

MC2002 Hand-Held Smart Card Read/Write Device (RWD) Optional Device Library Manual: SDLC Accessing Library

Version 1.0

Sep 2002

REVISION HISTORY

Version Number	Date	Description	Author
1.0	16 Sep, 2002	Initial release	Hwang Toung

Contents

CHAPTER 1	5
INTRODUCTION	5
1.1 POS	6
1.2 SDLC	6
1.3 SYSTEM REQUIREMENT	7
1.4 SCOPE OF THIS MANUAL	7
CHAPTER 2	8
OVERVIEW	8
2.1 TRANSACTION FLOW	9
2.2 PROGRAM FLOW	10
2.3 INITIALIZATION	11
2.4 FINITE STATE MACHINE ENTRY	11
2.5 SEND AN INFORMATION PACKET	12
2.6 DISCONNECTING	12
2.7 CALLBACKS	12
2.7.1 <i>Idle dealer</i>	12
2.7.2 <i>Link establishment dealer</i>	12
2.7.3 <i>Information packet dealer</i>	13
2.7.4 <i>Hint message dealer</i>	13
2.7.4.1 dialing	13
2.7.4.2 link layer receiving	13
2.7.4.3 information packet sending	13
2.7.4.4 link layer maintenance	14
2.8 ERROR STRING	14
CHAPTER 3	15
SUBROUTINES AND CALLBACKS	15
3.1 INITIALIZATION	16
3.2 FINITE STATE MACHINE ENTRY	17
3.3 SEND AN INFORMATION PACKET	18
3.4 DISCONNECTING	19
3.5 CALLBACKS	20
3.5.1 <i>Idle dealer</i>	20
3.5.2 <i>Link establishment dealer</i>	20
3.5.3 <i>Information packet dealer</i>	21
3.5.4 <i>Hint message dealer</i>	21
CHAPTER 4	22

SAMPLES	22
4.1 ABOUT IT.....	23
4.2 MAIN().....	23
4.3 CALLBACKS.....	23
4.3.1 <i>Idle dealer</i>	23
4.3.2 <i>Link establishment dealer</i>	24
4.3.3 <i>Information packet dealer</i>	24
4.3.4 <i>Hint dealer</i>	24

Chapter 1

Introduction

Introduction

1

1.1 POS

POS (Point Of Sale), literally means the physical location of a transaction, but usually refers to any device or system that is used to record the transaction for the retailer. POS should be able to deal with bank cards, in which case the POS should exchange the transaction information with the host computer of the bank/card center. POS transactions typically consist of a request message from the terminal to the host and a response message from the host to the terminal. Traditional POS terminals use an internal modem and a telephone line to contact a modem that answers the call. The request message is then transferred from the terminal to the network using some type of protocol. The response returns to the terminal using the protocol and the modems. Most hosts use a networking product as a front end so that the host is not responsible for driving the protocol for every terminal it supports. The networking product interfaces with the host system using X.25, SNA, Token Ring, Ethernet, or even an Asynchronous interface for installations where performance requirements are not particularly high.

Here are the components in a typical dial network for POS:



PSTN means Public Switched Telephone Network, and NAC is the networking product referred to above, which means Network Access Controller, the communications device that performs protocol translation, routing, and network concentration.

1.2 SDLC

Synchronous Data Link Control (SDLC) was developed by IBM in the mid-1970s for use in the Systems Network Architecture (SNA) environments. It was the first link layer protocol based on synchronous bit-oriented operation. It has been adopted by various standard committees, and several standards have been created based on SDLC. But SDLC remains the primary SNA link layer protocol for WAN links.

All NACs support SDLC.

For more information about SDLC, try searching 'SDLC' in Internet. Here's something to get started.

http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/sdlcetc.htm

1.3 System requirement

MC2002 could be used as a POS terminal provided some optional communication peripherals are installed.

An optional synchronous modem must be connected to communicate to NAC via PSTN. The modem extension board for MC2002 is based on SiLab's SI2403/14/33/56 series chips, which is capable of both synchronous (HDLC) and asynchronous serial communications.

An additional software object module named **sdlc_fsm.obj** should be added to the project, and a header file **sdlc.h** should also be included in the source code.

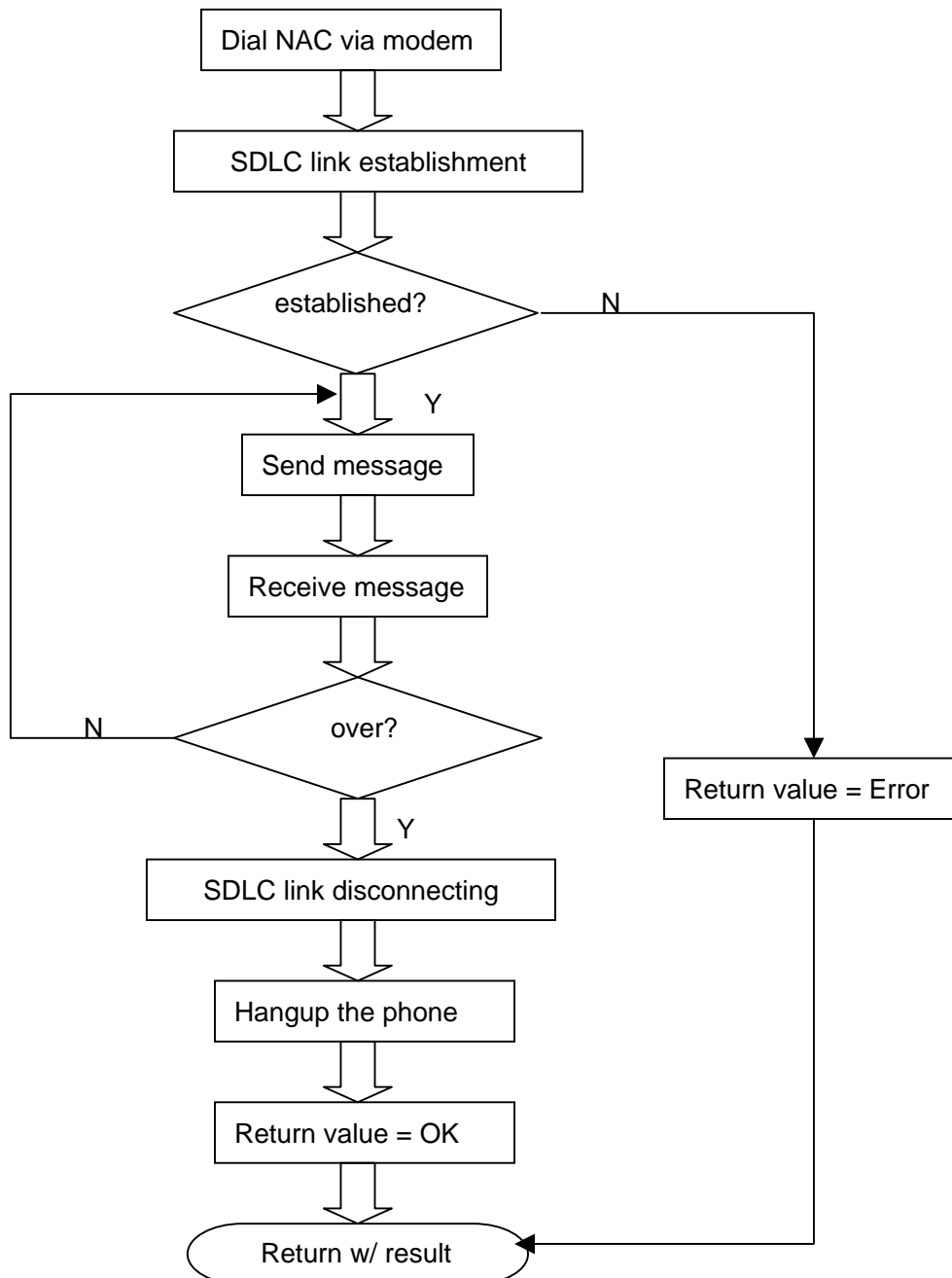
1.4 Scope of this manual

This manual only deals with the SDLC communication part of the transaction, i.e. the link layer. The messages needed for a complete transaction are based on ISO8583 series protocols, which are not included in this manual.

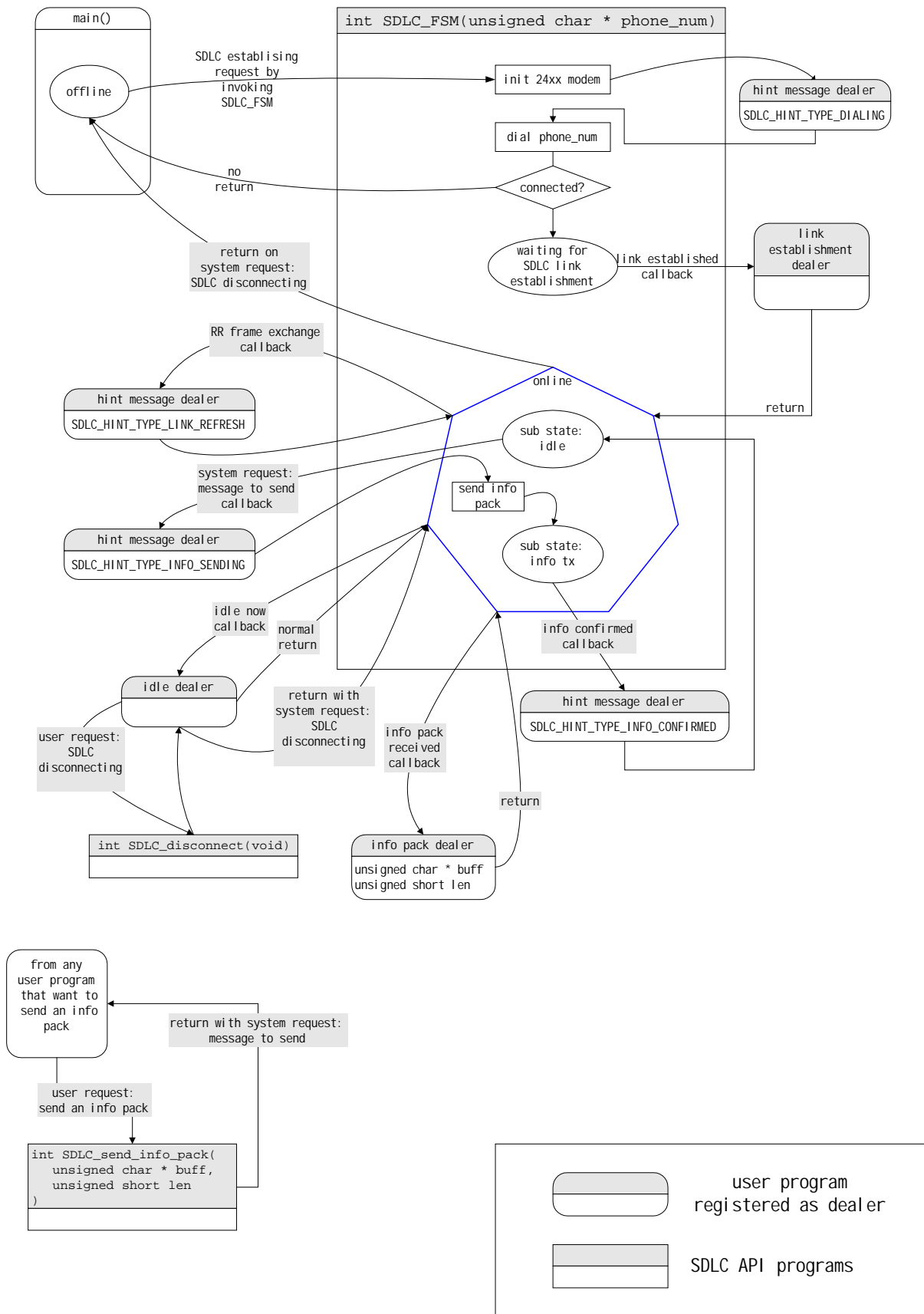
Chapter 2

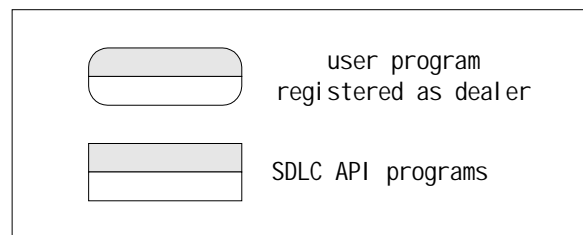
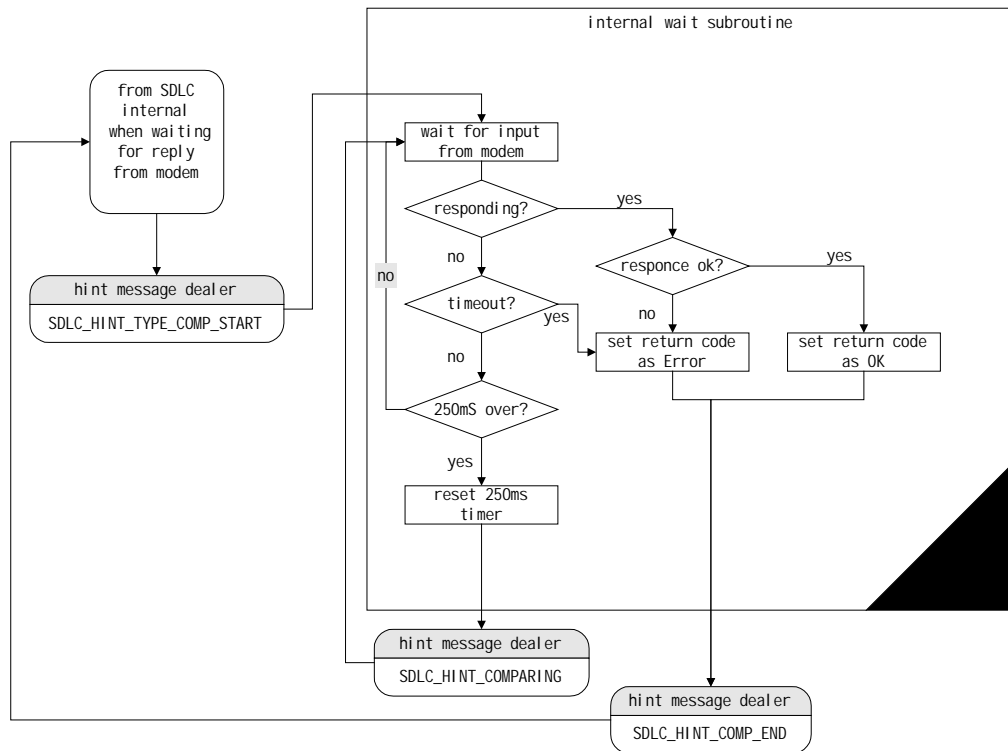
Overview

2.1 Transaction Flow



2.2 Program Flow





2.3 Initialization

Before entering the SDLC finite state machine, all the callbacks must be registered, otherwise there will be no callback available for user program.

Subroutine used:

[SDLC_register_dealers\(\)](#);

2.4 Finite State Machine entry

After the user gets into the finite state machine, the program flow will not return until the state machine has been terminated. The state machine handles a lot of things: dialing NAC, link establishing, link maintaining, information package routing. However, user could still intercept the program flow via callbacks. The only recommended method for returning from the state machine is to tell it from the idle callback. C.f. [callbacks](#).

The modem must be turned on as the current UART device before entering this FSM.

Subroutine used:

[SDLC_FSM\(\)](#);

2.5 Send an information packet

User could send an information packet while the link is available.

Since this subroutine will not work until the link has been established, it only makes sense to invoke it inside a callback.

The packet is NOT sent on the return of this subroutine, it's just put in the queue to be sent via I-frame. C.f. [information packet sending](#) section of [Hint message dealer](#)

Subroutine used:

[SDLC_send_info_pack\(\)](#);

2.6 Disconnecting

User could terminate an existing SDLC link by disconnecting it.

Since this subroutine will not work until the link has been established, it only makes sense to invoke it inside a callback.

Disconnecting a link does not make the state machine to return. But because the SDLC link would be lost after that, it's recommended that the state machine be terminated.

Subroutine used:

[SDLC_disconnect\(\)](#);

2.7 Callbacks

Callbacks are the only ways to keep the system under your control and be advised when something happens.

There are actually 4 types of callbacks here in the SDLC component:

The idle dealer, the link establishment dealer, information packet dealer, and the hint message dealer.

2.7.1 Idle dealer

This callback is called whenever the state machine is idle. Since it's invoked so often, the program flow should not choke here, or the entire state machine would collapse.

Almost everything needed to control the communication should be done here: sending package, disconnecting, check for user input, etc.

However, it's still possible to enter power saving mode here, since we're working as the secondary station, it is natural for the system to go to sleep when it's idle, as long as it could be waked up by any input, including the serial port (SDLC link data) and the keypad.

The return value for this dealer determines the termination of the entire state machine.

Please c.f. [sample](#) section for more.

2.7.2 Link establishment dealer

This callback is called once the SDLC link has been established. It's invoked only once per

session.

Nothing really has to be done for this dealer, but many could be. Such as starting your own state machine for high level protocol, exchanging message right now, or much simpler, as in the sample, just drawing something to show that we are online.

2.7.3 Information packet dealer

This callback is called if a valid information packet (I frame) has been received. You could decide what to do with it here. But whatever it is, return as fast as you can, because the primary site is still waiting for the response.

2.7.4 Hint message dealer

Several types of hint messages are sent by invoking this dealer with various parameters.

2.7.4.1 dialing

This message is sent once the dialing has begun. It might take some time to connect to NAC, but it shouldn't be too long, less than 8 seconds most of the time.

The dialing may be successful, while the link establishment dealer is invoked; or unsuccessful, while the state machine would just return with the error code.

Parameter passed in:

SDLC_HINT_TYPE_DIALING

2.7.4.2 link layer receiving

There're 3 messages in this message group. They are sent while the state machine is waiting for a certain reply from the modem. This could occur anytime while initializing, dialing, or exchanging data.

The 3 messages stand for: *waiting started*, *waiting*, *waiting ended*. *Waiting* message is sent 250 milliseconds after the *waiting started* message and every 250 milliseconds after that till it's over. So for those waiting periods shorted than 250 ms, there would be only 2 messages sent.

Parameter passed in:

SDLC_HINT_TYPE_COMP_START

SDLC_HINT_TYPE_COMPARING

SDLC_HINT_TYPE_COMP_END

2.7.4.3 information packet sending

There're 2 messages in this message group. They are sent while the state machine is sending an information packet (I frame).

The 2 messages are: *sending*, *confirmed*. Message *sending* is sent right before the sending of I frame. And Message *confirmed* is sent while the primary station has acknowledged the frame via the link layer.

The packet would not be sent immediately after the information packet sending subroutine is invoked. So the sending begins with message *sending* , and only after Message *confirmed* could

the packet be considered as delivered.

Parameter passed in:

SDLC_HINT_TYPE_INFO_SENDING

SDLC_HINT_TYPE_INFO_CONFIRMED

2.7.4.4 link layer maintenance

2 messages are sent for link layer maintenance: *link refresh*, *pack sending error*.

Message *link refresh* is sent each time an RR frame exchange occurs. The exchange is originated by the primary station, and for most of the time, it's just checking if the secondary station is still online.

Message *pack sending error* is sent whenever a packet could not be sent to the modem. In this case, there must be something wrong with the interface.

Parameter passed in:

SDLC_HINT_TYPE_LINK_REFRESH

SDLC_HINT_TYPE_PACK_SEND_ERROR

2.8 Error string

After the state machine returns, the error string hold the reason for its returning.

extern unsigned char SDLC_err_string[];

Chapter 3

Subroutines and Callbacks

3.1 Initialization

`SDLC_register_dealers()` is the state machine initialization subroutine.

```
void SDLC_register_dealers(  
    int (* SDLC_dealer_idle)(void),  
    int (* SDLC_dealer_link_establishment)(void),  
    int (* SDLC_dealer_info_pack)(unsigned char *, unsigned short),  
    int (* SDLC_dealer_hint)(unsigned short)  
);
```

Parameters:

SDLC_dealer_idle
Address of the idle dealer.

SDLC_dealer_link_establishment
Address of the link establishment dealer.

SDLC_dealer_info_pack
Address of the information packet dealer.

SDLC_dealer_hint
Address of the link hint message dealer.

Notes:

Each of the parameters could be NULL, which indicates that there is no customised dealer for that type. A default dealer will be used instead, which is just a return.

3.2 Finite state machine entry

`SDLC_FSM()` is the entry for the SDLC state machine.

```
int SDLC_FSM(unsigned char * phone_num)
```

Parameters:

Phone_num

NULL terminated string storing the phone number of the NAC to be dialled to.

Return Values

0: succeeded

Other values: state machine terminated with reason indicated in `SDLC_err_string`.

Notes:

- 1) The phone number must be a number that is accessible from current phone line attached to the handheld. The number is dialled "as is", no prefix or pause is added. So, if you want to write a program that could handle different dialup network situations, you got to do it yourself, with some configuration UI/API.
- 2) Return value 0 is not applicable in current version.

3.3 Send an information packet

`SDLC_send_info_pack ()` puts an information packet into I-frame sending queue.

```
Void SDLC_send_info_pack(  
    unsigned char * buff,  
    unsigned short len  
)
```

Parameters:

buff

Pointer to a data buffer that contains data to send.

len

length of the data to send.

Note:

The data would be packed in an SDLC I-frame.

c.f. [send a information packet](#) section of [overview](#)

3.4 Disconnecting

`SDLC_disconnect()` is used to disconnect the SDLC link and hang up the phone.

```
int SDLC_disconnect(void)
```

Return Values:

0: succeeded. The state machine has been disconnected.
Other values: failed. There must have been something wrong with the modem

Notes:

The SDLC link would be lost after this subroutine. C.f. [disconnecting](#) section of [overview](#)

3.5 Callbacks

3.5.1 Idle dealer

The state machine invokes the idle dealer whenever it's idle.

```
int my_idle_dealer(void)
```

Return Values:

SDLC_IDLE_DEALER_NORMAL:

Stay in the state machine.

SDLC_IDLE_DEALER_ABORT:

Terminate the state machine with SDLC_err_string set to "user abort".

SDLC_IDLE_DEALER_HANG_UP:

Terminate the state machine.

Notes:

C.f. [idle_dealer](#) section of [overview](#)

3.5.2 Link establishment dealer

The state machine invokes the link establishment dealer when the link is established.

```
int my_link_establishment_dealer(void)
```

Return Values:

N/A

Notes:

C.f. [link_establishment_dealer](#) section of [overview](#)

3.5.3 Information packet dealer

The state machine invokes the information packet dealer when an valid I-frame has been received.

```
int my_info_pack_dealer(  
    unsigned char * buff,  
    unsigned short len  
)
```

Parameters:

buff

Pointer to a data buffer that contains data received.

len

length of the data received.

Notes:

C.f. [information packet dealer](#) section of [overview](#)

3.5.4 Hint message dealer

The hint message dealer is called in many occasions. This dealer is a like a message center, anytime when the state machine has something to say, it calls this dealer. C.f. [hint message dealer](#) section of [overview](#).

```
int my_hint_dealer( unsigned short htype)
```

Parameters:

Htype:

The message sent by the state machine. Could be one of the following in this version:

SDLC_HINT_TYPE_DIALING: dial started.

SDLC_HINT_TYPE_COMP_START: modem receiving started.

SDLC_HINT_TYPE_COMPARING: modem receiving processing.

SDLC_HINT_TYPE_COMP_END: modem receiving ended.

SDLC_HINT_TYPE_INFO_SENDING: information packet sending started.

SDLC_HINT_TYPE_INFO_CONFIRMED: information packet sent confirmed.

SDLC_HINT_TYPE_PACK_SEND_ERROR: error sending any packet

SDLC_HINT_TYPE_LINK_REFRESH: link refreshment.

Chapter 4

Samples

Here's an example on how to use the SDLC component.

4.1 About it

This is a project to demonstrate the usage of the SDLC APIs. Here's the file list of the source code package:

File name	Size(bytes)	date	time	description
M2A-F-2M.LD	1,934	08-17-02	11:43	Ld file for the project
SDLC2414.PRJ	1,509	08-17-02	11:44	Project file
CRT0.C	421	04-25-01	15:39	Start up file
C_TIMER.H	297	08-17-02	9:48	Header for timer function
TIMER.C	1,537	08-17-02	9:48	Source for timer function
SDLC.H	890	08-31-02	10:27	Header for the SDLC component
SDLC_FSM.OBJ	4,373	08-17-02	9:53	Precompiled object for the SDLC component
BMP_DATA.H	596	08-16-02	16:22	Header for bitmap data
BMP_DATA.C	7,245	08-16-02	16:38	Bitmap data in C source code as constant
TEST.C	12,444	08-19-02	10:38	The main program of this sample

4.2 main()

After the initialization, the program enters the main loop for offline mode. It's nothing more than an old fashioned key dealer: key 0~9 for phone number entering, CLR for backspace, F1 to toggle modem on/off, and F2 to enter SDLC state machine.

Since SDLC_FSM() does not return until the SDLC link broken(either not established successfully or user abortion), the program flow will not come back to the main loop until then.

4.3 Callbacks

4.3.1 Idle dealer

my_idle_dealer().

The system enters power saving mode here by invoking sys_msg() in the idle dealer. It won't leave that mode until a key has been pressed, or a character has been received from the UART or SPT time out. Please note that the parameter for sys_msg() should be SM_STAY_AWAKE to keep the UART alive (C.f. MC2002 API manual).

Actually it's another key dealer: key F1 to hang up the phone and quit the state machine, key

F2 to send a sample packet (ISO8583 format).

It also deals with the high-level timer time out. A timer is initiated while sending the information packet, and it's triggered again when the pack is being sent actually, when the packet is confirmed by the link layer of primary station, and when the primary station replies this packet with another information packet (high level message exchange). So, if any of above fails, time out will occur. We just print a line saying timeout in this sample.

4.3.2 Link establishment dealer

`my_link_establishing_stub()`

Nothing happens here except for some screen repainting: redraw the hint for key F1 & F2, paint the background of the animation for link refresh, and the status line on the top of the LCD.

4.3.3 Information packet dealer

`my_info_pack_dealer()`

Just print out the length of the data received. We do not have much room on the LCD to display the whole data packet.

The high level timer is discarded here, because we do not need it anymore.

4.3.4 Hint dealer

`my_hint_dealer()`

This is the most sophisticated one in the sample:

For `SDLC_HINT_TYPE_DIALING`:

Print "dialing" on the status line at the top of the LCD. Enable hourglass animation for modem communication.

For `SDLC_HINT_TYPE_COMP_START`:

Initialize the hourglass animation if enabled: allocate the animation timer, and put 1st frame.

For `SDLC_HINT_TYPE_COMPARING`:

Put next frame of the hourglass animation if enabled.

For `SDLC_HINT_TYPE_COMP_END`:

Ends the hourglass animation if it's enabled: restore screen, discard the animation timer.

For `SDLC_HINT_TYPE_INFO_SENDING`:

Reload the high level time out timer, prompt "sending info", and enable the hourglass animation.

For `SDLC_HINT_TYPE_INFO_CONFIRMED`:

Reload the high level time out timer, prompt "info confirmed info".

For `SDLC_HINT_TYPE_LINK_REFRESH`:

Draw a moving dot on the background to show that the link is on. The ugly calculation of `cntr_cn_block` is for moving the dot back and forth.