OMESH®
NETWORKS

OMESH Networks

OPM15 Simplified Application Programming Interface (S-API)

Programmer's Guide

Version: 0.9.2

Date: Jan. 24, 2012

Email: info@omeshnet.com
Web: http://www.omeshnet.com/omesh/

Contents

1.0 Introduction

OMESH Networks provides a Simplified Application Programming Interface (S-API) written in C language for interfacing various microcontroller platforms to the OPM15 radio module. To date the S-API supports the **Rapid-Mesh OPM15 Development Board**, the **OPM15 Carrier Board**, and the **Arduino** platform. This document serves as a programmer's guide to using the various S-API functions. By using the S-API you can add wireless mesh networking to your project in minutes!

1.1 Design Environment and Microcontroller Resources

The recommended design environment and C compilers supported by the S-API are shown in **Table 1.1**. All of the listed development tools are license-free and are available as free downloads.

Hardware Platform	Chipset	Design Environment	Compiler
OPM15 Carrier Board	Microchip dsPIC30F3013	Microchip MPLAB IDE[1]	MPLAB C30 C compiler for dsPIC DSCs[1].
Rapid-Mesh OPM15 Development Board	Microchip PIC18F25K22 or Arduino	Microchip MPLAB IDE, Arduino IDE[2]	MPLAB C18 C Compiler for PIC18[1], Hi-Tech C for PIC18[1]
Rapid-Mesh OPM15 Shield	Arduino	Arduino IDE[2]	

Table 1.1: Recommended design environments and free compilers for using the S-API.

The microcontroller resources used by the S-API for the different chipsets are shown in **Table 1.2.** The amount of program memory required is dependent on the compiler, so the values provided in the table should be used as an approximation.

Chipset	Source Files	Program Memory	Peripherals
dsPIC30F3013	`opm15.h, opm15.c`	1.72 kBytes	TIMER1, UART2
PIC18F25K22	`opm15.h, opm15.c`	4 kBytes	TIMER0, UART2
Arduino	`opm15.h, opm15.cpp`	4 kBytes	Serial1

Table 1.2: Microcontroller resources used by the S-API.

[1] Latest version available at **http://www.microchip.com**.

[2] Available with integrated compiler at **http://www.arduino.cc**.

1.2 Including the OPM15 Simplified API in your Workspace

Integrating the OPM15 Simplified API in your project is easy, it simply involves adding the correct header and source files to your design workspace.

Microchip PIC18F:

Include the header file *opm15.h* and the source file *opm15.c* from the source folder */OPM15 API/source/pic18f/* in your MPLAB project workspace.

Microchip dsPIC30F:

Include the header file *opm15.h* and the source file *opm15.c* from the source folder */OPM15 API/source/dspic30f/* in your MPLAB project workspace.

Arduino:

Copy the folder */OPM15 API/source/Arduino/OPM15* to your Arduino */libraries/* folder. You can then access the S-API by importing the **OPM15** library into your Arduino sketch by going to **Sketch -> Import Library -> OPM15**. The **OPM15** example projects for Arduino can be accessed by going to **File -> Examples -> OPM15**.

2.0 S-API Data Structures

The following is a definition of data types that the programmer should familiarize themself with to properly use the OPM15 Simplified API. The types are defined in the header file *opm15.h*.

2.1 `Node_Configuration` Type

Summary

OPM15 radio configuration structure.

Definition

```
Node_Configuration varName;
```

Members

`uint8 Baud_rate`	Baud rate setting, range 0x00-0x07.
`uint8 CT_RT_enable`	CT/RT of serial port enable (0x01) or disable (0x00).
`uint8 State_line`	Carrier board LED state line configuration, range 0x00-0x01.
`uint8 Backoff_enable`	Back-off window enable (0x01) or disable (0x00).
`uint8 Previous_address`	Backup previous address enable (0x00) or disable (0x01).
`uint8 Channel_config`	Channel select, range 0x00-0x03.
`uint8 Net_address[3]`	Network address of the radio, three bytes.
`uint8 Radio_power`	Radio power setting, range 0x00-0x08.
`uint8 Pass_code[16]`	Passcode of the radio, sixteen bytes.
`uint8 Retry_limit`	Number of retries limit setting, range 0x00-0xFF.
`uint8 Mesh_address[3]`	Mesh address of the radio, three bytes.
`uint8 Mac_address[6]`	MAC address of the radio, six bytes.
`uint8 Interference_level`	Interference level setting, range 0x00-0x0F.
`uint8 Power_save_enable`	Power save enable (0x01) or disable (0x00).
`uint8 Route_limit[4]`	Route limit parameters, four bytes
`Query_Configuration Qconf`	Query configuration.

Notes

The default values of the members should be set manually in code. Refer to *OPM15 Software API Guide* [1] for details on configuration features of the OPM15 radio module.

Example

```
#include “opm15.h”


int main(void)
{
      Node_Configuration my_node;

      my_node.Baud_rate = 0x05;           //select 230400 baud
      my_node.CT_RT_enable = 0x00;        //disable CT/RT line
      my_node.Backoff_enable = 0x00;      //disable back-off
      my_node.Previous_address = 0x00;    //don’t keep previous address
      my_node.Channel_config = 0x00;      //use all 3 channels
      my_node.Net_address[0] = 0x30;      //network address is 0x323130
      my_node.Net_address[1] = 0x31;
      my_node.Net_address[2] = 0x32;

      my_node.Qconf.Query_rsp_enable = 0x00;    //disable query records

      ...
      return 1;
}
```

2.2 `Read_Only_Data` Type

Summary

OPM15 Radio read only data structure.

Definition

```
Read_Only_Data varName;
```

Members

`uint8 Radio_state`	Current state of the radio.
`uint16 Statistics_data[6]`	Word-sized statistics data of the radio, 6 words.
`uint8 Version_id[3]`	Version ID of radio, 3 bytes.
`uint8 Device_id[6]`	Device ID of radio, 3 bytes.
`uint8 Calibration_data[17]`	Calibration data read, 17 bytes.

Notes

This type stores read-only data from the OPM15 radio.

Example

```
#include “opm15.h”

int main(void)
{
      Read_Only_Data rData;

      Node_read_static_data(&rData);      //read read-only data from radio

      ...
      return 1;
}
```

2.3 `Query_Configuration` Type

Summary

OPM15 radio query configuration type.

Definition

```
Query_Configuration varName;
```

Members

`uint8 Query_rsp_enable`	Query enable (0x01) or disable (0x01).
`uint8 Rssi_record_num`	Number of RSSI records to collect, range 0x01-0x10.

Notes

This member is typically used only within the Node_Configuration type.

2.4 `RSSI_Record` Type

Summary

Structure containing RSSI record information for queries.

Definition

```
RSSI_Record varName;
```

Members

`uint8 rssi`	RSSI value.
`uint16 seq_id`	Sequence ID of RSSI record.

Notes

Used primarily with Query_Record type to store RSSI data, see **2.5 Query_Record Type**.

2.5 `Query_Record` Type

Summary

Structure containing Query records received from OPM15 radio.

Definition

```
Query_Record varName;
```

Members

`uint8 Record_num`	Number of RSSI records retrieved.
`uint8 Source_addr[3]`	Network address of source node.
`RSSI_Record Rssi_rec[16]`	RSSI record data read, up to 16 records total.

Notes

Refer to *OPM15 Software API Guide* [1] for complete details on query record data.

3.0 S-API Functions

The following is a complete list of the OPM15 Simplified API functions. All of the functions are microcontroller independent except for the initialization function *OPM15_mcu_init*, which is used for initializing the microcontroller hardware peripherals used by the S-API.

3.1 `OPM15_mcu_init`

Summary

Initializes microcontroller peripherals including I/O ports, timers, and UARTs. The source implementation is microcontroller dependent.

Definition

`uint16 OPM15_mcu_init();`

Arduino: `uint16 OPM15_mcu_init(uint32 baud_rate);`

Parameters

baud_rate	Serial port baud rate (for Arduino use only).

Return Value

Always returns nonzero value.

Notes

This function must be called if S-API is to be used, and must be called prior to any other S-API function. For Arduino platforms, this function must be called in the setup() function of the sketch.

Example 1 – Microchip PIC

```
#include "opm15.h"

int main(void)
{
      OPM15_mcu_init();       //initialize mcu peripherals

      while(1)
      {
            ...
      }
      return 1;
}
```

Example 2 – Arduino

```
#include "opm15.h"

void setup()
{
      OPM15_mcu_init(9600);   //initialize S-API and set Serial1 to 9600 baud
}

void loop()
{
      //main loop code here
}
```

3.2 OPM15_node_set_configuration

Summary

Write configuration data to OPM15 radio.

Definition

```
uint16 OPM15_node_set_configuration(Node_Configuration *nconf);
```

Parameters

nconf	Pointer to *Node_Configuration* type.

Return Value

If the configuration was successful, returns nonzero.
If any of the configuration commands failed, returns zero.

Notes

The default values of the *Node_Configuration* type should be set in code before being written to the radio. To simplify this process, you can call the **OPM15_node_read_configuration** function to read the current configuration prior to customizing the parameters.

Example 1 – Microchip PIC

```
#include "opm15.h"

int main(void)
{
      OPM15_mcu_init();                    //initialize mcu peripherals
      Node_Configuration my_node;

      my_node.Baud_rate = 0x05;            //select 230400 baud
      my_node.CT_RT_enable = 0x00;         //disable CT/RT line
      my_node.Backoff_enable = 0x00;       //disable back-off
      my_node.Previous_address = 0x00;     //don't keep previous address
      my_node.Channel_config = 0x00;       //use all 3 channels
      my_node.Net_address[0] = 0x30;       //network address is 0x323130
      my_node.Net_address[1] = 0x31;
      my_node.Net_address[2] = 0x32;
      my_node.Qconf.Query_rsp_enable = 0x00;    //disable query records

      OPM15_node_set_configuration(&my_node);   //write to radio
                                                //remember to check errors!
      while(1)
      {
            ...
      }
      return 1;
}
```

Example 2 – Arduino

```
#include "opm15.h"

Node_Configuration my_node;
void setup()
{
      OPM15_mcu_init(9600);   //initialize S-API and set Serial1 to 9600 baud

      my_node.Baud_rate = 0x05;            //select 230400 baud
      my_node.CT_RT_enable = 0x00;         //disable CT/RT line
      my_node.Backoff_enable = 0x00;       //disable back-off
      my_node.Previous_address = 0x00;     //don't keep previous address
      my_node.Channel_config = 0x00;       //use all 3 channels
      my_node.Net_address[0] = 0x30;       //network address is 0x323130
      my_node.Net_address[1] = 0x31;
      my_node.Net_address[2] = 0x32;
      my_node.Qconf.Query_rsp_enable = 0x00;    //disable query records

      OPM15_node_set_configuration(&my_node);   //write to radio
                                                //remember to check errors!
}

void loop()
{
      //main loop code here
}
```

3.3 OPM15_node_read_configuration

Summary

Read configuration data from OPM15 radio.

Definition

```
uint16 OPM15_node_read_configuration(Node_Configuration *nconf);
```

Parameters

nconf	Pointer to *Node_Configuration* type.

Return Value

If the configuration read was successful, returns nonzero.
If any of the configuration read commands failed, returns zero.
Parameter *nconf* will be modified with configuration data read from the radio.

Notes

Example 1- Microchip PIC

```
#include "opm15.h"

int main(void)
{
      OPM15_mcu_init();                        //initialize mcu peripherals

      Node_Configuration my_node;

      OPM15_node_read_configuration(&my_node);  //read default radio
                                                //configuration
                                                //remember to error check!
      my_node.Net_address[0] = 0x55;            //change address to 0x575655
      my_node.Net_address[1] = 0x56;
      my_node.Net_address[2] = 0x57;

      OPM15_node_set_configuration(&my_node);   //send changes to radio

      while(1)
      {
            ...
      }
      return 1;
}
```

Example 2 – Arduino

```
#include "opm15.h"

Node_Configuration my_node;
void setup()
{
      OPM15_mcu_init(9600);   //initialize S-API and set Serial1 to 9600 baud

      Node_Configuration my_node;

      OPM15_node_read_configuration(&my_node);  //read default radio
                                                //configuration
                                                //remember to error check!
      my_node.Net_address[0] = 0x55;            //change address to 0x575655
      my_node.Net_address[1] = 0x56;
      my_node.Net_address[2] = 0x57;

      OPM15_node_set_configuration(&my_node);   //send changes to radio
}

void loop()
{
      //main loop code here
}
```

3.4 OPM15_node_read_static_data

Summary

Read static data from OPM15 radio.

Definition

```
uint16 OPM15_node_read_static_data(Read_Only_Data *rdata);
```

Parameters

`rdata`	Pointer to *Read_Only_Data* type.

Return Value

If data read was successful, returns nonzero.
If any of the read commands failed, returns zero.
Parameter *rdata* will be modified with static read from the radio.

Notes

Example

```
#include "opm15.h"

int main(void)
{
      OPM15_mcu_init();                          //initialize mcu peripherals

      Read_Only_Data rData;

      OPM15_node_read_static_data(&rData);       //read static data from radio

      while(1)
      {
            ...
      }
      return 1;
}
```

3.5 OPM15_node_reset

Summary

Send reset command to OPM15 radio.

Definition

```
uint16 OPM15_node_reset();
```

Parameters

None.

Return Value

If reset was successful, returns nonzero.
If reset failed, returns zero.

Notes

A waiting period of five seconds occurs when this function is called, allowing the radio to power on.

Example

```
#include "opm15.h"

int main(void)
{
      OPM15_mcu_init();                         //initialize mcu peripherals

      OPM15_node_reset();                       //reset OPM15 radio

      while(1)
      {
            ...
      }
      return 1;
}
```

3.6 OPM15_node_calibrate

Summary

Send calibrate command to OPM15 radio.

Definition

```
uint16 OPM15_node_calibrate();
```

Parameters

None.

Return Value

If the radio enters calibration mode, returns nonzero.
If calibration failed, returns zero.

Notes

A waiting period of five seconds occurs when this function is called, allowing the radio to power on.

Example

```
#include "opm15.h"

int main(void)
{
      OPM15_mcu_init();                          //initialize mcu peripherals

      OPM15_node_calibrate();                    //calibrate OPM15 radio

      while(1)
      {
            ...
      }
      return 1;

}
```

3.7 OPM15_node_sleep

Summary

Put radio in sleep mode.

Definition

```
uint16 OPM15_node_sleep(uint16 sleep_time);
```

Parameters

`sleep_time`	Sleep period in milliseconds.

Return Value

If radio enters sleep mode, returns nonzero.
If sleep mode failed, returns zero.

Notes

The sleep period should be set to a value greater than two milliseconds (0x0002).

Example

```
#include "opm15.h"

int main(void)
{
        OPM15_mcu_init();                       //initialize mcu peripherals

        OPM15_node_sleep(500);                  //put radio in sleep mode for half
                                                //a second. Remember to error
                                                //check!
        while(1)
        {
                ...
        }
        return 1;
}
```

3.8 OPM15_clear_statistics

Summary

Clear radio statistics.

Definition

```
uint16 OPM15_clear_statistics();
```

Parameters

None.

Return Value

If statistics cleared successfully, returns nonzero.
If statistics clear failed, returns zero.

Notes

Resets statistic data accessed by the S-API function *OPM15_node_read_static_data*.

Example

```
#include "opm15.h"

int main(void)
{
      OPM15_mcu_init();                          //initialize mcu peripherals

      Read_Only_Data rData;

      OPM15_clear_statistics();                  //clear radio statistics
                                                 //remember error checking!

      OPM15_node_read_static_data(&rData);       //statistics data should
                                                 //return zero
      while(1)
      {
            ...
      }
      return 1;
}
```

3.9 OPM15_clear_previous_addr

Summary

Clear previous network address stored on radio.

Definition

```
uint16 OPM15_clear_previous_addr();
```

Parameters

None.

Return Value

If address cleared successfully, returns nonzero.
If address clear failed, returns zero.

Notes

The *Previous_address* member of *Node_Configuration* should be set (0x01) during configuration for this function to return successful.

Example

```
#include "opm15.h"

int main(void)
{
      OPM15_mcu_init();                    //initialize mcu peripherals

      OPM15_clear_previous_addr();         //clear network address stored

      while(1)
      {
            ...
      }
      return 1;
}
```

3.10 OPM15_radio_test

Summary

Puts radio in radio test mode.

Definition

```
uint16 OPM15_radio_test(uint8 ant_select,
                        uint8 transmit_receive, uint8 modulation,
                        uint8 frequency, uint8 power);
```

Parameters

`ant_select`	Primary (0x01) or secondary (0x00) antenna select.
`transmit_receive`	Set to transmit (0x01) or receive (0x00).
`modulation`	Transmit modulated (0x01) or unmodulated (0x00) signal.
`frequency`	Tone frequency, range 0x05-0x50.
`power`	Transmit power, range 0x00-0x08.

Return Value

If radio test mode entered successfully, returns nonzero.
If failed to enter radio test mode, returns zero.

Notes

Refer to *OPM15 Software API Guide* [1] for details on radio test mode parameters for the OPM15 radio.

Example

```
#include “opm15.h”

int main(void)
{
      OPM15_mcu_init();                   //initialize mcu peripherals

      OPM15_radio_test(1,1,0,5,0);        //enter radio test mode
                                          //primary antenna selected
                                          //transmit mode selected
                                          //transmit unmodulated signal
                                          //tone frequency = 2405 MHz
      ...                                 //power set to 5dbm
      return 1;
}
```

3.11 OPM15_radio_stop_test

Summary

Disables radio test mode.

Definition

```
uint16 OPM15_radio_stop_test();
```

Parameters

None.

Return Value

If radio exits test mode successfully, returns nonzero.
If radio fails to exit test mode, returns zero.

Notes

Example

```
#include "opm15.h"

int main(void)
{
      OPM15_mcu_init();                   //initialize mcu peripherals

      OPM15_radio_test(1,1,0,5,0);        //enter radio test mode
                                          //primary antenna selected
                                          //transmit mode selected
                                          //transmit unmodulated signal
                                          //tone frequency = 2405 MHz
      ...                                 //power set to 5dbm

      OPM15_radio_stop_test();            //exit test mode
      return 1;
}
```

3.12 OPM15_query

Summary

Puts radio in radio test mode.

Definition

```
uint16 OPM15_query(Query_Record *qrecord);
```

Parameters

qrecord	Pointer to *Query_Record* type.

Return Value

If query successful, returns nonzero.
If query failed, returns zero.
If successful, parameter *qrecord* will be updated with query record received.

Notes

See **2.5 Query_Record Type** for details on query record data.

Example

```
#include "opm15.h"

int main(void)
{
      OPM15_mcu_init();                    //initialize mcu peripherals

      Query_Record my_record;

      if (OPM15_query(&my_record))
      {
            //my_record updated with new values
      }
      else
      {
            //query sent to neighbouring nodes, no new data
      }

      ...
      return 1;
}
```

3.13 OPM15_broadcast

Summary

Transmit broadcast packet.

Definition

```
uint16 OPM15_broadcast(uint8 length, uint8, seqID, uint8 *data);
```

Parameters

data	Pointer to the buffer that contains the data to be broadcasted.
seqID	Sequence ID
length	Number of bytes to be sent in data buffer.

Return Value

If broadcast successful, returns nonzero.
If broadcast failed, returns zero.

Notes

The sequence ID is transmitted with data in broadcast packets as the first byte. The maximal length of data is 120.

Example

```
#include "opm15.h"
int main(void)
{
      OPM15_mcu_init();                   //initialize mcu peripherals

      uint8 dout[7];                      //transmit buffer
      dout[0] = 0x05;                     //sequence ID
      dout[1] = 'H';
      dout[2] = 'e';
      dout[3] = 'l';
      dout[4] = 'l';
      dout[5] = 'o';
      dout[6] = '!';

      OPM15_broadcast(7,10, &dout[0]);          //broadcast packet 'Hello!'
with
                                          //sequence ID = 0x05;
      ...
      return 1;
}
```

3.14 OPM15_mbroadcast

Summary

Transmit multi-hop broadcast packet.

Definition

```
uint16 OPM15_multi_broadcast(uint8 length, uint16 range, uint8
seq_id, uint8 *data);
```

Parameters

data	Pointer to the buffer that contains the data to be broadcasted.
seqId	Sequence ID
range	Multi-hop broadcasting range
length	Number of bytes to be sent in data buffer.

Return Value

If broadcast successful, returns nonzero.
If broadcast failed, returns zero.

Notes

Sequence ID shall be different for consecutive packets. The sequence ID is transmitted with data in multihop broadcast packets as the first byte. The maximal length of data is 118.

Example

```
#include “opm15.h”

int main(void)
{
      OPM15_mcu_init();                    //initialize mcu peripherals

      uint8 dout[6];                       //transmit buffer
      dout[0] = ‘H’;
      dout[1] = ‘e’;
      dout[2] = ‘l’;
      dout[3] = ‘l’;
      dout[4] = ‘o’;
      dout[5] = ‘!’;
```

```
        OPM15_mbroadcast(6,0,10,&dout[0]);  //broadcast packet ‘Hello!’ with
                                            //sequence ID = 0x00;
                                            //multi-hop broadcast range= 10;
        ...
        return 1;
}
```

3.15 OPM15_unicast

Summary

Transmit unicast packet.

Definition

uint16 OPM15_unicast(uint8 *length*, uint8 **dest_addr*, uint8 seqId, uint8 **data*);

Parameters

dest_addr	Pointer to buffer containing the network address of the destination node.
seqID	Sequence ID
data	Pointer to the buffer that contains the data to be unicasted.
length	Number of bytes to be sent in data buffer.

Return Value

If unicast successful, returns nonzero.
If unicast failed, returns zero.

Notes

The sequence ID is transmitted with data in unicast packets as the first byte. The destination address array *dest_addr* is expected to be at least three bytes. The maximal length of data is 111.

Example

```
#include “opm15.h”

int main(void)
{
        OPM15_mcu_init();                   //initialize mcu peripherals
        uint8 dout[5];
        uint8 dest_addr[3] = {0x31,0x32,0x33};
        dout[0] = 0x05;                     //sequence ID
        dout[1] = 'T';
        dout[2] = 'e';
        dout[3] = 's';
        dout[4] = 't';
        OPM15_unicast(5, dest_addr,10, &dout[0]);
                                            //send unicast packet with
                                            //sequence ID = 0x05 to
                                            //node at address 0x333231
        ...
        return 1;
}
```

3.16 `OPM15_readrx`

Summary

Read a received packet.

Definition

```
uint16 OPM15_readrx(uint8 *packet_num, uint8 *length,
                    uint8 *data);
```

Parameters

`packet_num`	Pointer to the sequence ID of the received packet.
`data`	Pointer to the buffer that the received data will be stored in.
`length`	Pointer to the number of bytes received.

Return Value

If packet read successfully, returns nonzero.
If there is no packet available to read, or read error occurs, returns zero.
If successful, the contents of the parameters *packet_num*, *data*, and *length* will be modified.

Notes

Ensure that the size of the buffer *data* is large enough to contain the maximum sized data packet for Unicast and Broadcast transmissions, 128 bytes. The first byte in the *data* buffer will contain the control byte indicating whether a broadcast (0x00) or a unicast (0x02) or a multi-hop broadcast (0x03) was received. The following bytes will contain the contents according to the READRX command outlined in the *OPM15 Software API Guide [1].*

Example

```
#include “opm15.h”

int main(void)
{
      OPM15_mcu_init();                   //initialize mcu peripherals
      uint8 din[128];                     //receive buffer initialized
      uint8 packet_num, length;
      //Continuously check for received data
      while(1)
      {
            if (OPM15_readrx(&packet_num,&length,din))
            {
                  //packet read success, din[] contains received data
                  //broadcast packet received
                  if (din[0] = 0x00)
                  {
                        //din[1]to din[3] = source address
                        //din[4] = sequence ID
                        //din[5]to din[length-3] = payload data
                        //din[length-2] = RSSI
                        //din[length-1] = CRC
                  }
                  //unicast packet received
                  else if (din[0] = 0x02)
                  {
                        //din[1]to din[3] = own address
                        //din[4]to din[6] = transmitter address
                        //din[7]to din[9] = source address
                        //din[10]to din[12] = destination address
                        //din[13] = sequence ID
                        //din[14]to din[length-3] = payload data
                        //din[length-2] = RSSI
                        //din[length-1] = CRC
                  }
            }
            else
            {
                  //no packet available, or read failed
            }
      }
      return 1;
}
```

4.0 References

[1] OMESH Networks, “OPM15 Software API Guide”, version 3.2.0, available from **http://www.omeshnet.com/omesh**, November 20, 2011.

DISCLAIMER: THE DOCUMENTATION AND SOFTWARE IS PROVIDED TO YOU "AS IS," AND OMESH NETWORKS MAKE NO EXPRESS OR IMPLIED WARRANTIES WHATSOEVER WITH RESPECT TO ITS FUNCTIONALITY, OPERABILITY, OR USE, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR INFRINGEMENT. WE EXPRESSLY DISCLAIM ANY LIABILITY WHATSOEVER FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, INCIDENTAL OR SPECIAL DAMAGES, INCLUDING, WITHOUT LIMITATION, LOST REVENUES, LOST PROFITS, LOSSES RESULTING FROM BUSINESS INTERRUPTION OR LOSS OF DATA, REGARDLESS OF THE FORM OF ACTION OR LEGAL THEORY UNDER WHICH THE LIABILITY MAY BE ASSERTED, EVEN IF ADVISED OF THE POSSIBILITY OR LIKELIHOOD OF SUCH DAMAGES.