System Function

Timer and Delay functions are used to design an application program.

SB_GetTick

Function	Returns the time passed in milliseconds after Eddy booted.
Format	Unsigned long SB_GetTick (Void);
Parameter	None
Returns	0 ~ 4,294,967,295
Notice	Returned value is the system tick counter in milliseconds. Unsigned Long type with maximum value of 0xffffffff. When it reaches the maximum value, it will start again from 0. (About every 49.7 days, the value will reach the maximum value and start from 0.)

SB_msleep

Function	Delay for specified milliseconds.	
Format	void SB_msleep (int msec);	
Parameter	msec	Delay for this amount of time.
Returns	none	
Notice	Delay for specified milliseconds.	

SB_AliveTime

Function	Returns uptime in days, hours, minutes, and seconds since Eddy booted.	
Format	void SB_AliveTime (int *day, int *hour, int *min, int *sec);	
Parameter	*day	operated day (0 ~ large number)
	*hour	hours (0 ~ 23)

	*min	minutes (0 ~ 59)
	*sec	seconds (0 ~ 59)
Returns	None	
Notice		

Eddy Environment Function

SB_ReadConfig

Function	Retrieve configuration from a file.	
Format	void SB_ReadConfig (char *FileName, char *Dest, int Size);	
Parameter	FileName	Path of the file including the folder location.
	*Dest	Point of the buffer to read the configuration.
	Size	Size of the file
Returns	Error Code	1 when successfully read, -1 when fails
Notice	Configuration file is saved in /etc and /flash folders.	
	When the users modify the settings from the web manager or telnet, they will be saved here which will be referenced by all Eddy applications.	

SB_WriteConfig

Function	Save configuration settings to a file.	
Format	void SB_WriteConfig (char *FileName, char *Source, int Size);	
Parameter	FileName	Path of the file including the folder location.
	Source	structure buffer point where the settings are saved
	Size	Size of the structure
Returns	Error Code	1 when successfully read, -1 when fails
Notice		

SB_GetSharedMemory

Function	Read from the pointer from the shared memory	
Format	void *SB_GetSharedMemory (int Key_ID, int Buffer_Size);	
Parameter	Key_ID	ID of the shared menory
	Buffer_Size	Size of shared memory in use
Returns	*buffer_address	Address of the shared memory
		-1 when fails
Notice	This function is a shared memory used by PortView and SNMP V1/2/3 to monitor and control the operation status of applications in Eddy. Each applications will update their status periodically and provide the information to PortView server and SNMP server in the network. To use this, PortView and SNMP Agent need to be set in the configuration.	

SB_SetSharedMemory

Function	Request the shared memory to be used and read the memory pointer.	
Format	void *SB_SetSharedMemory (int Key_ID, int Buffer_Size);	
Parameter	Key_ID	ID of shared menory to be registered.
	Buffer_Size	Size of the shared memory
Returns	*buffer_address	Address location of the shared memory
		Returns -1 when fails.
Notice	This function is used for PortView and SNMP agent. The users can use this to share the memory with other application.	

Serial Function

Functions to control internal serial port (UART)

SB_OpenSerial

Function	Open serial port	
Format	int SB_OpenSerial (int Port_No, int modem, inte interface);	
Parameter	Port_No	serial port number
		0 : first serial port
		15 : second serial port
	Model	Corresponding model number (Number of ports in SerialGate)
	Interface	Select from 1, 2, 4, 8, or 16 port model
		Select serial interface type
		RS232 0
		RS422_PTOP 1
		RS422_MULTIDROP 2
		RS485_NONE_ECHO 3
		RS485_ECHO 5
Returns	-1 ~ N	Handle of opened serial port
		-1 : open error
		N : handle of opened serial port
Notice	From 1 to 16 serial ports are supported depending on the models.	

SB_InitSerial

Function	Initialize the data communication format in serial port.	
Format	Void SB_InitSerial (int Handle, char Speed, char LCR, char Flow);	
Parameter	Handle	Handle of serial port from OpenSerial
	Speed	Communication speed

0 :	150 BPS,	1 :	300 BPS
2 :	600 BPS	3 :	1200 BPS:
4 :	2400 BPS	5 :	4800 BPS
6 :	9600 BPS	7 :	19200 BPS
8 :	38400 BPS	9 :	57600 BPS
10 :	115200 BPS	11 :	230400 BPS
12 :	460800 BPS	13 :	921600 BPS

LCR X X P P S D D (8 bits binary)

P P : Parity Bits

0 0 : None, 0 1 : Odd, 1 0, 1 1: Even

S : Stop Bits

0 : 1 bits, 1 : 2 bits

D D : Data Bits

0 0 : 5 bits, 0 1 : 6 bits

1 0 : 7 bits, 1 1 : 8 bits

FlowControl Select flow control from below.

0: no flow control

1: RTS/CTS flow control

2: Xon/Xoff flow control

Returns None

Notice

SB_SendSerial

Function Output data by serial port.

Format Void SB_SendSerial (int handle, char *data, int length);

Parameter handle handle that opens serial port or socket

data Pointer of the data

length Length of the data

Returns None

Notice When the transmission buffer is full, it will retry 10 times with

20msec interval, then returns after the output is complete.

SB_ReadSerial

Function	Read data from the serial port.	
Format	int SB_ReadSerial (int handle, char *data, int length, int wait_msec);	
Parameter	handle	Handle which opened the serial port
	data	buffer pointer of the data
	length	length/size of the buffer
	wait_msec	Stanby time between each data
Returns	0 ~ n	Size of the data
Notice	When wait_msec is set to 0, read the data from serial receiver buffer. When it is set to a value larger than 0, it will wait for specified amount of time then read the data in one packet. The maximum size it can read is set in “length”. The wait_msec can be used from the value by using SB_GetDelaySerial or entered with manually computer value.	

SB_GetMsr

Function	Read the MSR register value from the serial port.	
Format	Char SB_GetMsr (int handle);	
Parameter	handle	Handle which opened the serial port.
Returns	Value	MSR Register Value Bit 7 6 5 4 3 2 1 0 Bit0: CTS change Bit1: DSR change Bit2: RI change Bit3: DCD change Bit4: CTS (0:Low, 1:High) Bit5: DSR (0:Low, 1:High)

Bit6: RI (0:Low, 1:High)

Bit7: DCD (0:Low, 1:High)

Notice

SB_SetRts

Function	Control the RTS signal in serial port.	
Format	Void SB_SetRts (int handle, int value);	
	handle	Handle which opened the serial port.
Parameter	Value	0: off Set RTS signal as low.
		1: on Set RTS signal as high.
Returns	None	
Notice		

SB_SetDtr

Function	Control DTR signal in serial port.	
Format	Void SB_SetDtr (int handle, int value);	
	handle	Handle which opened the serial port
Parameter	Value	0: off Set DTR signal as low.
		1: on Set DTR signal as high.
Returns	None	
Notice		

Ethernet Function

In/Output the information of the current network and related field. This function is optimized for Eddy API. The users can use POSIX compatible standard socket API to develop if they wish to.

SB_GetIp

Function	Read the IP address assigned to the Eddy	
Format	Unsigned int SB_GetIp (char *interface);	
Parameter	Interface	Network Interface Name Set WAN port to “eth0”, and LAN port to “eth1”.
Returns	Unsigned int	Return the IP address as unsigned int type.

Notice	Operating IP address is read instead of set IP address. If the IP address is assigned from the DHCP server, it will get that address.
	To convert the IP address to a string type, use the following example.
	struct in_addr addr;
	addr.s_addr = SB_GetIp ();
	printf ("IP Address : %s ", inet_ntoa(addr));	

SB_GetMask

Function	Read the assigned subnet mask address from the Eddy.	
Format	Unsigned int SB_GetMack (char *interface);	
Parameter	Interface	Name of the interface to be read Set WAN port to “eth0”, and LAN port to “eth1”.
Returns	Unsigned int	Return the Mask address as unsigned int type.
Notice	Refer to SB_GetIp.	

SB_GetGateway

Function	Read the assigned Gateway address from the Eddy.	
Format	Unsigned int SB_SetGeteway(void);	
Parameter	None	
Returns	Unsinged int	Return the Geteway address as unsigned int type.
Notice	Refer to SB_GetIp.	

SB_ConnectTcp

Function	Connect to specified server using the TCP socket.	
Format	Int SB_ConnectTcp (char *IP_Address, int Source_Socket_No, int Dest_Socket_No, int Wait_Sec, Int Tx_Size, int Rx_Size);	
Parameter	IP_Address	The server IP address in string type
	Source Socket_No	Socket number of current device
	Dest Socket_No	Socket number of the server
	Wait_Sec	Standby time (in seconds)
	Tx_Size	Tx buffer size of the socket (in kilo bytes; KB)
	Rx_Size	Rx buffer size of the socket (in kilo bytes)
Returns	-1 ~ N	Handle number of connected socket -1 : Connection Failed N : Handle number of connected socket
Notice	When the connection is not established right away, specifited time in wait_sec will be used then return. Socket buffer size of Tx,Rx_Size can be used from 1 to 64. When the value is smaller than 1, by default, 4 kbytes is used. When the value is larger than 64, 64 kbytes will be used.	

SB_ListenTcp

Function	Standby for connection to TCP socket.	
Format	Int SB_ListenTcp (int Socket_No, Int Tx_Size, int Rx_Size);	
Parameter	Socket_No	TCP socket No. that will standby for connection.

	Tx_Bytes	Tx buffer size of the socket (in K bytes)
	Rx_Bytes	Rx buffer size of the socket (in K bytes)
Returns	-1 ~ N	Handle number of the socket to connect to TCP -1 : Fail to standby for connection N : TCP socket handle number waiting to connect.
Notice	This function will not use Non Blocking function to request connection and wait but will return immediately. Connection waiting will be processed from SB_AcceptTcp. Socket buffer size of Tx,Rx_Size can be used from 1 to 64. When the value is smaller than 1, by default, 4 kbytes is used. When the value is larger than 64, 64 kbytes will be used.	

SB_AcceptTcp

Function	TCP socket handle will wait for network connection.	
Format	Int SB_AcceptTcp (int Socket_No, int wait_msec);	
Parameter	Socket_No	Handle number of TCP socket waiting to connect. (Return value of SB_ListenTcp)
	wait_msec	Standby time (in msec)
Returns	-1 ~ N	The new handle number of the socket to connect to TCP. -1 : socket error 0 : Standby for connect N : New handle number for connected TCP socket
Notice	When the connection is established, new handle number will be assigned. Handles before this function will be closed in this function.	

SB_AcceptTcpMulti

Function	Allow multiple network connection to TCP socket handles waiting to be connected.	
Format	Int SB_AcceptTcpMulti (int Socket_No, int wait_msec);	
Parameter	Socket_No	Handle number of TCP socket waiting to connect. (Return value of SB_ListenTcp)
	wait_msec	Standby time (in msec)
Returns	-1 ~ N	The new handle number of the socket to connect to TCP. -1 : socket error 0 : waiting to connect N : New handle number for connected TCP socket
Notice	When the connection is established, new handle number will be assigned. However, previous handles will not be closed and maximum 1024 socket connections are allowed.	

SB_ReadTcp

Function	Read data from connected TCP socket.	
Format	Int SB_ReadTcp (int Handle, char *Buffer, int Buffer_Size);	
Parameter	Handle	Handle number of connected TCP socket
	Buffer	Buffer point to read and store the packet data
	Buffer_Size	Buffer size
Returns	-1 ~ N	Size of the data -1 : socket error 0 : No data to be read. N : Length of the data
Notice	If the return code is -1, the connection is lost. In this case, TCP socket handle should be closed.	

SB_CloseTcp

Function	Close the TCP socket handle.	
Format	Int SB_CloseTcp (int Handle);	
Parameter	Handle	TCP socket handle to be closed
Returns	None	
Notice	Shutdown socket handle to terminate the communication and close	

SB_BindUdp

Function	Bind UDP socket	
Format	Int SB_BindUdp (int Socket_No);	
Parameter	Socket_No	UDP socket number to be bound
Returns	Handle	Bound UDP socket handle number
		-1: Binding Failed
		N : Bound UDP socket handle number
Notice		

SB_ReadUdp

Function	Read the data to be sent to the bound UDP socket.	
Format	Int SB_ReadUdp (int Handle, char *Buffer, int Buffer_Size);	
Parameter	Handle	Bound UDP socket handle number
	Buffer	Buffer point to save the packet data.
	Buffer_Size	Buffer size
Returns	-1 ~ N	Size of the data.
		-1 : socket error
		0 : No data was read
		N : Length/Size of the data

Notice	When data is sent through UDP socket, this function will remember the IP address and the socket number internally so that they can be used in SB_SendUdpServer.

SB_SendUdpServer

Function	Send data to UDP socket. (Server mode)	
Format	Int SB_SendUdpServer (int Handle, char *Buffer, int Data_Size);	
Parameter	Handle	Handle number bind to UDP socket
	Buffer	Buffer point of the data to be sent
	Data_Size	Size of the data
Returns	None	
Notice	This function sends the data to bound UDP socket to check the target network information in order to be used. In other word, SB_ReadUdp needs to be used first. To send the data first, SB_SendUdpClient should be used.	

SB_SendUdpClient

Function	Send data to UDP socket. (Client mode)	
Format	Int SB_SendUdpClient (int Handle, char *Buffer, int Data_Size, Char *IP_Address, int Socket_No);	
Parameter	Handle	Handle number bind to UDP socket
	Buffer	Buffer point of the data to be transmitted.
	Data_Size	Size of the data
	IP_Address	Target IP address where the data will be sent
	Socket_No	Target socket number
Returns	None	
Notice	This function can be used only when the target IP address is known using UDP socket. To send the data, SB_SendUdpClient function must be used.	

Debugging Function

In Eddy, each applications can be debugged in real-time using telnet. When SB_DEBUG is ON for each applications, debug log messages will be displayed in telnet screen.

SB_LogDataPrint

Function	Print data by each bytes in hex or ASCII code.	
Format	void SB_LogDataPrint (char *RTx, char *buff, int data_len);	
Parameter	*RTx	Description of the data
	*Buff	Buffer address where data to be output is stored
	Data_len	Length of the Data
Returns	None	
Notice	Display the message to the first logged in telnet sh.	
	When output is displayed, the system tick counter, increment by 1 msec in Eddy, will be displayed by following format.	
	SB_LogDataPrint (“Send”, “\t12345\n”, 8);	
	[191020202] Send 8 = 08,1,2,3,4,5,0d,0a	
	--- --- --- ---	
	Tick Counter RTx data_Len buff	
	Debugging for each application in Eddy can be set with using “def” command. (Refer to def.c)	
	# def po <1/2/all> debug <on/off>	

SB_LogMsgPrint

Function	Output can be displayed as same as Printf format.	
Format	void SB_LogMsgPrint (const char *Format, . . .);	
Parameter	*Format	Printf type format
Returns	None	
Notice	Display the message to the first logged in telnet sh.	
	When output is displayed, the system tick counter, increment by 1 msec	

in Eddy, will be displayed by following format.

SB_LogMsgPrint (“%s means Real-Time\n”, “Eddy”);

[191020202] Eddy means Real-Tile

Debugging for each application in Eddy can be set with using “def” command. (Refer to def.c)

Modifying HTML and CGI

This chapter explains CGI modules called by HTML code regarding device configuration.

1.1 WEB Configuration

All source code related to Eddy is located in /src/Eddy_APPs/web/htdocs/.
CGI source code related to HTML are in /src/Eddy_APPs/web/cgi/.

getagent.c

Read the configuration file from /etc folder to show the values in HTML page which will be shown in the web browser.

setagent.c

It will save the configuration to /etc folder when the users make any change from HTML pages.
Finally, by clicking Save & Reboot the configuration file will be saved to /flash folder.

1.2 Sample HTML Code

Example below is a part from the main.html. In HTML, variables as in C language cannot be used to code, therefore, symbols related to CGI are used. The symbolic links are marked with red font color which will pass the value from getagent.c.

(network.html summary)

```
<tr bgcolor="#FFFFFF">
<td class="content">IP Address</td>
<td class="content"><input type="text" size="16" maxlength="16" name="N_IP" value="[v,n_ip]" >

<tr bgcolor="#FFFFFF">
<td class="content">Subnet Mask</td>
```

```
<td class="content"><input type="text" size="16" maxlength="16" name="N_MASK" value="[v,n_mask]" >

<tr bgcolor="#FFFFFF">
<td class="content">Gateway</td>
<td class="content"><input type="text" size="16" maxlength="16" name="N_GW" value="[v,n_gw]" >

<tr bgcolor="#FFFFFF">
<td class="content">DNS</td>
<td class="content"><input type="text" size="16" maxlength="16" name="N_DNS" value="[v,n_dns]" >

<tr bgcolor="#FFFFFF">
<td class="content">Telnet Service</td>
<td class="content"><select name="N_TELNET">
<option [v, n_telnet_di] value="0">Disable</option>
<option [v, n_telnet_en] value="1">Enable</option>
</select>

<tr bgcolor="#FFFFFF">
<td class="content">Telnet Service</td>
<td class="content"><select name="N_WEB">
<option [v, n_web_di] value="0">Disable</option>
<option [v, n_web_en] value="1">Enable</option>
</select>
```

There are **name** and **value** by each record used from above to link with CGI. The name saves the modified information by the user in HTML. When submit button is clicked, information modified by setagent.c is saved to it. The value will get the value from getagent.c and display in HTML so that the users can change the value.

1.3 Sample CGI Codes

The CGI files in Eddy DK consist of getagent.cgi and setagent.cgi. The getagent.c reads configuration file from HTML document in /dtc/ folder to display the environmental variable information. The setagent.c lets modified configuration settings to be saved in /etc/ folder. Eventually, settings are saved in /flash/ so that when Eddy reboots, they are not lost.

The following is the part of the process in getagent.c to show the configuration value in HTML page.

[Summary of the source code]

```
 if (cgiFormStringNoNewlines("N_IP", buff, 16) == cgiFormNotFound)          {
            sprintf(buff, "%d.%d.%d.%d",cfg.system.ip[0], cfg.system.ip[1],cfg.system.ip[2],cfg.system.ip[3]);
listPutf(list, "n_ip", buff);
 }
 else       listPutf(list, "n_ip", buff);

 if (cgiFormStringNoNewlines("N_MASK", buff, 16) == cgiFormNotFound)      {
sprintf(buff, "%d.%d.%d.%d",cfg.system.mask[0], cfg.system.mask[1],
cfg.system.mask[2],cfg.system.mask[3]);
            listPutf(list, "n_mask", buff);
 }
 Else       listPutf(list, "n_mask", buff);

 if (cgiFormStringNoNewlines("N_GW", buff, 16) == cgiFormNotFound)         {
            sprintf(buff, "%d.%d.%d.%d", cfg.system.gateway[0], cfg.system.gateway[1],
cfg.system.gateway[2],cfg.system.gateway[3]);
            listPutf(list, "n_gw", buff);
 }
 Else       listPutf(list, "n_gw", buff);

if (cgiFormStringNoNewlines("N_DNS", buff, 16) == cgiFormNotFound)         {
sprintf(buff, "%d.%d.%d.%d",cfg.system.dns[0], cfg.system.dns[1],
cfg.system.dns[2],cfg.system.dns[3]);
listPutf(list, "n_dns", buff);
}
else        listPutf(list, "n_dns", buff);
```

```
cgiFormInteger("N_TELNET", &value, cfg.system.telnet_server);
if (value == 1)         {
listPutf(list, "n_telnet_di", "");
                listPutf(list, "n_telnet_en", "selected");
}
else        {
listPutf(list, "n_telnet_di", "selected");
listPutf(list, "n_telnet_en", "");
}
cgiFormInteger("N_WEB", &value, cfg.system.web_server);
if (value == 1)         {
listPutf(list, "n_web_di", "");
listPutf(list, "n_web_en", "selected");
  } else      {
listPutf(list, "n_web_di", "selected");
listPutf(list, "n_web_en", "");
 }
```

Below is the part of the process from setagent.c when **submit** button is clicked from HTML page allowing modified settings to be saved.

[Summary of the source code]

```
value2 = cgiFormStringNoNewlines("N_IP", buff, 16);
if (value2 != cgiFormEmpty)   convert_address (buff, cfg.system.ip);

value2 = cgiFormStringNoNewlines("N_MASK", buff, 16);
if (value2 != cgiFormEmpty)   convert_address (buff, cfg.system.mask);

value2 = cgiFormStringNoNewlines("N_GW", buff, 16);
if (value2 != cgiFormEmpty)   convert_address (buff, cfg.system.gateway);

value2 = cgiFormStringNoNewlines("N_DNS", buff, 16);
if (value2 != cgiFormEmpty)   convert_address (buff, cfg.system.dns);
```

```
cgiFormInteger("N_TELNET", &value, cfg.system.telnet_server);
cfg.system.telnet_server = value;

cgiFormInteger("N_WEB", &value, cfg.system.web_server);
cfg.system.web_server = value;
```