommand 11
cs_get_tx_errors()	Status subcommand 11
cs_get_tx_retries()	Status subcommand 11
cs_get_vector_number()	
cs_interrupt()	
cs_init()	Init subcommand 5
cs_init_trace()	Init subcommand 9
cs_is_installed()	
cs_open_rf()	Open subcommand 1
cs_read_msg()	Read subcommand 0 Transfer_Type = Message
cs_remove()	Close subcommand 3
cs_reset_msg_stats()	Status subcommand 12
cs_reset_status()	Status subcommand 2
cs_rx_trace_read()	
cs_set_defaults()	Init subcommand 1
cs_set_destination_id()	Init subcommand 4 then 2
cs_set_open_timeout()	
cs_set_rx_buffer_size()	Init subcommand 4 then 2
cs_set_tx_buffer_size()	Init subcommand 4 then 2
cs_set_write_timeout()	
cs_start_trace()	Open subcommand 6
cs_stop_trace()	Open subcommand 2

Figure B-2. MDALib to CommServ Call Relationships (Part 2 of 3)

MDALib Function Call	CommServ Call
cs_tx_trace_read()	
cs_write_addr_msg_ignore_ack()	Write subcommand 1 Transfer_Type = Addressed Message
cs_write_msg_ignore_ack()	Write subcommand 1 Transfer_Type = Message
cs_write_addr_msg_no_wait()	Write subcommand 3 Transfer_Type = Addressed Message
cs_write_msg_no_wait()	Write subcommand 3 Transfer_Type = Message
cs_write_addr_msg_wait_ack()	Write subcommand 2 Transfer_Type = Addressed Message
cs_write_msg_wait_ack()	Write subcommand 2 Transfer_Type = Message
cs_write_msg_ack_status()	Write subcommand 3 Transfer_Type = Message

Figure B-3. MDALib to CommServ Call Relationships (Part 3 of 3)

This page intentionally left blank.

Appendix C

Coding Examples

C-1. Introduction

The following information gives example coding, that can be used. The sequence indicated in the examples is recommended.

C-2. MDALib Calling Order

The table below shows the minimum number of calls required to use a port (the read and write calls may be in any order with each other, and both read and write need not be used. All function return codes should be checked.

COM1

cs_set_destination_id()
cs_init()
cs_open_rf()
cs_write_msg_wait_ack()
cs_read_msg()
cs_close()

C-3. Reading the Rx Buffer

The following cs_get calls check the buffer contents. The ***cs_get_rx_buffer_count*** can be used with the message transfer type, but protocol bytes are reported. That can be misleading if not taken into account.

COM1

cs_get_rx_msg_count()
cs_read_msg()

C-4. Using Trace and User_Abort Calls

The sample shown is used with a Radio Packet modem on port COM1, but is applicable to other ports. Trace and Abort can be used on individual calls, and are independent of the port Open and Close calls. Trace can be started anytime, and User_Abort can be enabled anytime.

Trace Call	User_Abort Call
cs_set_destination_id()	cs_set_destination_id()
cs_set_write_timeout()	cs_set_write_timeout()
cs_alloc_trace_buffers()	
cs_init_trace()	
cs_start_trace()	cs_enable_user_abort()
cs_open_rf()	cs_open_rf()
cs_write_msg_wait_ack()	cs_write_msg_wait_ack()
cs_get_rx_msg_count()	cs_get_rx_msg_count()
cs_read_msg()	cs_read_msg()
cs_rx_trace_read()	
cs_tx_trace_read()	
cs_stop_trace()	cs_disable_user_abort()
cs_free_trace_buffers()	
cs_close()	cs_close()

Appendix D

Return Codes and API Calls

D-1. Introduction

CommServ generates most of the Return Codes; the MDAPI passes them to the application. The following tables associate each return code to the MDAPI function call that could return it.

	Return Code	Function Call
2	NO_MESSAGES	cs_read_msg()
4	UNRECOGNIZED_MESSAGE_DISCARDED	cs_read_msg()
-1	DCB_INV_PORT	Any call with Port as a parameter.
-2	DCB_INV_XFR_TYPE	(1)
-3	DCB_INV_COMMAND	(1)
-4	DCB_INV_SUB_COMMAND	(1)
-5	DCB_INV_SUB_CMD_FOR_XFR_TYPE	cs_write and cs_open calls
-7	DCB_INV_ID_TYPE	cs_write calls
-8	DCB_INV_DESTINATION_ID	cs_write and cs_set_destination_id
-9	Reserved	
-10	DCB_INV_POINTER	cs_write calls and cs_read_msg() with a NULL Buffer
-11	DCB_INV_BUFFER_SIZE	cs_read_msg()
-12	Reserved	
-13	Reserved	
-14	Reserved	
-15	Reserved	
-16	DCB_INV_NEXT_DCB_POINTER	cs_interrupt()
-20	IS_INV_PORT_ATTRIBUTE	(1)
-21	IS_INV_PARITY	(1)
-22	IS_INV_STOP_BITS	(1)
-23	Reserved	
-24	IS_INV_WORD_LEN	(1)
-25	IS_INV_PORT_ADDR	cs_init()
-26	IS_INV_INT_NUM	(1)
-27	IS_INV_IRQ_NUM	(1)

Figure D-1. Return Codes and MDALib Calls (Part 1 of 3)

NOTE: 1. CommServ should not return this error if MDALib is being used. If this error appears there is a bug in the MDAPI library or the environment in which it is running.

	Return Code	Function Call
-28	IS_INV_DESTINATION_ID	cs_init() See cs_set_destination_id()
-29	IS_INV_Rx_Tx_MODE	(1)
-30	IS_INV_Rx_BUFFER_SIZE	cs_init() See cs_set_rx_buffer_size()
-31	IS_INV_Tx_BUFFER_SIZE	cs_init() See cs_set_tx_buffer_size()
-32	IS_NO_ROOM_IN_BUFFER_POOL	cs_init() See cs_set_rx_buffer_size() See cs_set_tx_buffer_size()
-33	IS_INV_BAUD_RATE	(1)
-37	IS_INV_FLOW_CONTROL	(1)
-41	IS_SHARED_PORT_INT_IRQ_ MISMATCH	cs_init()
-42	IS_NO_UART_FOUND	cs_init()
-43	IS_INV_RxTx_MODE_FOR_ PORT_ATTRB	(1)
-44	IS_INTERRUPT_PENDING	cs_init()
-60	TR_NO_TRACE_OPTION	(1)
-61	TR_INV_TRACE_STRUC_PTR	(1)
-62	TR_INV_TRACE_BUFFER_SIZE	(1)
-63	TR_INV_TRACE_BUFFER_PTR	(1)
-64	TR_TRACE_NOT_INIT	cs_start_trace()
-70	CS_OP_TIMEOUT	cs_open and cs_write and calls that support timeouts.
-71	CS_INTERNAL_ERROR	
-72	CS_ATTEMPT_TO_REENTER	
-73	CS_UNSOLICITED_INTERRUPT	
-80	ST_PORT_NOT_USED	cs_init()
-81	ST_PORT_NOT_CLOSED	cs_init()
-82	ST_PORT_NOT_INIT	cs_open Calls
-83	ST_PORT_NOT_OPEN	cs_write() and cs_read_msg() calls
-84	ST_PORT_BUSY	(2)
-85	ST_PORT_NOT_WAITING_FOR_ACK	cs_write_msg_ack_status()

Figure D-2. Return Codes and MDALib Calls (Part 2 of 3)

NOTE:

1. CommServ should not return this error if MDALib is being used. If this error appears there is a bug in the MDAPI library or the environment in which it is running.

NOTE:

2. When CommServ is in the ***cs_waiting_for_ack*** state, many calls, not ***cs_gets***, return this.

	Return Code	Function Call
-120	RF_NO_MODEM_RESPONSE	cs_write calls.
-121	RF_OUT_OF_RANGE	cs_write calls.
-122	RF_NAKed_MESSAGE	cs_write calls.
-126	RF_PACKET_ERROR	cs_write_msg calls.
-128	RF_MSG_TOO_LONG	cs_write_msg calls.
-134	RF_TIMEOUT_VALUE_TOO_SMALL	cs_open_rf()
-150	AB_ABORT_SEQ_ZERO_LEN	cs_enable_user_abort()
-151	AB_ABORT_SEQ_TOO_LONG	cs_enable_user_abort()
-152	AB_USER_ABORT	cs_open, cs_write, calls that support user abort.
-160	Rx_BUFFER_EMPTY	cs_read_msg()
-161	Rx_USER_BUFFER_TOO_SMALL	cs_read_msg()
-162	Rx_BAD_LEN_CHECK	cs_read_msg()
-163	Rx_OVERRUN_ERROR	cs_read_msg() and cs_write() calls
-164	Rx_PARITY_ERROR	cs_read_msg() and cs_write() calls
-165	Rx_FRAMING_ERROR	cs_read_msg() and cs_write() calls
-167	Rx_BUFFER_OVERFLOW	cs_read_msg() and cs_write() calls
-500	LB_NO_CS_PRESENT	All calls that call CommServ.
-502	LB_INV_POINTER	cs_interrupt()
-503	LB_NO_MEMORY	cs_alloc_trace_buffers()
-506	LB_TRACE_ALREADY_ALLOCATED	cs_alloc_trace_buffers()
-507	LB_TRACE_ALREADY_FREED	cs_free_trace_buffers()
-508	LB_TIMEOUT_VALUE_TOO_SMALL	cs_set_open_timeout()
-509	LB_INV_POINTER_ACTUAL_COUNT	cs_write, cs_read_msg, cs_rx_trace_read and cs_tx_trace_read calls

Figure D-4. Return Codes and MDALib Calls (Part 3 of 3)

NOTE: 1. CommServ should not return this error if MDALib is being used. If this error appears there is a bug in the MDAPI library or the environment in which it is running.

ERICSSON

Ericsson GE Mobile Communications Inc.
Mountain View Road • Lynchburg, Virginia 24502

Printed in U.S.A.