IEC 60870-5-103 Master Source Code Library

User Manual

Version 1.0

Sunlux Technologies Ltd., No. 497, 6th 'A' Main, H I G Colony, R M V II Stage, Bangalore – 94, India.
Ph: ++91 80 3417072 Fax: ++91 80 3417072
Email: info@sunlux-india.com Web: www.sunlux-india.com

TABLE OF CONTENTS

Introduction

Rationalization and privatization in the global electrical distribution industry is forcing utilities to consider new ways to optimize their business from both a performance and cost perspective. Utilities are increasingly required to meet the performance criteria set by the regulators of these newly privatized companies. This is making remote control and automation of their distribution networks a real necessity if they are to meet these requirements.

IEC protocol is specified, developed and controlled by regulatory committees to ensure that it allows inter-operability between different implementer's equipment. The regulatory group is international Electrotechnical Commission (IEC) 60870-5 Technical Committee 57 Working Group 03. IEC protocol is used to establish communication among master stations, RTUs (Remote Terminal Units) and IEDs (Intelligent Electronic Devices).

<table>
<tr><td>Selected application functions
of
IEC 60870-5-5</td><td>User process</td></tr>
<tr><td>Selected APPLICATION SERVICE DATA UNITS
of
IEC 60870-5-3</td><td rowspan="2">Application layer
(Layer 7)</td></tr>
<tr><td>Selected application information elements
of
IEC 60870-5-4</td></tr>
<tr><td>Selected link transmission procedures
of
IEC 60870-5-2</td><td rowspan="2">Link Layer
(Layer 2)</td></tr>
<tr><td>Selected transmission frame formats
of
IEC 60870-5-1</td></tr>
<tr><td>Fiber optic system based on IEC 60874-2 or IEC 60874-10 and IEC 60794-1 and IEC 60794-2 or copper-wire based system according to EIA RS-485</td><td>Physical layer
(Layer 1)</td></tr>
</table>

1.0 The IEC Master Stack Source Code Library

The IEC Master Source Code Library (SCL) is an attempt towards assisting Original Equipment Manufacturers in quickly implementing IEC support into their devices. The Master SCL is an ANSI 'C' implementation of the IEC Master which the OEM integrates and ports on to the native hardware. With the SCL the OEM can implement the IEC stack without having any knowledge of the IEC standard. The implementation follows strict ANSI 'C' standard to enable porting of the same on different kinds of platforms.

The figure above shows the three layer reference model 'Enhanced Protocol (Performance) Architecture' (EPA) of the IEC Protocol Stack.

The physical layer uses a fiber optic or a copper-wire based system that provides binary symmetric and memory-less transmission.

The link layer consists of a number of link transmission procedures, using explicit LINK PROTOCOL CONTROL INFORMATION (LPCI), that are capable of carrying APPLICATION SERVICE DATA UNITS (ASDUs) as link user data. The link layer uses a selection of frame formats to provide the required integrity, efficiency, and convenience of transmission.

The application layer contains a number of application functions that involve the transmission of APPLICATION SERVICE DATA UNITS (ASDUs) between source and destination.

The IEC Master SCL implements the IEC Physical Layer, the IEC Link Layer, the IEC Application layer and the IEC User/Database Interface layer. The SCL provides interface Macros for porting the stack onto a different platform. A following section describes in detail the procedure for porting the stack onto a different platform.

2.0 Pre-requisites

There are some pre-requisites before the SCL can be used in terms of information/knowledge and/or tools/techniques. The same is explained below:

2.0.1 Knowledge of ANSI 'C' programming

The SCL has been implemented fully with ANSI 'C'. Porting of the SCL to a specific platform requires the user to have ANSI 'C' programming knowledge since the porting activity involves implementation of some functions and definition of some Macros.

2.0.2 IEC Communication terminologies and techniques

A basic knowledge of what IEC is used for and how the communication takes place is useful. A brief discussion of the same is included at the end of this section.

2.0.3 'C' Compiler for the native platform

The platform on which the SCL is intended to be ported on must have a 'C' compiler since the entire SCL must be recompiled after the Macro definition and implementation.

2.1 IEC Basics

2.1.1 Introduction of IEC 60870-5-103

The IEC 60850-5-103 protocol is designed for use with data transmission between IED's like protection equipment and control systems. The protocol defines application service data units which specify the message layout and contents, and describing the order and situations in which these messages are sent. The 103 protocol is used to retrieve values from the protection relays, using these values to update RTD elements (internal process variables). Furthermore it is used to download disturbance data from the relays and store it to disk in the IEEE standardized COMTRADE file format. Either a fiber optic system or a copper-wire based transmission system is used in this companion standard between the protection equipment and the control system. The data circuit terminating equipment (DCE) of the protection equipment may either be realized as a fiber optic transmission system or as a copper-wire-based transmission system. If a fiber optic transmission system is used, the compatible interface is a fiber optic connector at the protection equipment. Separate optical fibers are used in the monitor direction and in the control direction. The DCE may be mechanically and/or electrically integrated into the data terminal equipment (DTE). As an alternative to the fiber optic transmission described above a copper-wire based transmission system may be used between the protection equipment and the control system. This transmission system shall comply with the EIA RS-485 standard. Due to the characteristic of the EIA RS-485 standard a maximum number of 32 units of load can be connected to one physical line.

2.1.2 Address Setting

IEC protocol is a multipoint protocol. This means that one Master can communicate with multiple slaves on the same communication line. Due to this a given slave must have a unique ID with which to address it – a Relay address. A slave's device address MUST be unique on a given communication network – duplicate addresses lead to bus collision. Relay addresses must lie in the range 1 to 254. Address 255 is reserved as a global broadcast address.

2.1.3 Supported Application Functions

2.1.3.1 Reset Command

A reset of the communication function is effected by means of a reset command from the control system. This is generally transmitted by the control system when:

- the control system is initialized;
- the protection equipment does not respond during a certain t_{wz} (cycle repeat time) period.

This reset command does not affect the protection function, but only resets the communication part of the protection equipment. The reset command can be transmitted as:

- reset frame count bit (FCB), or
- reset communication unit (CU).

In the case of reset FCB, the internal FCB bit in the protection equipment is set to '0'. Messages in the transmission buffer are not deleted.

In the case of reset CU, the messages in the transmission buffer are additionally deleted. For more information refer section 7.2.1 and section 7.2.2.

2.1.3.2 General Interrogation

General Interrogation is used to retrieve the state of certain events at the time of the interrogation. For more information refer section 7.2.3.

2.1.3.3 Request Link Status

This command is used to request the status of data link. A slave will respond to this request with 'Link Status Acknowledge' message. For more information refer section 7.2.4.

2.1.3.4 Time Synchronization

Usually, the time synchronize command is used to synchronize time of all secondary devices on a network. This command is also used to set the time of an individual secondary station. For more information refer section 7.2.5.

2.1.3.5 General Command

Usually this command is used to send 'general command' to protection equipment. For more information refer section 7.2.6.

2.1.4 Supported Application Service Data Unit (ASDU) Types

- Time-tagged message
- Time-tagged message with relative time
- Measurands I
- Time-tagged measurands with relative time
- Identification message
- Time synchronization
- General interrogation
- Measurands II
- General command

2.1.5 Measurands in Monitor Direction*

I – Current
V – Voltage
P – 3-Phase Active power
Q – 3-Phase Reactive power
f – Frequency
I_N – Phase Current
V_{EN} – Neutral voltage
I_{L1} – Current L_1
I_{L2} – Current L_2
I_{L3} – Current L_3
V_{L1} – Voltage L_{1-E}
V_{L2} – Voltage L_{2-E}
V_{L3} – Voltage L_{3-E}

* Monitor Direction – Direction of transmission from the protection equipment to control system.

3.0 Components of the IEC Master Source Code Library

The IEC Master SCL is implemented using the following files:

3.0.1 IECDefs.h – This file contains the type definitions of some basic data types and definitions of some symbolic literals. The end user should not make modifications to this file.

3.0.2 IECDriver.c – This file contains the functions used to initialize various data structures like the 6 queues (Application Transmit Queue, Application Receive queue, Database queue, User Request queue, Link receive queue, Link Transmit queue). This file also contains functions to initialize different networks and sessions. The end user should not make modifications to this file.

3.0.3 IECDriver.h – This file contains the declarations for the various functions used in the IECDriver.c file. The end user should not make modifications to this file.

3.0.4 IECPhysical.c – This file contains the physical layer functions like physical receive and physical send functions. The end user should not make modifications to this file.

3.0.5 IECLink.c – This file contains the link layer functions like link receive, link transmit, poll transmit and status update transmit functions. The end user should not make modifications to this file.

3.0.6 IECLink.h – This file contains the physical & link layer functions and structure declarations and definitions of symbolic literals. The end user should not make modifications to this file.

3.0.7 IECAppl.c – This file contains the Application layer functions like Application receive queue read, Application Transmit queue read, functions that handle the user requests at the application layer level, functions that handle the replies from the IEC slave, interprets the data and then sends to database layer. The end user should not make modifications to this file.

3.0.8 IECAppl.h – This file contains the declarations of all the different functions used in Application Layer i.e. IECAppl.c. All the data structures related to Application Layer are declared in this file. This file also contains the different codes for Frame types, function types, cause of transmission in both control direction and monitor direction. The end user should not make modifications to this file.

3.0.9 IECData.c – This file contains the IEC 60870-5-103 data storage functions. The end user should not make modifications to this file.

3.0.10 IECData.h – This file contains the IEC 60870-5-103 data storage functions and structure declarations. The end user should not make modifications to this file.

3.0.11 IECUI.c – This file contains the User Interface functions. The end user should not make modifications to this file.

3.0.12 IECui.h – **This** file contains Declaration for User Interface functions, predefined macros to access structure members and predefined macros for Function Indices. The end user should not make modifications to this file.

3.0.13 IECTime.c – This file contains the IEC 60870-5-103 time related functions. The end user should not make modifications to this file.

3.0.14 IECTime.h – This file contains the declarations for IEC 60870-5-103 time related structures and functions. The end user should not make modifications to this file.

3.0.15 IECQue.c – This file contains the Generic Queue functions. The end user should not make modifications to this file.

3.0.16 IECque.h – This file contains the declarations for Generic queue related structures and functions. The end user should not make modifications to this file.

3.0.17 IECTarg.h – This file contains the target platform specific macros. A sample declaration of all Macros is already made to ease the process for the end user. The user may change the declarations as required.

3.0.18 IECSession.h – This file contains the session related & network related structure declarations. The end user should not make modifications to this file.

3.0.19 IECErr.h – This file contains the error definitions of various layers. The end user should not make modifications to this file.

3.0.20 IECUser.c – This file contains the IEC 60870-5-103 User level functions to send application layer request. The end user should not make modifications to this file.

3.0.21 IECUser.h – This file contains the IEC 60870-5-103 User level functions and User structure declarations. The end user should not make modifications to this file.

3.0.22 IECLinkUser.c – This file contains the Link Layer user defined functions which the user can use for defining his functions. A sample definition of all functions is already made to ease the process for the end user. The user may change the definitions as required.

3.0.23 IECLinkUser.h –This file contains the MACRO calls which have to be filled by the user with his function. A sample declaration of all Macros is already made to ease the process for the end user. The user may change the declarations as required.

3.0.24 IECApplUser.c – This file contains the Application layer user defined functions which the user can use for defining his functions. A sample definition of all functions is already made to ease the process for the end user. The user may change the definitions as required.

3.0.25 IECAppluser.h – This file contains the MACRO calls which have to be filled by the user with his function. A sample declaration of all Macros is already made to ease the process for the end user. The user may change the declarations as required.

3.0.26 IECDataUser.h – This file contains the MACROS calls which has to be filled by the user with his database storage function for the given frame type. A sample declaration of all Macros is already made to ease the process for the end user. The user may change the declarations as required.

4.0 Porting the Source Code Library

Since the SCL has been written using ANSI 'C' it is portable between operating systems and also between hardware platforms. The implementation is independent of physical transmission layer giving the user full freedom to choose the media. Based on the physical layer chosen, the physical layer Macros must be implemented by the user. Similarly the user interface of the driver (i.e. the manner in which the SCL interacts with the user application and database) is also left open – the user can implement it in any manner desired by him. So the porting of the SCL involves defining the Macros (i.e. mapping the Macros to actual function names) and implementing the functions.
The porting of the SCL to a native platform is done in five steps as below:

- 4.0.1 Define and implement the Database Storage Interface Macros and Functions
  - *4.0.1.1* IECDATAUSER_INIT_DATABASE
  - *4.0.1.2* IECDATAUSER_DEINIT_DATABASE
  - *4.0.1.3* IECDATAUSER_STORE_FT1_DATA
  - *4.0.1.4* IECDATAUSER_STORE_FT2_DATA
  - *4.0.1.5* IECDATAUSER_STORE_FT3_DATA
  - *4.0.1.6* IECDATAUSER_STORE_FT4_DATA
  - *4.0.1.7* IECDATAUSER_STORE_FT5_DATA
  - *4.0.1.8* IECDATAUSER_STORE_FT9_DATA
- 4.0.2 Define and implement the Application Layer Interface Macros and Functions
  - *4.0.2.1* IECAPPLUSER_GETASDUADDR
  - *4.0.2.2* IECAPPLUSER_GETSYSTEMTIME
  - *4.0.2.3* IECAPPLUSER_GETAPPLRETRYCOUNT
  - *4.0.2.4* IECAPPLUSER_GETAPPLTIMEOUT
- 4.0.3 Define and implement the Link Layer Interface Macros and Functions
  - *4.0.3.1* IECLINKUSER_GETTOTALNOOFNETWORKS
  - *4.0.3.2* IECLINKUSER_GETBYTESFROMCOMMNPATH
  - *4.0.3.3* IECLINKUSER_READFROMCOMMNPATH
  - *4.0.3.4* IECLINKUSER_GETSESSIONNO
  - *4.0.3.5* IECLINKUSER_GETLINKRETRYCOUNT
  - *4.0.3.6* IECLINKUSER_WRITETOCOMMNPATH
  - *4.0.3.7* IECLINKUSER_GETSLAVEADDRESS
  - *4.0.3.8* IECLINKUSER_GETNETWORKNO
  - *4.0.3.9* IECLINKUSER_INITCOMMNPATH
  - *4.0.3.10* IECLINKUSER_CLOSECOMMNPATH
  - *4.0.3.11* IECLINKUSER_GETTOTALNOOFSESSIONS
  - *4.0.3.12* IECLINKUSER_GETLINKTIMEOUT
  - *4.0.3.13* IECLINKUSER_GETSTATUSUPDATEINTERVAL
  - *4.0.3.14* IECLINKUSER_GETPOLLINGTIMEINTERVAL
  - *4.0.3.15* IECLINKUSER_GETNOOFSLAVESINEACHNW
  - *4.0.3.16* IECLINKUSER_GETSLVSESSIONNO
  - *4.0.3.17* IECLINKUSER_GETSLAVENO
  - *4.0.3.18* IECLINKUSER_GETNOOFCLASSPOLLINGREQUESTS
  - *4.0.3.19* IECLINKUSER_LINKCHECKSUMERROR
  - *4.0.3.20* IECLINKUSER_LINKCHANGESTATUS
- 4.0.4 Define the Stack Control Macros
  - *4.0.4.1* INTEL_Motorola
  - *4.0.4.2* IECTARG_UIENABLED
  - *4.0.4.3* IECTARG_DEBUGENABLED
- 4.0.5 Define the Other Macros
  - *4.0.5.1* IECTARG_GETSYSTIMEMILLISECS
  - *4.0.5.2* IECTARG_MEMORYFULL

4.1 Database Storage Interface Macros and Functions

These Macro calls have to be filled by the user with his database storage function for the given frame type. The following Macros are available:

4.1.1 IECDATAUSER_INIT_DATABASE()

This Macro is called by the stack to create and initialise an IEC 60870-5-103 database and returns the status of the Database creation and initialisation.

Expected return value: unsigned char – TRUE (1) if creation and initialisation of Database is successful,
– FALSE (0) otherwise

Parameters: None

A sample implementation is as below:

```
// IecDataUser.h file
#define IECDATAUSER_INIT_DATABASE() InitDatabase()

// Simple demo of IECDATAUSER_INIT_DATABASE Macro implementation. It simply returns 1 (TRUE). End
// user may modify this function depending on his requirements.
  Initdatabase() { return(1);}
```

4.1.2 IECDATAUSER_DEINIT_DATABASE()

This Macro is called by the stack to free or de-initialise an IEC 60870-5-103 database.

Expected Return Value: None

Parameters: None

A sample implementation is as below:

```
// IecDataUser.h file
#define IECDATAUSER_DEINIT_DATABASE() DeInitDatabase()

// Simple demo of IECDATAUSER_DEINIT_DATABASE Macro implementation. End user may modify this
// function depending on his requirements.
  DeInitDatabase ()
  { return; }
```

4.1.3 IECDATAUSER_STORE_FT1_DATA(ApplInfo, Data)

This Macro is called by the stack to Store Time tagged message into Database. Time tagged frame used to return General Interrogation and General Command Responses. They are extracted as the response to a request for class 1 data . The type of information being returned can be determined by checking the cause of transmission (COT) octet, the supplementary information octet is dependent on the COT octet.

Description	COT	Supplementary Information
General Interrogation	9	GI Scan Number from GI Initialization Message
Command Acknowledgement Positive	20	Return Information Identifier from Command Message
Command Acknowledgement Negative	21	Return Information Identifier from Command Message

Expected Return Value: unsigned char – TRUE if no errors occurred during storing,
– FALSE otherwise

Parameters:

*4.1.3.1 IECAPPL_APPLN_RECEIVE_DATA *ApplInfo – A structure variable containing the data which is received by Application Layer.*

4.1.3.2 *IECDATA_FT1DATA *Data – A structure variable containing the data to be written to user database. The members of this structure are –*

- *unsigned char DpiValue – Double Point Information*
- *unsigned char Sin – Supplementary Information*
- *unsigned char InformationElements – Number of Information Elements*
- *IECTIME_FOUR_OCTET_TIME *Time – IEC Time*

A sample implementation is as below:

```
// IecDataUser.h file
#define IECDATAUSER_STORE_FT1_DATA(ApplInfo, Data) StoreFT1Data(ApplInfo, Data)

// Simple demo of IECDATAUSER_STORE_FT1_DATA Macro implementation. For demonstration purposes,
// Data is assumed to be made available in STORETYPE1FRAME object. The StoreFT1Data function simply
// copies the data from the buffer to this global object.
  typedef struct {
      IECDEFS_UCHAR Sequence; // Sequence
      IECDEFS_UCHAR CauseOfTrans; //Cause of Transmission
      IECDEFS_UCHAR Fun_Type; //Function Type
      IECDEFS_UCHAR InfoNo; //Information Number
      IECDEFS_UCHAR Asdu_Addr; //ASDU Address
      IECDEFS_UCHAR SupplementaryInfo; //Supplementary Information
      unsigned char  DpiValue;
      IECDEFS_UCHAR Days;
      IECDEFS_UCHAR Months;
      IECDEFS_UCHAR Years;
      IECDEFS_UCHAR Hours;
      IECDEFS_UCHAR Minutes;
      IECDEFS_SHORT MSecs;
}STORETYPE1FRAME;

STORETYPE1FRAME RefData;
unsigned char StoreFT1Data(IECAPPL_APPLN_RECEIVE_DATA *ApplInfo, IECDATA_FT1DATA *Data)
{    RefData.Sequence = ApplInfo->DataUnit_Id.VarStr_Id.Sequence;
      RefData.CauseOfTrans = ApplInfo->DataUnit_Id.CauseOfTrans;
      RefData.Fun_Type = ApplInfo->Object_Id.Fun_Type;
      RefData.InfoNo = ApplInfo->Object_Id.InfoNo;
      RefData.Asdu_Addr = ApplInfo->DataUnit_Id.Asdu_Addr;
      RefData.SupplementaryInfo = ApplInfo->SupplementaryInfo;
      RefData.DpiValue = Data->DpiValue; // Double Point Information
      // FOUR OCTET IEC 60870-5-103 Time in Hours:Minutes:Milliseconds
      RefData.Hours = Data->Time->Hour.Hours;
      RefData.Minutes = Data->Time->Min.Minutes;
      RefData.MSecs = Data->Time->MSecs;
      return(1);
}
```

4.1.4 IECDATAUSER_STORE_FT2_DATA(ApplInfo, Data)

This Macro is called by the stack to store Relative Time tagged message into database. Time tagged frame with relative time, extracted as the response to a request for class 1 data. The relative time is relative to the start/pickup of the protection equipment and is given in milliseconds. The fault number is used to tag events so it can be seen which events are related to which faults. The fault number will be incremented each time a new fault occurs. Relative time and the Fault Number are not relevant for general interrogation. Any general interrogation responses which are returned in this type of frame will have these values set to zero. No Command responses will be returned in this type of frame.

Expected Return Value: unsigned char – TRUE (1) if no errors occurred during storing.
– FALSE (0) otherwise

Parameters:

*4.1.4.1 IECAPPL_APPLN_RECEIVE_DATA *ApplInfo - A structure variable containing the data which is received by Application Layer.*

*4.1.4.2 IECDATA_FT2DATA *Data – A structure variable containing the data to be written to user database. The members of this structure are –*

- *unsigned char DpiValue – Double Point Information*
- *unsigned short RelTime – Relative Time elapsed*
- *unsigned short FaultNumber– Fault Number*
- *unsigned char Sin – Supplementary Information*

- *unsigned char InformationElements – Number of Information Elements*
- *IECTIME_FOUR_OCTET_TIME *Time – IEC Time*

A sample implementation is as below:

```
// IecDataUser.h file
#define IECDATAUSER_STORE_FT2_DATA(ApplInfo,Data)  StoreFT2Data(ApplInfo, Data)

// Simple demo of IECDATAUSER_STORE_FT2_DATA Macro implementation. For demonstration purposes,
// Data is assumed to be made available in STORETYPE2FRAME object. The StoreFT2Data function simply
// copies the data from the buffer to this global object.
  typedef struct {
     unsigned char Sequence;         // Sequence
     unsigned char CauseOfTrans;  // Cause of transmission
     unsigned char Fun_Type;        // Function Type
     unsigned char InfoNo;           // Informaton Number
     unsigned char Asdu_Addr;      // ASDU Address
     unsigned char SupplementaryInfo; // Supplementary Information
     unsigned char  DpiValue;   // Double point information
     unsigned short RelTime;    // Relative time
     unsigned short FaultNo;    // Fault Number
     // FOUR OCTET IEC 60870-5-103 Time in Hours:Minutes:Milliseconds
     unsigned char Hours;
     unsigned char Minutes;
     short MSecs;
  }STORETYPE2FRAME;

  STORETYPE2FRAME RefData;
  unsigned char StoreFT2Data(IECAPPL_APPLN_RECEIVE_DATA *ApplInfo, IECDATA_FT2DATA *Data)
  {    RefData.Sequence = ApplInfo->DataUnit_Id.VarStr_Id.Sequence;
       RefData.CauseOfTrans = ApplInfo->DataUnit_Id.CauseOfTrans;
       RefData.Fun_Type = ApplInfo->Object_Id.Fun_Type;
       RefData.InfoNo = ApplInfo->Object_Id.InfoNo;
       RefData.Asdu_Addr = ApplInfo->DataUnit_Id.Asdu_Addr;
       RefData.SupplementaryInfo = ApplInfo->SupplementaryInfo;
       RefData.DpiValue = Data->DpiValue;
       RefData.RelTime = Data->RelTime;
       RefData.FaultNo = Data->FaultNumber;
       RefData.Hours = Data->Time->Hour.Hours;
       RefData.Minutes = Data->Time->Min.Minutes;
       RefData.MSecs = Data->Time->MSecs;
       return(1);
  }
```

4.1.5 IECDATAUSER_STORE_FT3_DATA (ApplInfo, Data)

This Macro is called by the stack to Store Measurand I Value into database. Type 3 data frames provide groups of measured values. The group contained is determined by the Information Elements octet which states how many measurands are present in the frame. This information and a description of the measurands is shown below.

Information Number	Description
144	Measurand I
145	Measurands I,V
146	Measurands I, V, P, Q
147	Measurands I_N, V_{EN}

Whenever possible, the relays background process prepares a Type 3 data frame containing a normalized measured value(s). If available, it is extracted from the relay by means of the Request Data Class 2.

Expected Return Value: unsigned char – TRUE (1) if no errors occurred during storing
– FALSE (0) otherwise

Parameters:

*4.1.5.1 IECAPPL_APPLN_RECEIVE_DATA *ApplInfo – A structure variable containing the data which is received by Application Layer.*

*4.1.5.2 IECDATA_FT3DATA *Data – A structure variable containing the data to be written to user database. The members of this structure are –*

- *IECDATA_MEASURAND *Value – Measurand Value*
  IECDATA_MEASURAND itself is structure which contains 13 bit measurand Value, Overflow bit and Error bit.
- *unsigned char InformationElements – Number of Information Elements*

The table below shows the octets when the Information Number is 145.

Address Location	Description
Value	Measurand I
Value + 1	Measurand V

A sample implementation is as below:

```
// IecDataUser.h file
#define IECDATAUSER_STORE_FT3_DATA(ApplInfo,Data) StoreFT3Data(ApplInfo, Data)

// Simple demo of IECDATAUSER_STORE_FT3_DATA Macro implementation. For demonstration purposes,
// Data is assumed to be made available in STORETYPE3FRAME object. The StoreFT3Data function simply
// copies the data from the buffer to this global object.

  // Structure to hold Measurand value
  typedef struct {
     unsigned short Overflow;
     unsigned short Error;
     unsigned short Value;
  } MEASURAND_VALUES;
  typedef struct {
       unsigned char Sequence;      //Sequence
       unsigned char CauseOfTrans;  //Cause of transmission
       unsigned char Fun_Type;      //Function Type
       unsigned char InfoNo;        //Informaton Number
       unsigned char Asdu_Addr;     //ASDU Address
       MEASURAND_VALUES *Value;
  }STORETYPE3FRAME;

  STORETYPE3FRAME RefData;
  unsigned char StoreFT3Data(IECAPPL_APPLN_RECEIVE_DATA *ApplInfo, IECDATA_FT3DATA *Data)
  {    int InfoElement;
       RefData.Sequence = ApplInfo->DataUnit_Id.VarStr_Id.Sequence;
       RefData.CauseOfTrans = ApplInfo->DataUnit_Id.CauseOfTrans;
       RefData.Fun_Type = ApplInfo->Object_Id.Fun_Type;
       RefData.InfoNo = ApplInfo->Object_Id.InfoNo;
       RefData.Asdu_Addr = ApplInfo->DataUnit_Id.Asdu_Addr;
       RefData.Value = (MEASURAND_VALUES*)calloc(Data->InformationElements,
       sizeof(MEASURAND_VALUES));
     if(RefData.Value == NULL){ printf("Not Enough Memory"); return(0);}
     for(InfoElement = 0; InfoElement < Data->InformationElements; InfoElement++)
     {     RefData.Value[InfoElement].Overflow = Data->Value[InfoElement].Overflow;
           RefData.Value[InfoElement].Error = Data->Value[InfoElement].Error;
           RefData.Value[InfoElement].Value = Data->Value[InfoElement].Value;
     }
     return(1);
  }
```

4.1.6 IECDATAUSER_STORE_FT4_DATA (ApplInfo, Data)

This Macro is called by the stack to Store time tagged Measurand Value into Database. The short-circuit location represents the location as fault reactance related to primary values. It is given in ohms. The relative time is reset at the beginning of a short circuit. It indicates the time in milliseconds from the start/pick-up of the protection equipment up to the present time. A short circuit, might cause several faults with trip and auto-reclosing, each fault being identified by an increased fault number.

Expected Return Value: unsigned char – TRUE (1) if no errors occurred during storing – FALSE (0) otherwise

Parameters:

*4.1.6.1 IECAPPL_APPLN_RECEIVE_DATA *ApplInfo – A structure variable containing the data which is received by Application Layer.*

*4.1.6.2 IECDATA_FT4DATA *Data – A structure variable containing the data to be written to user database. The members of this structure are –*

- *IECDATA_SCL Scl – Short Circuit Location*

IECDATA_SCL itself is structure which contains 1 bit sign, 23 bit mantissa, 8 bit exponent.

- *unsigned short RelTime – Relative Time elapsed*
- *unsigned short FaultNumber– Fault Number*
- *IECTIME_FOUR_OCTET_TIME *Time – IEC Time*
- *unsigned char InformationElements – Number of Information Elements*

A sample implementation is as below:

```
// IecDataUser.h file
#define IECDATAUSER_STORE_FT4_DATA(ApplInfo,Data) StoreFT4Data(ApplInfo, Data)

// Simple demo of IECDATAUSER_STORE_FT4_DATA Macro implementation. For demonstration purposes,
// Data is assumed to be made available in STORETYPE4FRAME object. The StoreFT4Data function simply
// copies the data from the buffer to this global object.
typedef struct {
   unsigned long Exponent:8;  //Exponent
   unsigned long Mantissa:23;  //Mantissa
   unsigned long Sign:1; // Sign
   unsigned short RelTime; // Relative Time elapsed
   unsigned short FaultNumber; // Fault Number
   unsigned char InformationElements;  //Information Elements
   unsigned char Sequence;      //Sequence
   unsigned char CauseOfTrans;  //Cause of transmission
   unsigned char Fun_Type;      //Function Type
   unsigned char InfoNo;        //Informaton Number
   unsigned char Asdu_Addr;     //ASDU Address
  // FOUR OCTET IEC 60870-5-103 Time in Hours:Minutes:Milliseconds
   unsigned char Hours;
   unsigned char Minutes;
   unsigned short MSecs;
}STORETYPE4FRAME;

STORETYPE4FRAME RefData;
unsigned char StoreFT4Data(IECAPPL_APPLN_RECEIVE_DATA *ApplInfo, IECDATA_FT4DATA *Data)
{    RefData.Exponent = Data->Scl.Exponent;
     RefData.Mantissa = Data->Scl.Mantissa;
     RefData.Sign = Data->Scl.Sign;
     RefData.FaultNumber = Data->FaultNumber;
     RefData.InformationElements = Data->InformationElements;
     RefData.Hours = Data->Time->Hour.Hours;
     RefData.Minutes = Data->Time->Min.Minutes;
     RefData.MSecs = Data->Time->MSecs;
     RefData.RelTime = Data->RelTime;
     RefData.Sequence = ApplInfo->DataUnit_Id.VarStr_Id.Sequence;
     RefData.CauseOfTrans = ApplInfo->DataUnit_Id.CauseOfTrans;
```

```
    RefData.Fun_Type = ApplInfo->Object_Id.Fun_Type;
    RefData.InfoNo = ApplInfo->Object_Id.InfoNo;
    RefData.Asdu_Addr = ApplInfo->DataUnit_Id.Asdu_Addr;
    return(1); /* Return True */
}
```

4.1.7 IECDATAUSER_STORE_FT5_DATA(ApplInfo, Data)

This Macro is called by the stack to Store Identification message into database. Type 5 data frames provide identification messages. These messages are returned in response to the events Reset Communications Unit, Reset Frame Count Bit and a Start Restart of the relay. They are extracted as the response to a request for class 1 data.

Table Showing Cause of Transmission and Information Number values for the events described above.

Information Number	COT	Event
2	3	Reset Frame Count Bit
3	4	Reset Communication Unit
4	5	Start / Restart

After the initial reset command to the relay it will prepare two events in type 5 frames, which will be the initial class 1 responses. The first will have COT = 4, INF = 3 if the command was a Reset Communications Unit, otherwise COT = 3, INF = 2 if a Reset Frame Check Bit. The second frame will always have COT = 5, INF = 4, signifying a restart.

When the user or control system requests Reset Communications Unit or Reset Frame Count a single type 5 response frame will be generated with COT and INF as described above for the respective commands. Again these have to be extracted with a request for class 1 data.

Expected Return Value: unsigned char – TRUE (1) if no errors occurred during storing
– FALSE (0) otherwise

Parameters:

*4.1.7.1 IECAPPL_APPLN_RECEIVE_DATA *ApplInfo – A structure variable containing the data which is received by Application Layer.*

*4.1.7.2 IECDATA_FT5DATA *Data – A structure variable containing the data to be written to user database. The members of this structure are –*

- *unsigned char CompatibilityLevel – Compatibility Level*
- *unsigned char Name[9] – Manufacturer Name*
- *unsigned char MIS_Identification[5] – Software Identification*

A sample implementation is as below:

```
// IecDataUser.h file
#define IECDATAUSER_STORE_FT5_DATA(ApplInfo,Data) StoreFT5Data(ApplInfo, Data)

// Simple demo of IECDATAUSER_STORE_FT5_DATA Macro implementation. For demonstration purposes,
// Data is assumed to be made available in STORETYPE5FRAME object. The StoreFT5Data function simply
// copies the data from the buffer to this global object.

typedef struct {
    unsigned char CompLevel; //Compatibility Level
    unsigned char Id[5]; // Software Identification
    unsigned char Name[9]; // Manufacturer Name
} STORETYPE5FRAME;

STORETYPE5FRAME RefData;
unsigned char StoreFT5Data(IECAPPL_APPLN_RECEIVE_DATA *ApplInfo, IECDATA_FT5DATA *Data)
{ RefData.CompLevel = Data->CompatibilityLevel;
  strcpy(RefData.Id, Data->MIS_Identification);
  strcpy(RefData.Name, Data->Name);
  return(1); /* Return True */
}
```

4.1.8 IECDATAUSER_STORE_FT9_DATA (ApplInfo,Data)

This Macro is called by the stack to Store Measurand II Value into database. Type 9 data frames provide measured values. The information number is 148 and represents nine measurands, $I_{L1,2,3}$, $V_{L1,2,3}$, P, Q, f where P and Q are the 3 phase active and reactive powers and f is the frequency. Type 9 data frames are very similar to type 3 except that not all the measurands need be sent. The group contained is determined by the Information Elements octet which states how many measurands are present in the frame and always starting at I_{L1}, the measurands can be truncated, for example, 4 would mean we are sending $I_{L1,2,3}$, V_{L1}. Type 9 frames are extracted from the relay by means of the Request Data Class 2.

Expected Return Value: unsigned char – TRUE if no errors occurred during storing,
– FALSE otherwise

Parameters:

*4.1.8.1 IECAPPL_APPLN_RECEIVE_DATA *ApplInfo – A structure variable containing the data which is received by Application Layer.*

*4.1.8.2 IECDATA_FT9DATA *Data – A structure variable containing the data to be written to user database. The members of this structure are –*

- *IECDATA_MEASURAND *Value – Measurand Value*
- *IECDATA_MEASURAND itself is structure which contains 13 bit measurand Value, Overflow bit and Error bit.*
- *unsigned char InformationElements – Number of Information Elements*

A sample implementation is as below:

```
// IecDataUser.h file
#define IECDATAUSER_STORE_FT9_DATA(ApplInfo, Data) StoreFT9Data(ApplInfo, Data)

// Simple demo of IECDATAUSER_STORE_FT9_DATA Macro implementation. For demonstration purposes,
// Data is assumed to be made available in STORETYPE9FRAME object. The StoreFT9Data function simply
// copies the data from the buffer to this global object.

// Structure to hold measurand value
  typedef struct {
     unsigned short Overflow;
     unsigned short Error;
     unsigned short Value;
  } MEASURAND_VALUES;

  typedef struct {
     unsigned char Sequence;      //Sequence
     unsigned char CauseOfTrans;  //Cause of transmission
     unsigned char Fun_Type;      //Function Type
     unsigned char InfoNo;        //Informaton Number
     unsigned char Asdu_Addr;     //ASDU Address
     MEASURAND_VALUES *Value;
  }STORETYPE9FRAME;

  STORETYPE9FRAME RefData;
  unsigned char StoreFT9Data(IECAPPL_APPLN_RECEIVE_DATA *ApplInfo, IECDATA_FT3DATA *Data)
  {
     int InfoElement;
     RefData.Sequence = ApplInfo->DataUnit_Id.VarStr_Id.Sequence;
     RefData.CauseOfTrans = ApplInfo->DataUnit_Id.CauseOfTrans;
     RefData.Fun_Type = ApplInfo->Object_Id.Fun_Type;
     RefData.InfoNo = ApplInfo->Object_Id.InfoNo;
     RefData.Asdu_Addr = ApplInfo->DataUnit_Id.Asdu_Addr;
     RefData.Value = (MEASURAND_VALUES*)calloc(Data->InformationElements,
  sizeof(MEASURAND_VALUES));
     if(RefData.Value == NULL)
     {          printf("Not Enough Memory");
                return(0);
```

```
        }
        for(InfoElement = 0; InfoElement < Data->InformationElements; InfoElement++)
        {       RefData.Value[InfoElement].Overflow = Data->Value[InfoElement].Overflow;
                RefData.Value[InfoElement].Error = Data->Value[InfoElement].Error;
                RefData.Value[InfoElement].Value = Data->Value[InfoElement].Value;
        }
        return(1); }
```

4.2 Application Layer Interface Macros and Functions

The following macros are used in application layer:

4.2.1 IECAPPLUSER_GETASDUADDR(SessionNo)

This Macro is called by the stack to get the corresponding ASDU Address for the Session Number.

Expected Return Value: unsigned char – The ASDU address for the session.

Parameters:

4.2.1.1 unsigned short SessionNo – Session Number

A sample implementation is as below:

```
// IecApplUser.h file
#define IECAPPLUSER_GETASDUADDR(SessionNo) GetAsduAdd(SessionNo)

// IecApplUser.c file – simple demo of IECAPPLUSER_GETASDUADDR Macro implementation.
IECDEFS_UCHAR GetAsduAdd(IECDEFS_USHORT RequestSessionNum) {
   if(RequestSessionNum==0) return(1);
   if(RequestSessionNum==1) return(2); }
```

4.2.2 IECAPPLUSER_GETSYSTEMTIME(SystemTime)

This Macro is called by the stack to get the system time and fills the individual variables in structure IECTIME_SEVEN_OCTET_TIME. The time variable is a pointer and the modification is reflected in the main macro.

Expected Return Value: None

Parameters:

4.2.2.1 IECTIME_SEVEN_OCTET_TIME SystemTime – A pre allocated buffer into which the Seven OCTET IEC 60870-5-103 Time must be stored.*

A sample implementation is as below:

```
// IECApplUser.h file
#define IECAPPLUSER_GETSYSTEMTIME(SystemTime) GetTime(SystemTime)

// IECApplUser.c file – simple demo of IECAPPLUSER_ GETSYSTEMTIME Macro implementation.
void  GetTime(IECTIME_SEVEN_OCTET_TIME* DateTime) {
        u_int32  Format=0,Time_1,Date_1,Ticks_1,Error;
        u_int16  Day_1;
        /* Gets Os9 system time */
        Error=_os9_getime( Format, &Time_1, &Date_1, &Day_1, &Ticks_1);
        DateTime->MSecs = ( Time_1 & 0xff ) * 1000;
        DateTime->Min = ( Time_1 & 0xff00) >> 8;
        DateTime->Hour = ( Time_1 & 0xff0000) >> 16;
        DateTime->Days = Date_1 & 0xff;
        DateTime->month = ( Date_1 & 0xff00 ) >> 8;
        DateTime->Years = ( Date_1 & 0xffff0000 ) >> 16;
}
```

4.2.3 IECAPPLUSER_GETAPPLRETRYCOUNT(NetworkNo)

The stack calls this macro to get the retry count for the Application layer for the particular Network.

Expected Return Value: unsigned short – Retry count

Parameters:

4.2.3.1 unsigned short NetworkNo– Network number

A sample implementation is as below:

```
// IECApplUser.h file
#define IECAPPLUSER_GETAPPLRETRYCOUNT(NetworkNo) 1        // Retry Count is 1
```

4.2.4 IECAPPLUSER_GETAPPLTIMEOUT(NetworkNo)

The stack calls this macro to get the Time Out for the Application layer for the particular network in terms of milliseconds.

Expected Return Value: unsigned long – timeout period in milliseconds

Parameters:

4.2.4.1 unsigned char NetworkNo – Network number.

A sample implementation is as below:

```
// IECApplUser.h file
#define IECAPPLUSER_GETAPPLTIMEOUT(NetworkNo) 10000    //Timeout = 10000 MilliSec.
```

4.3 Physical/Link Layer Interface Macros and Functions

The following macros are used in Link layer:

4.3.1 IECLINKUSER_GETTOTALNOOFNETWORKS()

The stack calls this macro to get the total no of networks present.

Expected Return Value: unsigned char – Total No of networks

Parameters: None

A sample implementation is as below:

```
// IECLinkUser.h file
#define IECLINKUSER_GETTOTALNOOFNETWORKS()      1  // Total Number of Networks = 1
```

4.3.2 IECLINKUSER_GETBYTESFROMCOMMNPATH(NetworkNo)

The stack calls this macro to get the no of bytes available on the port to be read for the given network.

Expected Return Value: unsigned long – Number of bytes available on the port for the given network.

– 0 (Zero) if no bytes are available on the port.

Parameters:

4.3.2.1 unsigned char NetworkNo – Network number.

A typical implementation for OS-9 would be as below:

```
// IECLinkUser.h file
#define IECLINKUSER_GETBYTESFROMCOMMNPATH(NetworkNo)  GetBytesFromComPath()

// IECLinkUser.c file -Simple demo of IECLINKUSER_GETBYTESFROMCOMMNPATH Macro implementation.
IECDEFS_ULONG GetBytesFromComPath() {
  int count;
  count = _gs_rdy(g_pathid);
  if( count == -1 ) { return(0); } // no bytes in port
  else { return((IECDEFS_ULONG)count); } // Number of bytes available on port
}
```

4.3.3 IECLINKUSER_READFROMCOMMNPATH(NetworkNo, NoOfBytesToRead, ReadBuf)

The stack calls this macro to get the data (given no of bytes) from the port into a buffer for the given network

Expected Return Value: unsigned char TRUE (1) – No errors occurred during reading from port.

FALSE (0) – Otherwise

Parameters:

4.3.3.1 unsigned char NetworkNo – Network number.

4.3.3.2 unsigned long NoOfBytesToRead – Number of bytes to be read.

*4.3.3.3 unsigned char *ReadBuf – A pre allocated buffer into which the data from the port must be stored.*

A typical implementation for OS-9 would be as below:

```
// IECLinkUser.h file
#define IECLINKUSER_READFROMCOMMNPATH(NetworkNo, NoOfBytesToRead, ReadBuf)
                                                ReadFromComPath(NoOfBytesToRead, ReadBuf)

// IECLinkUser.c file - simple demo of IECLINKUSER_READFROMCOMMNPATH Macro implementation.
IECDEFS_BOOL ReadFromComPath(IECDEFS_ULONG NoOfBytesToRead, IECDEFS_UCHAR *ReadBuf) {
  if( (errno=read(g_pathid,ReadBuf,(int)NoOfBytesToRead)) == -1 ) {
          printf("Error No=%d\n",errno);
          printf("can't read from path\n");
          fflush(stdout);
          return 0;}
  return(1); }
```

4.3.4 IECLINKUSER_GETSESSIONNO(NetworkNo, StationAddr)

The stack calls this macro to Get the session number for the given network number and station address.

Expected Return Value: unsigned short – Session number

Parameters:

4.3.4.1 unsigned char NetworkNo – Network number

4.3.4.2 unsigned char StationAddr – Station Address

A sample implementation is as below:

```
// IECLinkUser.h file
#define IECLINKUSER_GETSESSIONNO(NetworkNo, StationAddr)  GetSesNo(StationAddr)

// IECLinkUser.c file – simple demo of IECLINKUSER_GETSESSIONNO Macro implementation.
IECDEFS_USHORT GetSesNo(IECDEFS_UCHAR StationAddr)
{ if(StationAddr==1) return(0);
  if(StationAddr==2) return(1);}
```

4.3.5 IECLINKUSER_GETLINKRETRYCOUNT(NetworkNo)

The stack calls this macro to get the Link Retry Count for the given network.

Expected Return Value: unsigned char – Link Retry Count

Parameters:

4.3.5.1 unsigned char NetworkNo – Network number

A sample implementation is as below:

```
// IECLinkUser.h file
#define IECLINKUSER_GETLINKRETRYCOUNT(NetworkNo)  3  // Link Retry Count = 3
```

4.3.6 IECLINKUSER_WRITETOCOMMNPATH(NetworkNo, RequestFrameBuf, NoOfBytesToWrite)

The stack calls this macro to write the data (given no of bytes) to the given port.

Expected Return Value: unsigned char TRUE (1) – if no errors occurred during writing to port.
FALSE (0) – Otherwise

Parameters:

4.3.6.1 unsigned char NetworkNo – Network number

*4.3.6.2 unsigned char *RequestFrameBuf –Buffer which contains data to write to the path.*

4.3.6.3 unsigned long NoOfBytesToWrite – Total number of bytes to write.

A sample implementation is as below:

```
// IECLinkUser.h file
#define IECLINKUSER_WRITETOCOMMNPATH(NetworkNo, RequestFrameBuf, NoOfBytesToWrite)
                                                    WriteComPath(RequestFrameBuf, NoOfBytesToWrite)

// IECLinkUser.c file – simple demo of IECLINKUSER_WRITETOCOMMNPATH  Macro implementation.
 IECDEFS_BOOL WriteComPath (IECDEFS_UCHAR *RequestFrameBuf, IECDEFS_ULONG NoOfBytesToWrite)
  {  if( (errno=write(g_pathid,RequestFrameBuf,(int)NoOfBytesToWrite)) == -1 )
    {          printf("Error No=%d\n",errno);
               printf("can't write to path\n");
               fflush(stdout);
               return 0; }
    return(1); }
```

4.3.7 IECLINKUSER_GETSLAVEADDRESS(RequestSessionNum)

The stack calls this macro to get the Station Address (Slave Address) for given session number.

Expected Return Value: unsigned char – Station Address (Slave Address)

Parameters:

4.3.7.1 unsigned short RequestSessionNum – Session Number

A sample implementation is as below:

```
// IECLinkUser.h file
#define IECLINKUSER_GETSLAVEADDRESS(RequestSessionNum)   GetSlaveAdd(RequestSessionNum)

// IECLinkUser.c file – simple demo of IECLINKUSER_GETSLAVEADDRESS Macro implementation.
```

```
IECDEFS_UCHAR GetSlaveAdd(IECDEFS_USHORT RequestSessionNum){
  if(RequestSessionNum==0) return(1);
  if(RequestSessionNum==1) return(2); }
```

4.3.8 IECLINKUSER_GETNETWORKNO(RequestSessionNum)

The stack calls this macro to get the network number for given session number.

Expected Return Value: unsigned char – Network Number

Parameters:

4.3.8.1 Unsigned short RequestSessionNum – Session Number

A sample implementation is as below:

```
// IECLinkUser.h file
#define IECLINKUSER_GETNETWORKNO(RequestSessionNum)    0 //Network Number
```

4.3.9 IECLINKUSER_INITCOMMNPATH(NetworkNo)

The stack calls this macro to Initialize the communication path for the given port or network.

Expected Return Value: unsigned char TRUE (1) – if no errors occurred during initialization of port.
FALSE (0) – Otherwise

Parameters:

4.3.9.1 unsigned char NetworkNo – Network Number

A typical implementation for OS-9 would be as below:

```
// IECLinkUser.h file
#define IECLINKUSER_INITCOMMNPATH(NetworkNo)    InitComPath()

// IECLinkUser.c file – simple demo of IECLINKUSER_INITCOMMNPATH Macro implementation.
IECDEFS_BOOL InitComPath(){
struct sgbuf path_opts;
  if( (g_pathid=open(PORTNAME, S_IREAD+S_IWRITE)) == -1) {
     printf("can't open path\n");
    fflush(stdout);
    return 0; }
  /*Get the different parameters for the path*/
  if( (_gs_opt(g_pathid,&path_opts)) == -1) {
     printf("can't get path options\n");
     fflush(stdout);
     return 0; }
  path_opts.sg_kbach = 0;   // Treat ^E(abort) character as normal character
  path_opts.sg_kbich = 0;   // Treat ^C(interrupt) character as normal character
  path_opts.sg_echo = 0;    // Don't echo character received
  path_opts.sg_rlnch = 0;   // Treat 0x04 as a normal character
  path_opts.sg_pause = 0;   // Don't pause after one page of character have been output
  path_opts.sg_alf = 0;     // No automatic line feed generation
  path_opts.sg_baud = 14;   // Set the baud rate to 9600
  path_opts.sg_xon = 0;     // Set XON=0
  path_opts.sg_xoff = 0;    // Set XOFF=0x80 for 485 mode
  path_opts.sg_parity = 0;  // Set no parity, 8bits/character, 1 stop bit
  path_opts.sg_eofch = 0;
  path_opts.sg_eorch = 0;
  path_opts.sg_psch = 0;
  //set options
  if( (_ss_opt(g_pathid,&path_opts)) == -1 ) {
     printf("can't set path options\n");
     fflush(stdout);
     return 0;}
  return(1); }
```

4.3.10 IECLINKUSER_CLOSECOMMNPATH(NetworkNo)

The stack calls this macro to Close the communication path for the given port or network.

Expected Return Value: unsigned char TRUE – if no errors occurred during close of port
FALSE – Otherwise

Parameters:

4.3.10.1 unsigned char NetworkNo – Network Number

A typical implementation for OS-9 would be as below:

```
// IECLinkUser.h file
#define IECLINKUSER_CLOSECOMMNPATH(NetworkNo) CloseComPath()

// simple demo of IECLINKUSER_CLOSECOMMNPATH Macro implementation.
CloseComPath(){
    if(close(g_pathid) == -1) {
        printf("can't close path\n");
        fflush(stdout);
        return(0);}
    return(1); }
```

4.3.11 IECLINKUSER_GETTOTALNOOFSESSIONS()

The stack calls this macro to get the total no of sessions present.

Expected Return Value: unsigned char – Total number of sessions
Parameters: None

A sample implementation is as below:

```
// IECLinkUser.h file
#define IECLINKUSER_GETTOTALNOOFSESSIONS()   1   //Total number of sessions present
```

4.3.12 IECLINKUSER_GETLINKTIMEOUT(NetworkNo)

The stack calls this macro to get the Link Timeout Interval for the given network.

Expected Return Value: unsigned long – Link Timeout Interval (in Milliseconds)
Parameters:

4.3.12.1 unsigned char NetworkNo – Network Number

A sample implementation is as below:

```
// IECLinkUser.h file
#define IECLINKUSER_GETLINKTIMEOUT(NetworkNo)  20000   // Link Timeout interval
```

4.3.13 IECLINKUSER_GETSTATUSUPDATEINTERVAL(SessionNo)

The stack calls this macro to get the Session Status Update Time Interval for the given session. This is the time interval at which an RCU command will be sent internally to an 'offline' session. If this feature has to be disabled for a session, then configure the time interval value as '-1' for that session.

Expected Return Value: unsigned long – Status Update TimeInterval (In Milliseconds)
Parameters:

4.3.13.1 unsigned short SessionNo – Session Number

A sample implementation is as below:

```
// IECLinkUser.h file
#define IECLINKUSER_GETSTATUSUPDATEINTERVAL(SessionNo) GetStatUpdt(SessionNo)

// IECLinkUser.c file – simple demo of IECLINKUSER_GETSTATUSUPDATEINTERVAL Macro implementation.
IECDEFS_ULONG GetStatUpdt(IECDEFS_USHORT SessionNo) {
    if(SessionNo==0) return(300000);
    if(SessionNo==1) return(300000); }
```

4.3.14 IECLINKUSER_GETPOLLINGTIMEINTERVAL(SessionNo)

The stack calls this macro to get the Polling Time Interval for the given session. The two types of data you can poll for are class 1 and class 2 .Class 2 is the measurands, all the others are class 1. Polling is performed cyclically at time intervals of for example, 50 milliseconds. This interval should be as short as is IEC870-5-103 Communication Interface practicable as you should download events in as close to real time as is possible. If polling time interval is zero, then polling is continuous.

Expected Return Value: unsigned long – Polling Time Interval (In Milliseconds)
Parameters:
4.3.14.1 Unsigned short SessionNo – Session Number
A sample implementation is as below:

```
// IECLinkUser.h file
#define IECLINKUSER_GETPOLLINGTIMEINTERVAL(SessionNo) 50 //polling time interval =50 milliseconds
```

4.3.15 IECLINKUSER_GETNOOFSLAVESINEACHNW(NetworkNo)

The stack calls this macro to get the number of slaves present for the given network.
Expected Return Value: unsigned char – Number of slaves in the given network.
Parameters:
4.3.15.1 Unsigned short NetworkNo – Network Number
A sample implementation is as below:

```
// IECLinkUser.h file
#define IECLINKUSER_GETNOOFSLAVESINEACHNW(NetworkNo) 1  // Number of slaves
```

4.3.16 IECLINKUSER_GETSLVSESSIONNO(NetworkNo, SlaveNo)

The stack calls this macro to get the session number given the network number and slave number.
Expected Return Value: unsigned short –Session Number
Parameters:
4.3.16.1 Unsigned char NetworkNo – Network Number
4.3.16.2 Unsigned char SlaveNo – Slave Number
A sample implementation is as below:

```
// IECLinkUser.h file
#define IECLINKUSER_GETSLVSESSIONNO(NetworkNo, SlaveNo) GetSlvSesNo(SlaveNo)

// IECLinkUser.c file – simple demo of IECLINKUSER_GETSLVSESSIONNO Macro implementation.
IECDEFS_USHORT GetSlvSesNo(IECDEFS_UCHAR SlaveNo) {
  if(SlaveNo==0) return(0);
   if(SlaveNo==1) return(1); }
```

4.3.17 IECLINKUSER_GETSLAVENO(RequestSessionNum)

The stack calls this macro to get the Slave number for the given session number.
Expected Return Value: unsigned char – Slave Number
Parameters:
4.3.17.1 Unsigned short RequestSessionNum – Session Number
A sample implementation is as below:

```
// IECLinkUser.h file
#define IECLINKUSER_GETSLAVENO(RequestSessionNum) GetSlvNo(RequestSessionNum)

// IECLinkUser.c file – simple demo of IECLINKUSER_GETSLAVENO Macro implementation.
IECDEFS_UCHAR GetSlvNo(IECDEFS_USHORT RequestSessionNum) {
    if(RequestSessionNum==0) return(0);
    if(RequestSessionNum==1) return(1); }
```

4.3.18 IECLINKUSER_GETNOOFCLASSPOLLINGREQUESTS(SessionNo)

The stack calls this macro to get the number of continuous class polling requests for the given session.
Expected Return Value: unsigned long – Number of continuous class polling requests
Parameters:
4.3.18.1 Unsigned short SessionNo– Session Number
A sample implementation is as below:

```
// IECLinkUser.h file
#define IECLINKUSER_GETNOOFCLASSPOLLINGREQUESTS(SessionNo) GetClassPollRqsts(SessionNo)

// IECLinkUser.c file – simple demo of IECLINKUSER_GETNOOFCLASSPOLLINGREQUESTS Macro
// implementation.
IECDEFS_ULONG GetClassPollRqsts(IECDEFS_USHORT SessionNo){
```

```
if(SessionNo==0) return(30);
if(SessionNo==1) return(10);}
```

4.3.19 IECLINKUSER_LINKCHECKSUMERROR(NetworkNo)

The stack calls this macro to notify the target system when a checksum error on a network occurs. Target system can implement a function to notify MMI about the error. Network Number specifies the network number of the error prone network.

Expected Return Value: None

Parameters:

4.3.19.1 Unsigned char NetworkNo – Network Number

A sample implementation is as below:

```
// IECLinkUser.h file
#define IECLINKUSER_LINKCHECKSUMERROR(NetworkNo)
```

4.3.20 IECLINKUSER_LINKCHANGESTATUS(NetworkNo, Status)

The stack calls this macro to notify the target system when the status of the network changes. It is called when a network goes from online(GOOD) to offline(BAD) state or from offline to online state. State of the network is passed in the Status. Target system can implement a function to notify MMI about the error.

Expected Return Value: None

Parameters:

4.3.20.1 Unsigned char NetworkNo – Network Number

4.3.20.2 Unsigned char Status – 1 (Good) or 0 (Bad)

A sample implementation is as below:

```
// IECLinkUser.h file
#define IECLINKUSER_LINKCHANGESTATUS(NetworkNo, Status)
```

4.4 Stack Control Macros

The Stack Control Macros control the behavior of the stack by changing the control flow by means of pre–processor directives. By setting appropriate values for these macros the user can control the compilation process. The following macros are provided:

4.4.1 INTEL_Motorola

The stack calls this macro to configure the target system for the Intel/Motorola (Little Endian/Big Endian) type processors.

Depending on the operating system and the hardware platform, there are two ways of storing a WORD variable in memory. In the first case called the BIG ENDIAN style the high byte of the WORD is stored first and then the low byte of the word. This style can generally be found in MOTOROLA based CPU architectures. In the other style called the LITTLE ENDIAN, the low byte of the WORD is stored first and then the high byte of the word. This style is more prominent in CPU's following the INTEL architecture.

Expected Return Value: unsigned char – 1 (TRUE) if system is Intel type.
– 0 (FALSE) if system is Motorola type.

Parameters: None

e.g.

```
// IECTarg.h file
#define INTEL_Motorola IECDEFS_TRUE
```

4.4.2 IECTARG_UIENABLED

The stack calls this macro to enable the User Interface on the target system.

Expected Return Value: unsigned char – TRUE (1) – If enable user interface
FALSE(0) – If disable user interface.

Parameters: None

e.g.

```
// IECTarg.h file
#define IECTARG_UIENABLED    1
```

4.4.3 IECTARG_DEBUGENABLED

The stack calls this macro to enable the User debugging printf's on the target system.

Expected Return Value: unsigned char – TRUE – If enable user debug
FALSE – If disable user debug.

Parameters: None

e.g.

```
// IECTarg.h file
#define IECTARG_DEBUGENABLED 0
```

4.5 Other Macros

4.5.1 IECTARG_GETSYSTIMEMILLISECS()

The stack calls this macro to get the system time in milliseconds.

Expected Return Value: unsigned long – the system time in milliseconds.(Native function call on the target OS for getting system time in milliseconds should be given)

Parameters: None

A typical implementation for OS-9 would be as below:

```
// IECTarg.h file
#define IECTARG_GETSYSTIMEMILLISECS() GetRunningMilliSecondTime()

// IECLinkUser.c file – simple demo of IECTARG_GETSYSTIMEMILLISECS Macro implementation.
IECDEFS_ULONG GetRunningMilliSecondTime()
{ u_int32 Time_1 , Ticks_1 , Elapsed_Ticks;
  unsigned long int Timer;
  _os_getime( &Time_1 ,&Ticks_1 );
  Timer = Time_1 * 1000;
  Elapsed_Ticks = ( Ticks_1 & 0xffff )  ;
  Timer = Timer + ( Elapsed_Ticks * 10 );
  return(Timer);}
```

4.5.2 IECTARG_MEMORYFULL()

The stack calls this macro to notify that memory in the target system is full.

Expected Return Value: None

Parameters: None

e.g.

```
// IECTarg.h file
#define IECTARG_MEMORYFULL()  NULL
```

5.0 List of Global Variables

The stack uses many global variables to optimize the execution of the code. The list below is given only to avoid the user using same name as a global variable in his variable declarations and as such the user should not modify the variable names. Doing so may affect the working of the SCL.

1. IECSESSION_SESSION_INFO IEC_SessionInfo – Object of "IECSESSION_SESSION_INFO" structure.
2. IECQUE_GENERICQUE IECData_DataBaseQue – Queue to hold Database entries
3. IECQUE_GENERICQUE IECUser_UserQue; Queue to hold User Request entries
4. IECQUE_GENERICQUE IECLink_LinkTransmitQue – Object of Link Transmit Queue manager
5. IECQUE_GENERICQUE IECLink_LinkReceiveQue – Object of Link Receive Queue manager
6. IECQUE_GENERICQUE IECAppl_ApplicationTransmitQueue – Application Transmit Queue manager
7. IECQUE_GENERICQUE IECAppl_ApplicationReceiveQueue – Application Receive Queue manager
8. NETWORK_INFO* IEC_NetworkInfo – Array of Structures to hold the network details of each network
   NETWORK_INFO structure has following members
   - unsigned char NetworkStatus – Flag to indicate the network status
   - unsigned char LinkBusy – Flag to indicate if the network is busy
   - unsigned char PollSlaveNo – Holds the slave no. of one particular network to be polled
9. IECDEFS_UCHAR IECErr_Errno – Variable to hold Error number.
10. IECDATA_FRAMETYPEINFO IECData_FrameTypeInfoList[32] – List of Frame Parse and store functions pointed at by function pointers
11. IECDEFS_UCHAR UI_DisplayString[1000] – String contains Frame Data.
12. IECDEFS_NONE (*FuncPtr[27])(IECDEFS_VOIDPTR) – FuncPtr is Array Of Function pointer, Each Function takes IECDEFS_VOIDPTR as parameter and returns none
13. IECDEFS_CHAR* IECErrorMsgs[40] – IECErrorMsgs is array of strings which contains Error Descriptions
14. IECDEFS_ULONG IECUser_PreviousScanTime – Variable to hold Previous Scan Time.

6.0 IEC Error checking

The SCL stores error information on the error it encounters in a global variable named 'IECErr_Errno' of type unsigned char. The possible error codes are defined in the IECErr.h file and are reproduced below for easy reference. The error messages are displayed only when 'IECTARG_UIENABLED' macro returns 1 (true).

6.1 Errors generated in Database Processing

6.1.1 IECERR_NOMEMORY – Not Enough Memory
6.1.2 IECERR_INVALID DATA COUNT – Invalid Data Count
6.1.3 IECERR_DATA NOT STORED – Data is not stored in database

6.2 Errors generated in Application Layer

6.2.1 IECERR_INVALID ASDU – Asdu address is wrong
6.2.2 IECERR_INVALID COT – Invalid Cause of Transmission
6.2.3 IECERR_INVALID SUPPLEMENTARY INFO – Invalid Supplementary Information
6.2.4 IECERR_DUPLICATE REQUEST – Duplicate Request
6.2.5 IECERR_INVALID_SCANNO – Invalid Scan number
6.2.6 IECERR_NEGATIVE_COT – Negative Cause of transmission
6.2.7 IECERR_APPL_TIMEOUT – Application Timeout Occurs
6.2.8 IECERR_APPL_RETRY_OVER – Application Retry Count is Exceeded
6.2.9 IECERR_INVALID_PRIORITY – Invalid Priority

6.3 Errors generated in Physical and Link Layer

6.3.1 IECERR_ FRAME RECEIVED FROM MASTER – Frame from master(primary) to slave(secondary)
6.3.2 IECERR_ MISMATCH IN SESSION NUMBER OF RESPONSE AND REQUEST – Mismatch in session number of Response and Request
6.3.3 IECERR_ MISMATCH IN FUNCTION CODE OF RESPONSE AND REQUEST – Mismatch in function code of response and request
6.3.4 IECERR_UNKNOWN COMMAND FROM MASTER – Unknown Command from Master
6.3.5 IECERR_MISMATCH IN STATION ADDRESS AND ASDU ADDRESS OF REQUEST – Mismatch in station address and ASDU address of request
6.3.6 IECERR_CHECKSUM ERROR IN RESPONSE FRAME – Checksum Error in Response Frame
6.3.7 IECERR_LENGTH 1 OF RESPONSE FRAME NOT AVAILABLE – Length 1 of response frame not available
6.3.8 IECERR_LENGTH 2 OF RESPONSE FRAME NOT AVAILABLE – Length 2 of response frame not available
6.3.9 IECERR_MISMATCH IN DUAL LENGTH BYTES OF RESPONSE FRAME – Mismatch in dual length bytes of response frame
6.3.10 IECERR_START OF FRAME NOT CORRECT IN RESPONSE FRAME – Start of frame not correct in response frame
6.3.11 IECERR_END OF FRAME NOT FOUND IN RESPONSE FRAME – End of frame not found in response frame
6.3.12 IECERR_COMMUNICATION READ ERROR – Communication read error
6.3.13 IECERR_COMMUNICATION WRITE ERROR – Communication write error
6.3.14 IECERR_COMMUNICATION INIT ERROR – Communication init error
6.3.15 IECERR_LINK_TIMEOUT – Link Timeout occurs
6.3.16 IECERR_LINK_RETRY_OVER – Link Retry Count is exceeded
6.3.17 IECERR_STATIONADDR_INVALID – Invalid Station Address
6.3.18 IECERR_UNKNOWN_REQUEST – Unknown Request
6.3.19 IECERR_MAX_NETWORKS_EXCEEDED – Network number exceeded configured maximum number of Networks.
6.3.20 IECERR_MAX_SESSIONS_EXCEEDED – Session number exceeded configured maximum number of sessions.
6.3.21 IECERR_SESSIONNO_INVALID – Invalid Session Number
6.3.22 IECERR_NETWORKNO_INVALID – Invalid Network number
6.3.23 IECERR_SLAVENO_INVALID – Invalid Slave Number
6.3.24 IECERR_MAX_SLAVESINNETWORK_EXCEEDED – Slaves in network exceeded

7.0 PROTOCOL Entry Functions and USER Level Functions

7.1 PROTOCOL Entry Functions

The end user needs to call the protocol entry functions in his application program to avail the functionality of the protocol. End user can also call the USER level functions depending on his requirement which are explained in the next section. The main interface function from the user application to the SCL is the 'IECDriver_Main' interface function. Driver must be initialized by calling 'IECDriver_Init' function before calling 'IECDriver_Main' function. Reset Communication Unit, Time Synchronization & General Interrogation commands are sent once to all the sessions when 'IECDriver_Init' function is called. 'IECDriver_Main' function must be followed by calling 'IECDriver_DeInit' function. The 'IECDriver_Main' function is called by the user application cyclically in a loop irrespective of availability of any data in the communication path buffer. This function sends requests to slave(s), tries to read data from the communication paths and processes the received data. An example of such a call is as given below:

```
main(void) {
   IECDriver_Init(); // Initialization of driver done successfully
   while(1) {  IECDriver_Main();}//infinite loop
   IECDriver_DeInit(); //De-initialization of driver done successfully
} /*end of main*/
```

The sections below describe those functions. The source code file name where these functions are defined is given against each function.

7.1.1 IECDriver_Init()

This function initializes all the queues, the other parameters like sessions and networks, initializes all the communication paths and also initializes the database. This function also internally sends Reset Communication Unit, Time Synchronization & General Interrogation commands once to all sessions. This function should be called before calling any other functions.

File Name: IECDriver.c
Expected Return Value: None
Parameters: None

7.1.2 IECDriver_Main ()

This function updates all the timers, processes the timeouts, scans the user request queue, does a physical read on all the communication paths, scans the link receive queue, application receive queue, database queue, application transmit queue, link transmit queue and does a physical write on the communication paths.

File Name: IECDriver.c
Expected Return Value: None
Parameters: None

7.1.3 IECDriver_DeInit ()

This function frees all the memory allocated to the sessions and networks, closes all the communication paths and de-initializes the database. This function has to be called before closing the user application.

File Name: IECDriver.c
Expected Return Value: None
Parameters: None

7.2 USER Level Functions

7.2.1 IECUser_ResetCommunicationUnit()

This User function is used to send a 'Reset Communication Unit' command to slave. Internally this function inserts a 'Reset Communication Unit' command into Application Transmit Queue/User Request Queue.

A reset of the communication function is effected by means of a reset command from the control system. This is generally transmitted by the control system when:

- the control system is initialized
- the protection equipment does not respond during a certain t_{wz} (cycle repeat time) period.

The reasons for the expiry of the t_{wz} period may be:

- protection equipment switched off;
- protection equipment not present;
- protection equipment being initialized;
- communication interrupted.

This reset command does not affect the protection function, but only resets the communication part of the protection equipment. The reset command can be transmitted as:

- reset frame count bit (FCB), or
- reset communication unit (CU).

The decision on this is taken in the control system.

The first access of the relay after start/restart or hardware/software reset must be with a Request for Communication Unit reset, or Request for Frame Count Bit reset. The relay will not respond to any message even if correctly addressed unless either has been successfully executed.

In the case of reset CU, this command resets the communication unit, Frame Count Bit and all buffers. Relay expects next frame's FCB bit to be set. Reset CU can be transmitted after a relatively long interruption of the communication link or after the initial set-up operation, so as to erase any old messages that may be in the transmission queue. Local indications of the protection equipment are not affected by this. A reset command having been received by the protection equipment is answered by the appropriate reset acknowledgement. The reset acknowledgement is always the first message to be sent, even if there are other messages in the transmission queue. Once the command is sent successfully slave will respond in type 5 frame format. For more information refer section 4.1.7.

File Name: IECDriver.c

Expected Return Value: unsigned char –TRUE (1) if no errors occurred during inserting command into Application Transmit Queue/User Request Queue.
– FALSE (0) Otherwise.

Parameters: unsigned short SessionNo – Session Number
*IECUSER_REQUEST *Request – Object of IECUSER_REQUEST Structure to hold user request Information. The members of this structure are –*
1) Priority of request - varies from 0 to 9
2) Repeat Time interval for request (In milliseconds)
3) User Callback Function.

The function 'IECUser_ResetCommunicationUnit' is called before the while loop. If 'Repeat Time Interval for request' (Second member of the Request structure) is Zero then Reset Communication Unit command is sent to slave only once else it is sent to slave after every specified time interval. A sample example is given at the end of this section.

Note: End user may call this function from inside the while loop, which is not usually done. In such case the 'Repeat Time interval for request' (Second member of the Request structure) should be Zero.

7.2.2 IECUser_ResetFCB()

This User function is used to send a 'Reset Frame Count bit' command to slave. Internally this function inserts a 'Reset Frame Count bit' command into Application Transmit Queue/User Request Queue.

A reset of the communication function is effected by means of a reset command from the control system. This is generally transmitted by the control system when:

- the control system is initialized
- the protection equipment does not respond during a certain t_{wz} (cycle repeat time) period.

The reasons for the expiry of the t_{wz} period may be:

- protection equipment switched off;
- protection equipment not present;
- protection equipment being initialized;
- communication interrupted.

This reset command does not affect the protection function, but only resets the communication part of the protection equipment. The reset command can be transmitted as:

- reset frame count bit (FCB), or
- reset communication unit (CU).

The decision on this is taken in the control system.

The first access of the relay after start/restart or hardware/software reset must be with a Request for Communication Unit reset, or Request for Frame Count Bit reset. The relay will not respond to any message even if correctly addressed unless either has been successfully executed.

In the case of reset FCB, the internal FCB bit in the protection is set to '0'. Messages in the transmission buffer are not deleted. Any general interrogation, disturbance data transmission or generic service in progress is aborted by the protection equipment without any message. A reset command having been received by the protection equipment is answered by the appropriate reset acknowledgement. The reset acknowledgement is always the first message to be sent, even if there are other messages in the transmission queue. Once the command is sent successfully slave will respond in type 5 frame format. For more information refer section 4.1.7.

File Name: IECDriver.c

Expected Return Value: unsigned char –TRUE (1) if no errors occurred during inserting command into Application Transmit Queue/User Request Queue.
– FALSE (0) Otherwise.

Parameters: unsigned short SessionNo – Session Number
*IECUSER_REQUEST *Request – Object of IECUSER_REQUEST Structure to hold user request Information. The members of this structure are –*
1) priority of request - varies from 0 to 9
2) Repeat Time interval for request (In milliseconds)
3) User Callback Function

The function 'IECUser_ResetFCB' is called before the while loop. If Repeat Time Interval for request is Zero then Reset of FCB command is sent to slave only once else it is sent to slave after every specified time interval. A sample example is given at the end of this section.

Note: End user may call this function from inside the while loop, which is not usually done. In such case the 'Repeat Time interval for request' (Second member of the Request structure) should be Zero.

7.2.3 IECUser_GeneralInterrogation ()

This User function is used to send a 'General Interrogation' command to slave. Internally this function inserts a 'General Interrogation' command into Application Transmit Queue/User Request Queue.

General Interrogation is used to retrieve the state of certain events at the time of the interrogation. In the control direction, general interrogation (GI) is initiated by means of a GI command. The GI initiation command is transmitted individually for each protection equipment from the control system. It is recommended that GI initiation should be transmitted at intervals of 15 min (or more). Furthermore, the GI initiation message is always sent after an initial procedure. The protection equipment keeps a list of all messages subject to general interrogation. When the

list of messages subject to GI has been processed following a GI command, a GI termination message is then transmitted. A new GI cycle is not initiated in the protection equipment until a new GI impulse is received from the control system. If a GI impulse occurs within a GI general interrogation cycle, then the current GI cycle will be aborted without a GI termination message. The new cycle will then recommence from the beginning (with the first message subject to GI). Once the command is sent successfully slave will respond in type 1 frame format. For more information refer section 4.1.3.

File Name: IECDriver.c
Expected Return Value: unsigned char –TRUE (1) if no errors occurred during inserting command into Application Transmit Queue/User Request Queue.
– FALSE (0) Otherwise.
Parameters: unsigned short SessionNo – Session Number
*IECUSER_REQUEST *Request – Object of IECUSER_REQUEST Structure to hold user request Information. The members of this structure are –*
1) *Priority of request - varies from 0 to 9*
2) *Repeat Time interval for request (In milliseconds)*
3) *User Callback Function*

The function 'IECUser_GeneralInterrogation' is called before the while loop. If Repeat Time Interval for request is Zero then General Interrogation command is sent to slave only once else it is sent to slave after every specified time interval. A sample example is given at the end of this section.
Note: End user may call this function from inside the while loop, which is not usually done. In such case the 'Repeat Time interval for request' (Second member of the Request structure) should be Zero.

7.2.4 IECUser_RequestLinkStatus()

This User function is used to send a 'Request Link Status' command to slave. Internally this function inserts a 'Request Link Status' command into Application Transmit Queue/User Request Queue.

This command is used to request the status of data link. A slave will respond to this request with 'Link Status Acknowledge' message.
File Name: IECDriver.c
Expected Return Value: unsigned char –TRUE (1) if no errors occurred during inserting command into Application Transmit Queue/User Request Queue.
– FALSE (0) Otherwise.
Parameters: unsigned short SessionNo – Session Number
*IECUSER_REQUEST *Request – Object of IECUSER_REQUEST Structure to hold user request Information. The members of this structure are –*
1) *priority of request - varies from 0 to 9*
2) *Repeat Time interval for request (In milliseconds)*
3) *User Callback Function*

The function 'IECUser_RequestLinkStatus' is called before the while loop. If Repeat Time Interval for request is Zero then Link Status Request command is sent to slave only once else it is sent to slave after every specified time interval. A sample example is given at the end of this section.
Note: End user may call this function from inside the while loop, which is not usually done. In such case the 'Repeat Time interval for request' (Second member of the Request structure) should be Zero.

7.2.5 IECUser_TimeSynchronisation ()

This User function is used to send a 'Time Synchronization' command to slave. Internally this function inserts a 'Time Synchronization' command into Application Transmit Queue/User Request Queue.

Usually, the time synchronize command is used to synchronize time of all secondary devices on a network. To achieve this we set the destination address to be Global (255). In this case we do not get replies from the secondary stations. You can also use the time synchronize command to set the time of an individual secondary station. In this case the address of the secondary station is used and Secondary station sends acknowledge with time synchronize confirmation message.

File Name: IECDriver.c
Expected Return Value: unsigned char –TRUE (1) if no errors occurred during inserting command into Application Transmit Queue/User Request Queue.
– FALSE (0) Otherwise.
Parameters: unsigned short SessionNo – Session Number
*IECUSER_REQUEST *Request – Object of IECUSER_REQUEST Structure to hold user request Information. The members of this structure are –*
1) priority of request - varies from 0 to 9
2) Repeat Time interval for request (In milliseconds)
3) User Callback Function

The function 'IECUser_TimeSynchronisation' is called before the while loop. If Repeat Time Interval for request is Zero then Time Synchronization command is sent to slave only once else it is sent to slave after every specified time interval. A sample example is given at the end of this section.
Note: End user may call this function from inside the while loop, which is not usually done. In such case the 'Repeat Time interval for request' (Second member of the Request structure) should be Zero.

7.2.6 IECUser_GeneralCommand()

This User function is used to send a 'General command' to slave. Internally this function inserts a 'General command' into Application Transmit Queue/User Request Queue.
General command is used to send following parameters.

Information Number	Description	DCO/DPI	Function Type (typical)
16	auto-recloser on/off	2 (ON)/1 (OFF)	t(z), I>>, ΔI_L
17	teleprotection on/off	2 (ON)/1 (OFF)	t(z), I>>
18	protection on/off	2 (ON)/1 (OFF)	t(z), I>>, ΔI_T, ΔI_L
19	LED reset	2 (ON)	t(z), I>>, ΔI_T, ΔI_L
23	activate characteristic 1	2 (ON)	t(z)
24	activate characteristic 2	2 (ON)	t(z)
25	activate characteristic 3	2 (ON)	t(z)
26	activate characteristic 4	2 (ON)	t(z)

Semantics of Function Type

Function Type	Description
128	t(z) (distance protection)
160	I>> (overcurrent protection)
176	ΔI_T (transformer differential protection)
192	ΔI_L (line differential protection)

Once the command is sent successfully slave will respond in type 1 frame format. For more information refer section 4.1.3.
File Name: IECDriver.c
Expected Return Value: unsigned char –TRUE (1) if no errors occurred during inserting command into Application Transmit Queue/User Request Queue.

– FALSE (0) Otherwise.

Parameters: unsigned short SessionNo – Session Number
*IECUSER_REQUEST *Request – Object of IECUSER_REQUEST Structure to hold user request Information. The members of this structure are –*
1) priority of request - varies from 0 to 9
2) Repeat Time interval for request (In milliseconds)
3) *User Callback Function*
unsigned char FunType – Function Type
unsigned char InfNo – Information Number
unsigned char DpiValue – Double point information Value

The function 'IECUser_GeneralCommand' is called before the while loop. If Repeat Time Interval for request is Zero then General Command is sent to slave only once else it is sent to slave after every specified time interval. A sample example is given at the end of this section. Note: End user may call this function from inside the while loop, which is not usually done. In such case the 'Repeat Time interval for request' (Second member of the Request structure) should be Zero.

e.g.

```
IECUSER_REQUEST request1 = {6,0,IECDEFS_NULL}, request2 = {5,0,IECDEFS_NULL},
                request3 = {5,30000,IECDEFS_NULL}, request4 = {5,180000,IECDEFS_NULL};
    main(void)
     { IECDriver_Init(); //Initialization of driver done successfully
       // this command resets the communication unit, Frame Count Bit and all buffers
        IECUser_ResetCommunicationUnit( 0, &request1);  // Time interval is Zero
       // internal FCB bit in the protection is set to '0'.
        IECUser_ResetFCB(0,&request2);                  // Time interval is Zero
       // request the status of data link
       IECUser_RequestLinkStatus(0, &request1);        // Time interval is Zero
       //Send General Interrogation to slave
        IECUser_GeneralInterrogation(0,&request4);      // Time interval is 180000 milliseconds
        //send time synchronization command to slave
        IECUser_TimeSynchronisation(0,&request3);       // Time interval is 30000 milliseconds
        //Send general command to slave
        IECUser_GeneralCommand(0,&request2,160,24,2); // Time interval is Zero
       while(1) { IECDriver_Main(); }   //  infinite loop
      IECDriver_DeInit(); //De-initialization of driver done successfully
} /*end of main*/
```

8 Technical Specifications

Parameter	Value
Standard	IEC 60870-5-103 Companion Standard, Technical Committee 57 Working Group 03 www.iec.au
Functions Supported	1) Reset Communication Unit 2) Reset FCB 3) General Interrogation 4) Request Link Status 5) Time Synchronization 6) General Command 7) Class1 Polling 8) Class2 Polling
Porting Methodology	User Definable 'C' Macros 1) For User Database Interface 2) For Link/Physical Layer Interface 3) For Application Layer Interface
Development Language	ANSI 'C'
Supported Operating Systems	Portable to any 'C' supporting platform