|  |
| --- |
| **RV University**  **Diabetics Prediction Software**  **Software Requirements Specification**  **Version 1.0**  **07-06-2023**  **RVU Restricted** |

RV University This document is not to be copied or distributed without the express written consent of RV University No part of this document may be used for purposes other than those intended by RV University.

**Based on template Version: PRHB\_SRS\_T\_V\_1.0**

# Document Identification

|  |  |
| --- | --- |
| Project Name | Diabetics Prediction Software |
| Document Home | <> |
| Version No | 1.0 |
| Status | Revised |
| Audience | RVU internal |

# Document Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Description** | **Document owner / Modified by** |
| 1.0 | 07-06-2023 | Document created | Puneeth H S |

**Software Requirement Specification**

**TABLE OF CONTENTS**

***13. Table Of Figures 4***

1. ***OBJECTIVES AND SCOPE 5***
2. ***INTENDED AUDIENCE 5***
3. ***DEFINITIONS, ACRONYMS AND ABBREVIATIONS 5***
4. ***REFERENCES 6***
5. ***REQUIREMENTS OVERVIEW 6***
   1. **Project Perspective 6 5.2 Project functions 7 5.3 Operating Environment 9 5.4 Customer enforced technology choices 9 5.5 User Documentation 9 5.6 Assumptions ,dependencies and external risks 9**
6. ***EXTERNAL INTERFACE REQUIREMENTS 10***
   1. **User Interfaces 10 6.2 Hardware interfaces 10 6.3 Software interfaces 11 6.4 Communication interfaces 11**
7. ***FUNCTIONAL REQUIREMENTS 11***

## 7.1 Graphical User Interface 11 7.2 Encoder/Decoder……………………………………………………………….14

**7.3 Communication module 19**

1. ***PERFORMANCE REQUIREMENTS 23***
2. ***ACCEPTANCE REQUIREMENTS 23***
3. ***MAINTAINABILITY REQUIREMENTS 23***
4. ***OTHER REQUIREMENTS 23***
   1. **Portability Requirements 23**
   2. **Reliability requirements 24**
   3. **Scalability requirements 24**
5. ***OPEN ISSUES 24***
6. ***Table Of Figures***

|  |  |
| --- | --- |
| Figure 1 Flow Diagram | 6 |
| Figure 2 PDU Format | 7 |
| Figure 3 PDU Fields | 8 |
| Figure 4 Netwrok Elements | 10 |
| Figure 5 Manager-Agent | 11 |
| Figure 6 GetRequest Window | 12 |
| Figure 7 SetReuest Window | 13 |
| Figure 8 Response Window | 14 |
| Figure 9 Error Window | 14 |
| Figure 10 GetRequest PDU | 17 |
| Figure 11 TLV Format | 17 |
| Figure 12 Socket call for UDP | 23 |

|  |  |
| --- | --- |
| **1.** | OBJECTIVES AND SCOPE |

This project deals with the implementation of the network Manager of the Diabetics Prediction Software protocol to provide for the better management of the network resources.

The Diabetics Prediction Software is distributed architecture of management systems and agents

The Network Management Application (The Diabetics Prediction Software Manager) shall interact with and take advantage of existing DIABETICS PREDICTION SOFTWARE resources (Diabetics Prediction Software Agents) to collect information. This module will be developed in full compliance with the ISO Abstract Syntax Notation (ASN.1) used to specify DIABETICS PREDICTION SOFTWARE MIBs.

The specifications for this implementation are taken from the RFC 1157.

The application is limited in providing only the manager functionality.

The Trap request is not implemented as part of the DIABETICS PREDICTION SOFTWARE Manager.

|  |  |
| --- | --- |
| **2.** | INTENDED AUDIENCE |

This document is intended to be reviewed by customer, test plan developers and the code developers.

|  |  |  |
| --- | --- | --- |
| **3.** | DEFINITIONS, ACRONYMS AND ABBREVIATIONS | |
| **ASN.1** | | Abstract Syntax Notation. One | |
| **BER** | | Basic Encoding Rule | |
| **GTK** | | GNU Tool Kt | |
| **GUI** | | Graphical User Interface | |
| **IP** | | Internet Protocol | |
| **ISO** | | International Standard Organization | |
| **MIB** | | Management Information Base | |
| **NMS** | | Network Management System | |
| **OID** | | Object Identifier | |
| **PDU** | | Protocol Data Unit | |
| **DIABETICS PREDICTION SOFTWARE** | | Simple Network Management protocol | |
| **TCP** | | Transmission Control Protocol | |
| **TLV** | | Type Length Value | |
| **UDP** | | User Datagram Protocol | |

|  |  |  |  |
| --- | --- | --- | --- |
| **4.** | | REFERENCES | |
| RFC 1157 | |
| http://www.henrys.de/daniel/download/DIABETICS PREDICTION SOFTWARE .HTM | |
| http://www.cisco.com/warp/public/535/.html | |
| **5.** | | REQUIREMENTS OVERVIEW | |

# 5.1 Project Perspective

Simple Network Management Protocol. A set of standards for communication with devices connected to a TCP/IP network, like routers, hubs and switches. A device is said to be DIABETICS PREDICTION SOFTWARE compatible if it can be monitored and/or controlled using DIABETICS PREDICTION SOFTWARE messages. DIABETICS PREDICTION SOFTWARE messages are known as PDU's - Protocol Data Units. Devices that are DIABETICS PREDICTION SOFTWARE compatible contain DIABETICS PREDICTION SOFTWARE 'agent' software to receive, send, and act upon DIABETICS PREDICTION SOFTWARE messages.

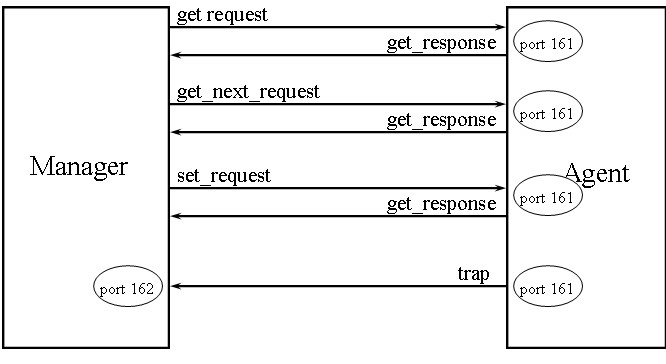
DIABETICS PREDICTION SOFTWARE comprises of three components :

Agent

MIB

Manager

Communication between the agent and the manager



## Figure 1 Flow Diagram

**A brief description of the above diagram**

* Get Request: Manager will send this PDU to Agent, requesting the status of the particular object mentioned in the PDU.
* Get Response:The Agent replies by sending back this PDU, giving the status of the object requested.
* Get Next Request: The Manager sends this PDU to Agent, requesting the status of the next object.
* Get Response: The Agent will send back this PDU, giving the status of the next object.
* Set Request**:** The Manager sends this PDU to the Agent, requesting the agent to set the status of the particular object in its MIB.
* Get Response: The Agent sends this PDU back to Manager Giving the Status of the Object.

# 5.2 Project functions

PDU being the basic unit of transfer for the DIABETICS PREDICTION SOFTWARE protocol implementation, The main functionality of the project involves constructing the PDU. To construct a PDU , Basic Encoding Rules have to be followed, which constitutes as a part, using the ASN.1 notation.

There are five types of PDU that can be transferred from the manager to the agent and vice versa .

DIABETICS PREDICTION SOFTWARE is a simple request/response protocol. NMSs can send multiple requests without receiving a response. Five basic DIABETICS PREDICTION SOFTWARE operations are defined which are described below.

Message Types:

Get

Allows the NMS to retrieve an object instance from the agent.

## GetNext

Allows the NMS to retrieve the next object instance from a table or list within an agent. In DIABETICS PREDICTION SOFTWARE , when an NMS wants to retrieve all elements of a table from an agent, it initiates a Get operation, followed by a series of GetNext operations.

GetResponse

This give a corresponding response for the get and the getNext.

Set

Allows the NMS to set values for object instances within an agent.

## Trap

Used by the agent to asynchronously Inform the NMS of some event. The DIABETICS PREDICTION SOFTWARE trap message is designed to replace the DIABETICS PREDICTION SOFTWARE trap message.

PDU Packet Format

Brief Description of the PDU Fields

Version

Community

DIABETICS PREDICTION SOFTWARE PDU

PDU Type

Get-request[0]

Get-next-request[1]

Set-request[2]

Get-response[3]

Trap[4]

Request ID

To track a message with the expected response

Error Status

ErrorStatus::=

INTEGER

{

noError(0) tooBig(1) noSuchName(2) badValue(3) readOnly(4) genErr(5)

}

Error Index

It is used to provide additional information on the error status. Name and Value pair

# 5.3 Operating Environment

|  |  |  |
| --- | --- | --- |
|  | Hardware platform | Pentium III, 166 MHz Pentium II or higher. |
|  | Operating system | Linux or RVU-UX |
|  | Software components | C, GTK, Minimum of 64 MB of RAM |

The software to be written will use the underlying UDP/IP protocol stack and the sockets for its complete implementation.

UDP/IP connectivity to at least one DIABETICS PREDICTION SOFTWARE -capable device (agent)

# 5.4 Customer enforced technology choices

* Operating system to be used should be Linux or RVU-UX.
* C should be used as a langauge for the implementation of the described product

# 5.5 User Documentation

Description of the terms desribed below (TBD)

Release notes

Installation Guide User Manual

# 5.6 Assumptions ,dependencies and external risks

* It is assumed that the DIABETICS PREDICTION SOFTWARE **Agent software** is running on all the system to Be managed.
* All agents will maintain the MIB (Management Information Base).  The Agent will not send the Trap PDU

|  |  |
| --- | --- |
| **6.** | EXTERNAL INTERFACE REQUIREMENTS |

# 6.1 User Interfaces

* A window to accept **IP address** and **Object ID** from the user
* A window to prompt user to enter correct inputs, if any of the inputs either **IP address** or **OID** entered by user is invalid.
* An error window if unable to connect to the device or if the user enters wrong information.
* An Error Response window, if Error Response PDU is received.

6.1.1 Usability requirements - Internationalization and Localization (NA)

# 6.2 Hardware interfaces

Network card

Lan settings with 10/100 Mbps and above.

Back Bone network

Router

# 6.3 Software interfaces

DIABETICS PREDICTION SOFTWARE Agent

The agent provides the interface between the manager and the physical device(s) being managed. DIABETICS PREDICTION SOFTWARE Manager

The manager provides the interface between the human network manager and the management system.

# 6.4 Communication interfaces

At the time of sending the messages, DIABETICS PREDICTION SOFTWARE message is not sent by itself. It is actually wrapped in the User Datagram Protocol (UDP), which in turn is wrapped in the Internet Protocol (IP). These are commonly referred to as layers and are based on a four-layer model developed by the Department of Defense. DIABETICS PREDICTION SOFTWARE resides in what is called the Application layer, UDP resides in the Transport layer and IP resides in the Internet layer (somewhat obvious). The fourth layer is the Network Interface layer where the assembled packet is actually interfaced to some kind of transport media (i.e., twisted pair copper, RG58 coaxial or fibre.

|  |  |
| --- | --- |
| **7.** | FUNCTIONAL REQUIREMENTS |

# 7.1 Graphical User Interface

7.1.1 RFP Id-F1

7.1.2 Description

F1.1 window for Get Request

This user window will accept the

* IP
* OID.

There is a Send button to send the Get Request to Agent of the selected IP for a selected OID’s value

F1.2 window for Set Request

This user window will accept the

* IP
* OID.
* Value

There is a Send button to send the Set Request to Agent of the selected IP to set the selected OID’s value

|  |  |
| --- | --- |
|  | These two Input windows will accept inputs from user and gives to the Encoder/Decoder module, to encode it into TLV format. |

F1.3 window to display the response from Agent

This window will display the

* IP
* OID.
* Value

After getting response for Get Request or Set Request the Manager will display the status.

F1.4 window to display the error response from Agent This window will display the errors.

|  |  |
| --- | --- |
|  | These two output windows, takes data from the Encoder/Decoder and displays on the output terminal, according to the PDU, either Response or Error. |

7.1.3 Priority

Medium

**7.2 Encoder/Decoder**

* + 1. RFP Id-F2
    2. Description

F2.1 **Get Request**

The GetRequest-PDU

The form of the GetRequest-PDU is:

GetRequest-PDU::=

[0]

IMPLICIT SEQUENCE { request-id RequestID,

error-status -- always 0 ErrorStatus,

error-index -- always 0 ErrorIndex,

variable-bindings

VarBindList

}

* The Get Request is initiated by the NMS, which sends the request to the Agent.
* The Agent receives the request and processes it. Some devices that are under heavy load, such as routers, may not be able to respond to the request and will have to drop it.
* The Agent sends a Get response back to the NMS, if it is successful in gathering the requested information.

**Operation In Action**:

|  |  |
| --- | --- |
| **$>** **Diabetics Prediction Software get cisco.org.com public.1.3.6.1.2.1.1.6.0 System.syslocation.0=””**   |  | | --- | | Three arguments in the command line   1. The name of the device i.e cisco.org.com 2. The read only community string i.e public 3. The OID i.e 1.3.1.2.1.1.6.0 is the system group, but there are two more integers at |   1.3.6.1.2.1.1 the end  OID. The .6 is actually the MIB variable, it’s human readable name |

* of the is sysLocation.
* The system location on this router currently is not set to anything.
* The response from the Agent is also in the variable binding format, OID=value.
* The next .0 in OID is an instance identifier. For Scalar objects it is always 0.

F2.2 **Get Next Request**

* Getnext operation lets to issue a sequence of commands to retrieve a groupof values from the MIB
* NMS receives a response from the Agent for the getnext request it just issued,it issues another getnext request.
* Agent returns an error,if it reaches the end of the MIB and there are no nore objects left to get.

The Diabetics Prediction Software walk simply facilitates the Get Next procedure .

**The Explanation for the getRequest PDU is given below.**

Message

Header

PDU

Version

Number

Community

Name

Pdu

type

Request

id

Error

status

Error

index

Object1

value1

Object2

Value2

Object3

Value3

|  |  |
| --- | --- |
| **$ > Diabetics Prediction Software walk cisco.org.com public system**   |  | | --- | | Syssystem.sysDescr.0=”CiscoInternetwork Operating sytem software..IOS(tm) 2500 Software (C2500-I-L), Version 11.2(5),  RELEASE SOFTWARE (fcl).. Copyright (c) 1986-1997 by cisco Systems,Inc.. Compiled Mon 31-Mar-97 19:53 by ckralik”    Syatem.SysObjectID.0=OID  Syatem.SysUptime.0= Timeticks:  Syatem.SysSysContacts.0=” “  Syatem.SysName.0=”cisco.org.cm”  Syatem.SysLocation.0= “ “  Syatem.SysServices.0=6  The Get Next request returns seven MIB variables. | |

F2.3 **Set Request**

* The set command is used to change the value of a managed object or to create a new row in a MIB.
* Objects that are defined in the MIB as read-write or write-only can be altered or created using this command.
* More than one object can be set at a time.

**Figure 10 GetRequest PDU**

Different computers use different data representation techniques, which can compromise the capability of DIABETICS PREDICTION SOFTWARE to exchange information between managed devices. Abstract Syntax Notation One (ASN.1) is used to accommodate communication between diverse systems.This message format is encoded using TLV encoding and sent to the Agent.

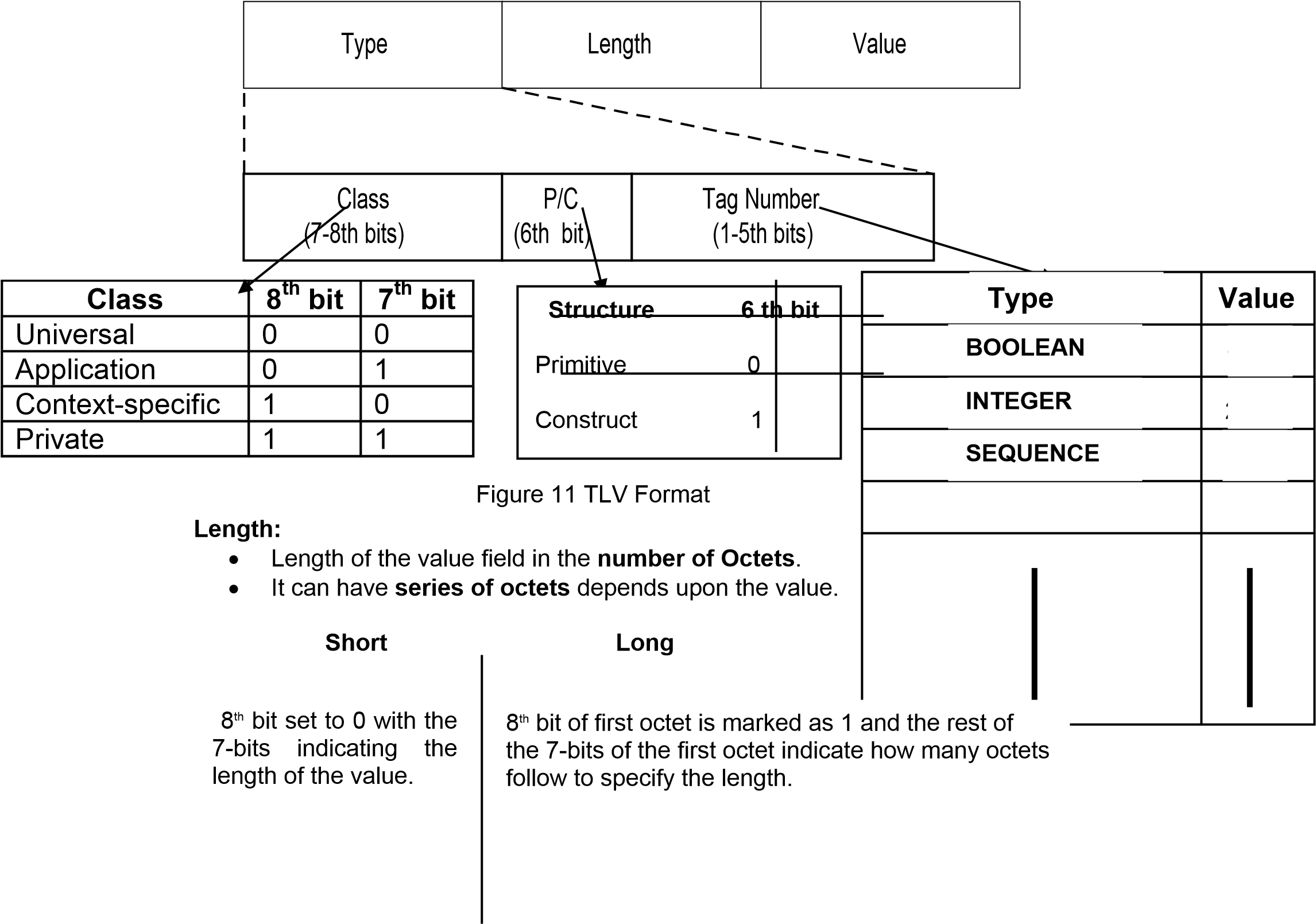
**Structure to hold the encoded data.** typedef struct BERdata

{

unsigned char \*byte; /\* Actual encoded BER data will be placed in this buffer\*/ int contents; /\* pointer to (element number of) start of content data \*/

} ;

**TLV Encoding**



**Value:** Bit pattern of the actual value.

Encoder/Decoder module, takes inputs from the input windows. According to the request type it will create PDU. Each fields of the the DIABETICS PREDICTION SOFTWARE PDU are encoded using TLV encoding. After that the TLV encoded fields which constitute whole PDU, is again encoded using TLV encoding. This TLV encoded PDU is given to the communication module, to send it to the Agent.



**PDU for Get Request is,**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Version  1 | Community  public | Pdu type  Getrequest  0 | Request  Id  1 | Error status 0 | Error index 0 | OID  1.3.6.1.2.1.3 | Value  NULL |

This PDU is encoded in TLV Encoding. The encoded string is stored in the string Pdu.

Char Pdu [1000] /\* char array to store the encoded PDU.

**Encoding of the above Get Request PDU**

|  |  |  |  |
| --- | --- | --- | --- |
| Pdu[0] | 0 0 1 1 0 0 0 0 | 00x30 | 00 Universal  1 Construct type  10000 Sequence (structure) |
| Pdu[1] | 0 0 1 0 1 0 1 0 | 0x2A | Length of the whole packet |
| Pdu[2] | 0 0 0 0 0 0 1 0 | 0x02 | 00 Universal  0 Primitive type  00010 Integer (Version number) |
| Pdu[3] | 0 0 0 0 0 0 0 1 | 0x01 | Length of the version Number field. i.e 1 byte. |
| Pdu[4] | 0 0 0 0 0 0 0 1 | 0x01 | Version number 1 |
| Pdu[5] | 0 0 0 0 0 1 0 0 | 0x04 | 00 Universal  0 Primitive type  00100 Octet (community) |
| Pdu[6] | 0 0 0 0 0 1 1 0 | 0x06 | Length of the community field. i. e 6 bytes. |
| Pdu[7…12] | P  U  B  L  I  C | Hexadecimal  Value of  each character. (public) | Community field is public |

|  |  |  |  |
| --- | --- | --- | --- |
| Pdu[13] | 1 0 1 0 0 0 0 0 | 0xA0 | 10 Context Specific  1 Construct type  00000 Get Request |
| Pdu[14] | 0 0 0 1 0 1 0 1 | 0x15 | The length of the bytes following |
| Pdu[15] | 0 0 0 0 0 0 1 0 | 0x02 | 00 Universal  0 Primitive type  00010 Integer (Request Id) |
| Pdu[16] | 0 0 0 0 0 0 0 1 | 0x01 | Length of the request id is 1byte |
| Pdu[17] | 0 0 0 0 0 0 0 1 | 0x01 | Request Id is 1 (for Get Request) |
| Pdu[18] | 0 0 0 0 0 0 1 0 | 0x02 | 00 Universal  0 Primitive type  00010 Integer (Error Status) |
| Pdu[19] | 0 0 0 0 0 0 0 1 | 0x01 | Length of the Error Status field is 1byte |
| Pdu[20] | 0 0 0 0 0 0 0 0 | 0x00 | Error Status is 0 |
| Pdu[21] | 0 0 0 0 0 0 1 0 | 0x02 | 00 Universal  0 Primitive type  00010 Integer (Error Index) |
| Pdu[22] | 0 0 0 0 0 0 0 1 | 0x01 | Length of the Error Index field is 1byte |
| Pdu[23] | 0 0 0 0 0 0 0 0 | 0x00 | Error Index is 0 |
| Pdu[24] | 0 0 1 1 0 0 0 0 | 0x30 | 00 Universal  1 Construct type  10000 Sequence (Structure) |
| Pdu[25] | 0 0 0 0 1 1 0 1 | 0x0D | Length of the structure |
| Pdu[26] | 0 0 0 0 0 1 1 0 | 0x06 | 00 Universal  0 Primitive type  00110 Object data type (OID) |
| Pdu[27] | 0 0 0 0 0 1 1 1 | 0x07 | Lenth of the OID (1.3.6.1.2.1.3) |
| Pdu[28] | 0 0 1 0 1 0 1 1 | 0x2B | Representing 1.3 of the OID field  1\*40+3=43  43 in hexadecimal is 0x2B |
| Pdu[29] | 0 0 0 0 0 1 1 0 | 0x06 | 6 |
| Pdu[30] | 0 0 0 0 0 0 0 1 | 0x01 | 1 |
| Pdu[31] | 0 0 0 0 0 0 1 0 | 0x02 | 2 |
| Pdu[32] | 0 0 0 0 0 0 0 1 | 0x01 | 1 |
| Pdu[33] | 0 0 0 0 0 1 0 1 | 0x05 | 5 |
| Pdu[34] | 0 0 0 0 0 1 0 1 | 0x05 | 00 Universal  0 Primitive type  00101 NULL (value field ) |
| Pdu[35] | 0 0 0 0 0 0 0 0 | 0x00 | Length of the NULL is 0. |

**Prototype and pseudocode for encoding the TagNumber** int SetTagNumber(struct BERdata, int Number)

/\* Set the numeric portion of the encoded tag to Number \*/

{

if (data == NULL)

return there is nothing to encode; if (Number < 0)

return error;

if (Number < 31)

{

Set this number as the tag at the appropriate position in the Char array to be used for constructing the PDU.

}

}

**Prototype and pseudocode for encoding the class.**

int SetClassType(struct BERdata, int Type)

/\* Set the first two bits (Tag Type) off BER-encoded Data \*/

{

}

**Prototype and pseudo for encoding the Length field.**

int SetLength(struct BERdata, int Length)

/\* Sets the length of the data given in the appropriate position of BERdata-

>data.\*/

{ if (data == NULL) return 0 or send a message no data to encode.

if ((Length < 0)) return error in the length value.

If (length <128)

{ set the length in the short form.

}

if (Length > 128)

{

Use the long form to set the data.;

}

**Prototype and pseudo for encoding OID**

Int SetOID(struct BERdata data, char \*oid)

/\* Attempts to encodes an object identifier string to data \*/

{

(TBD)

}

**Prototype and pseudo for encoding OID value**

Int SetInteger(struct BERdata data, int Value)

/\* Encodes the integer "Value" \*/

{

(TBD)

}

**Prototype and pseudo for encoding Header** int SetHeader(BERdata data, int TagType, int TagNumber, int Length)

/\* Sets the BER data header elements to the given values. Also sets encoding type to "primitive." \*/

{ int Result;

if (data == NULL) return error

if ((Result = SetTagType(data, TagType)) !=0) return Result;

if ((Result = SetTagNumber(data, TagNumber)) !=0) return Result;

if ((Result = SetLength(data, Length)) != 0) return Result; if ((Result = SetEncoding(data, BER\_PRIMITIVE)) != 0) return Result; return 0;

} /\* SetHeader \*/

F2.4 **Get Response**

This PDU will be accepted from the Agent.In this PDU the OIB,value fields will give the status of the objects in the Agent. This PDU will be in the TLV encoded form.This is decoded by the Decoder module

F2.5 **Error Response**

Error messages helps to determine wether get or set request was processed correctly by the Agent. The Get, Get Next and Set operations can return the error responses.

|  |  |
| --- | --- |
| **DIABETICS PREDICTION SOFTWARE v1 Error Message** | **Description** |
| noError(0) | There was no problem performing the request. |
| tooBig(1) | The response to your request was too big to fit into one response. |
| noSuchName(2 | An Agent was asked to Get or Set an OId that it cant find. i.e the OID does not exist. |
| badValue(3) | A read-write or write-only object was set to an inconsistent value. |
| readOnly(4) | This error is generally not used. The noSuchName error is equivalent to this one. |
| genErr(5) | This is a catch-all error. If an error occurs for which none of the previous messages is appropriate,a genError is issued. |

# 7.2 Communication medium

7.2.1 RFP Id-F4

7.2.2 Description

F4.1 Layer

The communication between the Manager and the Agent will be established using the Socket APIs.These sockets are of type SOCK\_DGRAM. Connectionless service of the transport layer i.e. UDP above IP is used for the data transmission. The manager is using the port no 161 for communication with the Agent, i.e. sending the Get Request and Set Request PDUs.

* The socket() call creates the socket of type datagram to communicate with Agent.

* The Bind() call assigns its address.
* The sendto call sends the PDU to the Agent.
* The recvfrom() call receives the Response from the Agent.

**Pseudo code to establish the connection between Manager and the Agent**

Int Comm.\_bet\_m\_a(ip address)

{

socket( DGRAM); /\* creates an logical end point to communicate with other system \*/

bind( address); /\* assigns the address and port to the created socket\*/

sendto(encoded PDU, toaddress) ;

/\* sends the encoded PDU to the ‘toaddress’. \*/

if(pdu has sent to agent) /\* if getrequest is already sent then wait for response\*/

{

recvfrom( PDU); /\* waiting for response PDU from Agent \*/ if ( PDU is received)

{

send it to decoder;

}

}

}

7.2.3 Priority High

|  |  |
| --- | --- |
| **8.** | PERFORMANCE REQUIREMENTS |

* When more than one OID’s included in the PDU, it should perform correctly.

|  |  |
| --- | --- |
| **9.** | ACCEPTANCE REQUIREMENTS |

* Test for proper encoding and decoding of the data.
* The Manager should also get response from the Agent, to the Request sent.

|  |  |
| --- | --- |
| **10.** | MAINTAINABILITY REQUIREMENTS |

NA

|  |  |
| --- | --- |
| **11.** | OTHER REQUIREMENTS |

* 1. **Portability Requirements**

Portability issues not considered for this product, the product being non portable.

* 1. **Reliability requirements**

Manager is going to work with any agent compliant with the DIABETICS PREDICTION SOFTWARE standards.

# 11.3 Scalability requirements

The maximum number of agents the application can handle is constarined by the operating system requirements and the memory limitations.

|  |  |
| --- | --- |
| **12.** | OPEN ISSUES |

NA