ALEPH 90-162
DATACQ 90-12
March 1990

J.F.Renardy SACLAY

ORGANISATION EUROPÉENNE POUR LA RECHERCHE NUCLÉAIRE
EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH
Laboratoire Européen pour la Physique des Particules
European Laboratory for Particle Physics

The ALEPH-LEP Communication System

The Aleph LEP Communication System (LEPCOM) is a set of routines which allows to exchange informations between ALEPH and the LEP machine.

This material was prepared and typeset using VAX DOCUMENT V1.1.

Revision/Update Information:	Documentation version 1.0
Software Version:	VAX/VMS Version 5.0 and above

Contents

Preface

The Communications between LEP and ALEPH are based on a general package provided by the LEP support group, supplemented by a set of routines designed to match the general package to the specificities of ALEPH. This document describes both the genaral package provided by LEP and the LEPCOM programs and libraries.

Structure of this Document

This document is divided into four distinct parts:

- A section that gives an overview of the LEP Communications with Experiments.
- A user guide section describing the components of LEPCOM.
- A section describing in detail how LEPCOM is implemented.
- A reference section that documents all the routine interfaces.

Intended Audience

This manual is intended for all users of the ALEPH-LEP Communication System.

PART I Overview of the Communications between LEP and the Experiments

This section provides an overview of the Communications between LEP and the Experiments, as provided by the LEP support group. It also show the need for an ALEPH specific package to map the general service provided by the LEP software to the ALEPH environement.

1 Overview of the LEP package

1.1 Introduction

The Experiments need to exchange data with the LEP machine. Following the CERN usge, the LEP support group has provided a package to allow this exchange. The relevant features of this package are summarized below. For more informations, the reader is refered to the detailed descriptions provided by LEP.

1.2 The LEP Tables

The data to be exchanged between LEP and the Experiments are organized in Tables. Each Table contains related data (for instance data for a given equipement, data updated at each SPS cycle, data interesting all CERN users...). The only rule is that all data in a Table originate either from LEP or from one Experiment. Thus, each Table has an owner who maintains data in the Table, the others are only reader of that Table.

All tables definitions are maitained by LEP. Time to time, new tables are added, data structures of existing tables are modified or old tables are discarded. When this happens, all software related to the modified Tables needs update. This update needs synchronization. The usual procedure, for data from LEP, is to allows an overlap period where both the old and the new tables are usable.

1.3 Implementation of the LEP package

According to the message passing paradigm of the LEP Control System, the Table access is based on exchange of data between a Table server in LEP and a client program in the Experiment's computer.

1.3.1 The Table Server

The master copy of each Table is held in a LEP machine: the Table Server. All users of the Tables, both in LEP and in the Experiments, refer to the Server for Table access. For efficiency reasons, the Server may be split into several machines, then each table is handled by a specified machine, but this is transparent to the user, and in the following we consider the Table Server as a single process.

1.3.2 The Table access primitives

1.3.2.1 GetTable/PutTable

The two basic functions provided to acces Tables are:

- A READ call to get a Table from the Table Server (GetTable)
- A WRITE call to put a Table in the Server (PutTable).

These functions have only one argument: the Table to transfer. The Write call is only allowed for the owner of the Table.

1.3.2.2 InitTable

Before transfering a Table, one need to initialize it. This is done by a set of routines (one for each Table). The InitTable routine fill the Table header with the informations needed by the GetTable/PutTable routines to perform the transfer. The main informations needed are the Table structure (from the Table name) and the name of the corresponding Server (from the Handle file).

1.4 Technical details

the Table access is based on remote procedure calls. This system is based on the UDP facility of TCP/IP. It is implemented with the Network Compiler developped by the LEP Control Group.

In the following we describe some of the technical details relevant to the use of the package in ALEPH.

1.4.1 Table definition

The Table definitions are maintained by LEP. For each Table, there is a master file .TAB. This file contains all the informations needed to define and access the Table. This file is writen in a data definition language, but is easy to understand.

From this file the network compiler produces all the files needed to access the Table. Some of these files are relevant to the Experiments, and are provided by LEP. The more interesting ones are the Include files describing the Table in FORTRAN and in C, and the InitTable routine. These files and the .TAB file are available in the directory a_lep$lsrc.

1.4.2 Server definition

The name of the Server containing a Table is subject to change, for performance reasons. It has been choosen to use a kind of logical name: Each Table is compiled with the name of its Server, but this name is not the name of an actual machine, only a generic Server name. The actual machine is selected at run time by translating the generic name to the actual name with the help of an equivalence file: lep$control:handle.equiv (following the Unix convention). This file is provided by LEP. **One has to be sure that this equivalence file is up to date.**

2 The ALEPH specific LEP Communication package

The ALEPH Online system uses the paradigm of processes communicating through shared memory. The LEP Tables must be accessible in a shared global section.

The shared global section aproach is not flexible: any change to the definition of the section imply a synchronized relinking of all user programs. So one needs independance against LEP Tables evolutions.

These two requirements have been met with two sets of routines: The LEP_Access library and the LST_Access library. These routines cannot eliminate the requirement of relinking the programs in case of an update of the Tables, but they can eliminate the need for a compilation, or in the worst case the need for modifications of the user code.

2.1 The LEP Access library

These routines help in writing code insensitive to modifications of the Tables. These routines are mainly used on the TCP/IP machine and are not directly used by the LEP data user.

To implement that, an extended Table format was defined, by adding a LEP_Access header in front of all Tables. This header contains the data needed by LEP_Access to identify and characterize the tables.

Two routines Create_one_Table and Init_one_Table are used to initialize a Table. Create_one_Table reseves the Table space and initialize the LEP_Access header. Init_one_Table calls the relevant IniTable routine.

The GetTable and PutTable routines are already Table insensitive, but they have been complemented by the routines Read_one_Table and Write_one_Table to accomodate the extended table format.

A check of the validity of the data in Tables was added to the LEP_Access routines. For each data item, bounds are defined. Data outside of these bounds are replaced by the nearest boundary value, and a return code is issued. A diagnostic is also writen to the log file. The check routines know that the value -1 is used by LEP software to indicate unknown value.

In addition LEP_Access contains various utility routines like Copy_Table, Get_Table_length...

2.2 The LST Global Section and the Server

The LEP Tables are grouped in a shared Global Section known as LST. This shared Global Section was implemented by using the GBL library of D.Botterill and J.Harvey. All users of the LEP data attach to this Global Section to access the data.

A server program (LEPCOM, file LST_Server) running on the TCP/IP machine (AL0W11) ensures that the data in the LST is coherent with the Tables in the LEP Server: If a Table is updated in LEP, it will be read into the LST. If an ALEPH owned Table is updated in the LST, it will be writen to the LEP Server.

The LST_Server and the users of the LST Global Section uses the services provided by the LEP_Access library routines to ease the access to the data.

2.2.1 The LST Access library

The LST is composed of two parts:

- A header describing the content of the LST.
- A data part composed of Tables.

The LST_Access library contains three kind of routines:

- Routines for mapping the Global Section. These routines are calling the corresponding GBL functions. These routines are needed to relieve the user of the compexity of the general routines, and to ensure a coherent usage of the LST by the various users.
- High level routines for Table access. For the user accessing only one Table in the LST, these LST_read and LST_write routines are the easyest way to get access to the data. They are not optimized for efficiency, but for ease of use (only one call).
- Utility routines.

2.2.2 The LST Server

This program is an infinite loop with two actions: an update routine and a wait until the next update. The update routine first reads the basic Table (currently PAGE101). Then if an ALEPH owned Table has been updated in the LST since the last update, this Table is writen to LEP. If no write is performed, then another Table is read from LEP. If one of the read Tables is updated, this Table is put in the LST Global Section. If the read Tables are already in the LST, the Global Section is not updated.

The validity of the Tables is derived from their timestamps. The frequency of the updates is automaticaly adjusted to the update frequency of the LEP Tables.

PART II LEPCOM user guide

This section contains four parts.

- The first deals with LEP_ACCESS, the library used to supplement the LEP provided software.
- The second one describes LST_ACCESS, the library providing access to the shared global section containing the LEP data.
- The third one describes the LST_SERVER program, the program responsible for maintaining the global section up to date.
- The fourth one describes the examples provided with the LEPCOM package.

3 The LEP Access library

This chapter describes the functions provided by the LEP_Access library. These subroutines are intended to be used by programs running on the TCP/IP machine (ALOW11). They are not of direct interest to the users of the LEP data, but are included for documentation purposes. All these routines returns a completion code which has to be checked.

3.1 Table initialization

The Table initialization routines are provided to create and initialize a Table in an array.

3.1.1 Table creation

The Table creation routine structures an array provided by the user into a Table. The array has to be large enough to contain the Table. The routine initializes the header of the Table. A Table has to be created by this routine to be usable by the other routines of LEPCOM. This routine sets the status of the Table in a state allowing further exchanges with LEP. If the default status is not the required one, the user has to set/reset explicitly the corresponding bits.

3.1.2 Table initialization

This routine performs the LEP initialization which is required if the Table has to be transfered to/from LEP.

3.2 Table read/write

These routines performs the actual tranfer of Tables between the program and the LEP Server. They also performs a check of the validity of the data being transfered. On a read, the data are checked after the read, invalid data are corrected, a diagnostic is writen to the log file and a warning return code is issued. On a write, the data are checked before the write, invalid data are corrected, a diagnostic is writen to the log file, an error return code is issued and the write is aborted.

3.3 Other functions

The LEP_Acess library contains three more utility subroutines:

- Copy_Table
  This routine is used internaly by the LST_Access library to transfer Tables between the LST and local Tables. It can be used when a Table copy is needed. It checks that the source and destination Tables are instances of the same Table.

- Get_Table_Name
  This inquiry routine returns the name of the Table. This is useful if the Table is obtained indirectly.
- Get_Table_Length
  This inquiry routine returns the length of the Table. This is the only way to obtain the length of a Table. It is specialy useful when one create several Tables in an array.

4 The LST Access library

This chapter describes the functions provided by the LST_Access library. All these routines returns a completion code which has to be checked.

4.1 The Global Section

The LST Global Section contains all the LEP data relevant to ALEPH. All programs refering to the LEP data should connect to the LST Global Section for data access. The data in the LST Global Section are maintained by the LEP Server, described in the next chapter. One can use the routines in the next sections to ease access to the LST Global Section. The file a_lep$dir:lstlib.opt is provided to help in linking programs using the LST Global Section. The LST Global Section is described by the include file: a_lep$inc:lst_gbl.inc

The services provided by the LST_Access library are:

- Mapping of the LST Global Section.
- Global access to the Tables in the LST.
- LST Initialization.
- Utilities.

4.2 Mapping of LST

In order to access a Global Section, one must map to that section. The GBL package provides routines to map and unmap to any Global Section following the ALEPH rules. To avoid the complexities of the calling sequences of these routines, these routines have been supplemented by similar routines taking into account the specificities of the LST Global Section. Three routines are provided:

- LST_Init
  This is the basic routine for mapping the LST Global Section. This routine can be called several time to modify the access mode to the LST.
- LST_Unmap
  This routine will remove the connection to the LST. This may be usefull to remove load on the cluster.
- LST_Remap
  This routine is a way to change the access mode to the LST. This is intended to be faster than the LST_Init call.

4.3 Global Table access

The Global Access routines are provided to minimize the programming effort in accessing the LST Global Section. The read or write of one Table in the LST is done via a single call to the access routine. The two Global Access routines are:

- LST_Read
  This routine retrieves a Table from the LST and passes it to the user program.
- LST_Write
  This routine copy the Table provided as argument to the LST.

4.4 LST Initialization

All the routines in the LST Access library assume that he LST Global Section exists before being accessed. A stand-alone utility program (LST_Create.exe) is used to (re)create the LST Global Section. This program can only be run when no other user of the LST Global Section is active. This program will erase all data present in the LST.

4.5 Utilities

- LST_Access
  This routine, auxiliary of LST_Read/Write, maps to the LST and search a named Table in it.
- LST_Entry_Init
  This routine is used by LST_Create to initialize a Table in the LST Global Section.
- Get_LST_Max_Tables
  This inquiry routine returns the number of Tables present in the LST Global Section.
- Get_LST_Table_address
  This inquiry routine returns the address of one Table present in the LST Global Section. The Table is identified by its rank in the LST Header, not by its name. Its name can be extracted subsequently by a call to Get_Table_Name.

5 The LST Server program

The LST Global Section is maintained by a server program (LST_ Server.exe) running on the TCP/IP machine (AL0W11) under the name LEPCOM_0. The Server try to ensure the identity of the Tables in the LST Global Section and in the LEP Server. The validity of the Tables is derived from their timestamps. If a Table is updated in LEP, it will be read into the LST Global Section. If an ALEPH owned Table is updated in the LST, it will be writen to the LEP Server.

5.1 Description of the Server

The LEPCOM program is an infinite loop with two actions: an update routine and a wait until the next update. This imply a delay between the update of a Table and its transfer to its destination. This delay is a few seconds in average, but can reach minutes in heavy load conditions.

5.1.1 The Update routine

The update routine first reads the basic Table. Then if an ALEPH owned Table has been updated in the LST since the last update, this Table is writen to LEP. If no write is performed, then another Table is read from LEP. If one of the read Tables is updated, this Table is put in the LST Global Section. If the read Tables are already in the LST, the Global Section is not updated.

In the previous paragraph, the tables in the LST have been divided in "the basic Table" and the others. The basic Table is currently PAGE101. It is the Table that LEP updates the most frequently. PAGE101 is updated at each SPS cycle (14s) in principle.

This algorithm implies that in case of hevy updating of the LST, a write will be performed on each update cycle and the updating of the other tables will be preempted.

5.1.2 Timing algorithms

The frequency of the updates is automaticaly adjusted to the update frequency of the basic LEP Table: The read frequency increases slowly, and is decreased when a read found an unmodified basic Table.

The other Tables are read in a round robbing fashion: at each cycle one read the next Table. If there are no write to the LST, the other Tables are thus read at the frequency of the basic Table divided by the number (-1) of Tables in the LST.

In case of read errors on the basic Table, the read frequency is progressively decreased from one read every 60 seconds to one read every 700 seconds. This is to avoid a rapid filling of the error log.

5.2 Starting/Stopping the Server

The LEPCOM server is controlled from the DAQ Server. There is no menu associated with the Server itself. All parameters controlling the operation of the LEPCOM Server are in the LST Header and can be modified by any program attaching to the LST.

5.3 Selection algorithms

For the time being, the other Tables in the LST are read sequentialy, in their order in the LST header. Any Table can be bypassed for the read if its read bit is reset.

When the LST is updated, the name of the process responsible of the update is used to define which Table has to be send to LEP. The current choice is to write the Table ALEPH1 for each update of the LST.

6 The Example Programs

This chapter describes the examples in the distributed files. The examples are described in groups according to the layer of LEPCOM used.
The details for the compilation and the linking of the examples can be found by looking at the DESCRIP.MMS file.

6.1 Examples using the basic access

The basic access is the access provided by the LEP package.

- The first example is the one found in the LEP documentation: READPAGE. It has been modified to take into account the evolution of the LEP software since that documentation. This example reads table PAGE101 and prints the comment lines.
- The second example is the LEPWRT (program file WRITE_ALEPH1) used in 1989 to send the ALEPH background data to the LEP machine. This program tries to read the a_lep$dat:lep_aleph1.dat file. If this file exists, it tries to send it to the LEP machine. If succesfull it deletes the file. Then, in any case, it waits for a few seconds before trying again.

6.2 Examples using the LEP ACCESS package

The LEP ACCESS provides a data driven interface and allows to write applications independently of the tables used.

- The display program reads (and sometime writes) tables. It uses an UPI interface to the user. The tables used are simply defined in a data statement (the declaration of the CHAR array table_name). To add or remove tables, one has only to modify the data statement.

The examples in the two previous clases use the TCP/IP communication with LEP, and thus can only be executed on one machine (currently ALOW11).

6.3 Examples using directly the LST

The LST is a Global Section maintained by the LEPCOM server. Any program can map to the LST and access its data. The program needs to know the structure of the LST (i.e. to include the file a_lep$inc:lst_gbl.inc).

- The program lst_print_times is using the data from the LST, to print the update time of all tables in the LST.
- The program lst_write_aleph1 is a template for a program updating the table ALEPH1 in direct mapping.

6.4 Examples using the LST ACCESS package

The LST ACCESS provides a data driven interface and allows to write applications independently of the tables used. It also allows the application to be insensitive to the evolution of the LST global section.

- The first example, print_lst_aleph1, shows the usage of the lst_read routine. This program don't needs update when the definition of the LST is changed.
- The LST_display program displays, with an UPI interface the content of the LST. **BEWARE, it accesses the LST in protected read mode and forbids updates of the LST** This program maps directly the data in the LST to UPI parameter pages. It uses the LST Access services to avoid the need to use to the LST include file.
- The lst_rw_display is similar, but it allows the update of the Tables. This program uses a local set of tables connected to UPI parameter pages, and uses the LST Access services to copy between the LST and the local tables.

PART III LEPCOM internals

This section provides a detailed explanation of how LEPCOM works. Users who are only interested in the programming interface should refer directly to Section IV.

This section contains the following chapters:

- How LEPCOM works: a description of the data structures used, and of the Clusterwide Shared Global Section mechanisms.
- How LST Server works: a description of the server's routines.
- How LST Create works: a description of the LST initialization program.
- How to modify LEPCOM: a guide to LEPCOM maintenance and upgrades.

7 How LEPCOM works

This chapter first describes the structures used in LEP_access and LST_accesss libraries and in the LST. It then describes the rules for using shared sections, the algorithms used in the LST_server and a few utilities routines.

7.1 The LEPCOM Table structures

The LEP_access library uses Tables. These Tables have a structure defined by the concatenation of the LEP header and of the LEPCOM header.

7.1.1 The LEP Table header

The LEP Table header is defined by the include file a_lep$src:lep_header.inc. It contains:

- An index, i.e. a pointer to a static description structure. This description is used to interpret the various fields of the table (machine dependent representation of the data). This description contains also the handle to the LEP Server.
- A version number.
- Two times (in UNIX format): the time of the last update of the Table and the time of the read into the user computer.

7.1.2 The LEPCOM Table header

The LEPCOM Table header is defined by the include file a_lep$src:table_header.inc. It contains check words, the name and size of the Table, and readable versions of the times in the LEP header.

7.1.3 Time representation

The LEP header contains time in UNIX format: one 32 bits integer containing the number of seconds since january 1st 1970. This time is GMT: it is the same at any time on all UNIX machines (The correspondance between the UNIX time and the local time is dependent on the location of the computer).

The LEPCOM header contains time in two formats: The DEC format, i.e. a character string (*18) DD_MMM_YYYY:HH:MM:SS.CC, and an ALEPH format, i.e. two integers, one date (10000*year + 100*mo +day) and one time (10000*h + 100*m +s)

7.2 The LST Global Section

The LST Global Section is described by the include file: a_lep$inc:lst_gbl.inc. It defines a structure LEPS, in the common /lst_gbls/. The Global Section comprises this common between two guard pages, see the opt file (a_lep$dir:lstlib.opt) for a description of the Global Section The LEPS structure contains a header, followed by the various tables, page aligned (the alignement is to provide guard space between tables).

7.2.1 The LST header

The LST header is described by the include file a_lep$inc:lst_header.inc. It contains a first check word, the length (in words) of the LST structure, the number of tables in the LST, the name of the Global Section ('LST_GBL'), an array of pointers to the Tables in the LST and a last check word. These check words are supplemented by the check words of the GBL package, they insure that the LST is up to date with the software version of the GBL and the LEP package.

7.2.1.1 The LST_entry_Init routine

This routine is provided as a tool to initialize the LST header. This routine is used only in the LST_create program. The aim of this routine was to pack in a single call all the actions needed to set-up a Table in the LST. The Tables in the LST are referenced by pointers in an array (in the header). The index to an entry is the main parameter of the LST_entry_init routine. The other parameters are the LST itself, the Table name and the start and end pointers to the Table.

This routine performs the following actions:

- Check the validity of its arguments and of the LST.
- Initialize the specified Table by a call to create_one_table.
- Set-up the Table pointer in the LST header.

7.2.2 The LST_Init routine

This routine initializes the LST Global Section in the requested mode, i.e. it performs the create and map global section VMS call. In fact it calls the GBL package written by J.Harvey and D.Botterill. There are several entry points in this routine:

- LST_init To initialize the LST. This call assumes that the section file exists.
- LST_init_s To initialize the LST for the Server (i.e. with the AST enabled).
- LST_init_0 To initialize the LST for LST_create (i.e. the creation of the section file is allowed).
- LST_remap To change the access mode to the LST.

- LST_unmap To unmap the LST.

In any case, the routine is merely a call to the corresponding entry in the GBL package, with the LST parameters, plus some checks: there are two check words in the LST, and they are checked by the LST_init routines.

There is also a blocking AST with LST init (server_blocking_ast). This AST is called each time a task update the LST Global Section (except the LST server itself). In this AST, one find the name of the writing task, and place it in a special common /lst_writer/. This allows LST_Server to know when the LST has been writen and by whom, and to take the corresponding actions.

7.2.3 GBL Access Modes

The LST Global Section must be accesed in one of the following four modes, depending on the action that is desired. These modes are defined by the GBL package.

Access Mode	Description
GBL_READ	Read mode grants read access to the LST and allows its sharing with other users. This mode is generally used to read data from the LST in an unprotected fashion, since other users could modify the data as it was being read.
GBL_READONLY	Read mode grants read only access to the LST and allows its sharing with other users. This mode is generally used to read data from the LST in an unprotected fashion, since other users could modify the data as it was being read.
It is not allowed to switch mode from readonly to write.	
GBL_PROT_READ	Protected read mode grants read access to the LST and allows its sharing with other readers. No writers are allowed access to the LST.
GBL_WRITE	Write mode grants write access to the LST and allows its sharing with concurrent read-mode readers. No other writers are allowed access to the LST.

The access modes described above are a sub-set of those defined by the VMS lock manager.

7.3 Utilities

7.3.1 The LST_Access subroutine

This routine is the common part of LST_read and LST_write. It contains the call to LST_init, followed by a loop on the LST entries to find the requested Table.

7.3.2 The LST_Write_Message subroutine

All the messages produced by LEPCOM are routed through a message dispatcher: LST_write_message. This allows for an easy customization of the message output. Users of LEPCOM libraries should provide their own version, based on any of the three routines provided in a_lep$src.

The LST_write_message is designed to call err_log_message, upi_write_message or any other logging system. It requests two arguments: a severity and a message. The severities used in all LEPCOM routines are choosen in the list: 'DEB0', 'INFO', 'INFD', 'ERRO'. 'DEBO' is used for debuging output and should not be printed in normal operations. 'INFO' is for similar messages. 'INFD' is used for significant informational messages, i.e. completion of an initialization, success of an update... 'ERRO' is a true error. Fatal errors cannot be catched and are not transmited to LST_write_message.

The LST_server uses its own version, in the lst_server.for file. The examples use two versions of this routine: the first one using upi_write_message (file LST_write_message) and one using straight prints (file LST_wr_mess_noupi). **These two versions contains dummy routines for err_log_message and upi_write_message.**

8 How LST Server works

8.1 LST Server initialization

The LST_Server initialization starts with initializations of the scheduler and of the LST Global Section, including a search of the number of Tables in the LST.

Then the Server builds copies of the LST Tables in local storage. These copies will be used for the actual transfers to and from LEP, that is why these copies are initialized.

The last part of initialization is the set-up of the lists to be used by a future version of the selection algorithm.

8.2 LST Server update pass

At each cycle, the Server performs:

- A read of STANDARD0 Table.
- If LST has been writen
  A write of ALEPH1.
  ELSE
  A read of the next Table (see selection algorithm).
- An update of LST, if needed (see update algorithm).
- A computation of the sleep delay (see timing algorithm).

8.2.1 Selection algorithm

The Table selection algorithm has been described previously. Currently it is a simple round robin, implemented with a static counter. The entry is search_next_table, in subroutine init_search_lists. This structure can be easily extended to support Table reading with priorities: the most important Tables being read more often.

8.3 Update algorithm

The Logical Function Check_if_Updated, compares Time stamps in the local Table with thoses of the LST Table. It returns an OK for update, if the local Table is more recent than the LST one.

But before, tests are made on the time stamps of the local Table: The basic asumption is that the time stamps in the LST are valid (if time stamps in the LST are corrupted, one may need to destroy and recreate the LST). Then, the time stamps in the local Table (just read from LEP) cannot be earlier than the ones in the LST, and cannot be later than now.

8.3.1 Timing algorithm

The compute_delay function determines the sleep time of the Server until the next update cycle. This sleep time is bounded between 5 and 200 seconds (except in case of permanent read failures where the uper bound can be increased to 500 seconds). The sleep time is computed in order to execute the next read "mean_delay" seconds after the last update of STANDARD0.

The value for "mean_delay" is slowly updated to converge towards "average_delay". "average_delay" is computed at each cycle as the estimation of the update frequency for STANDARD0. "average_delay" is computed from the difference in the creation time stamps of two consecutive Tables.

9 How LST Create works

This program is intended to initialize the LST Global Section. It uses the special entry LST_init_0 allowing the creation of the section file. If one wants to redefine the LST, the section file must not exist when this program is called.

Once the LST is mapped, LST_Create will built the LST header and the Table headers with calls to lst_entry_init. To do that it uses the LST_GBL include fille. But the number and name of Tables to be created is hard coded in the calls to lst_entry_init. This is why LST_Create has to be edited in case of a modification of the LST Global Section. The LST Global Section is set to 0 during the initialization.

At the end, LST_Create will exit, this forces the writting of the new data to the section file.

10 How to modify LEPCOM

The LEP Tables will evolve, this imply relinking of many programs, but also from time to time the need for modifications of the program sources. The various modifications can be clasified according to the level of modification needed.

10.1 How to add/delete a Table

This is the simplest intervention. The basic idea is to modify the steering routines to add/remove the table name, and to provide/remove the routines specific to the Table. The MMS file has to be modified also.

10.1.1 Modifying the steering routines

All the steering routines are very short and have the same structure. The modification is straightforward. The steering routines are:

- Create_one_Table in file Init_one_Table.for
- Init_one_Table in file Init_one_Table.for
- Check_one_Table in file Check_one_Table.for
- Display_one_Table in file Display_one_Table.for

10.1.2 Modifying the specific routines

For the specific routines, a new one must be created for each Table and each routine. One can use the rouines for he oher Tables as a guide. The specific routines are:

- Check_xxxx in file Check_one_Table.for
- Cre_upi_pp_xxxx in file Display_one_Table.for

10.1.3 Modifying the Descrip.mms file

The description file contains two lines for each Table: The compilation of the corresponding InitTable routine. The insertion of the result of the compilation in the LEP_Access library.

10.2 How to Create a different LST

If one have to modify the LST Global Section, either to add/remove a Table or because of a Table modification, one have to follow the sequence:

- Stop all processes using the LST.
- Delete the section file a_lep$dat:lst_gbl.gbl.

- Modify the LST_GBL include file.
- Modify the LST_Create program.
- Update the programs referencing the new Table.
- Run MMS.
- Run LST_Create.

Remark that in the current LST there is some free space after each Table. This free space is provided to avoid an LST reconstruction in case of a small increase in Table length.

10.3 How to modify LST Server

The modifications to the Server should be limited to the timing and selection algorithms, or it is a major revision of the full LEPCOM system.

PART IV LEPCOM Reference Manual

This section describes the programming interface to the LEPCOM routines. It uses the same format as the VMS System Services Reference Manual.

11 LEP_ACCESS library

CREATE_ONE_TABLE—Initialize one table

The Create one Table routine initializes an array with the requested table structure.

FORMAT

CREATE_ONE_TABLE *table, name, free_size*

RETURNS

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

table

VMS Usage:	**identifier**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Table identifier. The Table argument is the Table to be initialized. It is the begining of a free area.

name

VMS Usage:	**char_string**
type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

Table name. The Name argument is the name of the table to be constructed.

free_size

VMS Usage:	**longword**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Table size. The free size argument is the length (in bytes) of the free area where the Table is to be created.

DESCRIPTION

The array Table, of length free_size bytes, is initialized with the Table structure corresponding to the Table Name. This is possible if The Table Name is a known name, and the free_size is larger (or equal) to the actual Table size. The actual Table size is not returned by Create one Table, but can be obtained easily by a call to Get Table Length.

RETURN VALUES

LEP$_NOERR	Normal sucessful condition
LEP$_UNKNOWNTABLE	The requested table is unknown
LEP$_NOSPACE	Not enough Free space to create the Table

INIT_ONE_TABLE—Prepare one table for LEP communication

The Init one Table routine calls the relevant initialization routine (INITxxx) provided by LEP.

FORMAT

INIT_ONE_TABLE *table*

RETURNS

VMS Usage:	**cond_value**
type:	**longword integer**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

table

VMS Usage:	**identifier**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Table identifier. The Table argument is the Table to be initialized.

DESCRIPTION

The Table must be properly formated before calling the Init one Table routine, i.e. by a call to Create one Table. Trying to initialize a badly formated table results in condition LEP$_BADTABLESTRUC being returned.

The Table must be enabled for LEP communication (this is the default of Crerate one Table). Otherwise one get the LEP$_NOTENABLED condition.

The Table should be known by the LEP ACCESS package. For instance a Table created by a new version may produce the LEP$_TABLEUNKNOWN condition when trying to initialize it with an old version of Init one Table.

The logical name LEP$CONTROL must be properly defined (see the LEP routine description), otherwise one get the LEP$_NOEQUIV condition.

RETURN VALUES

LEP$_NOERR	Normal sucessful condition
LEP$_BADTABLESTRUC	The argument is not a valid Table
LEP$_TABLEUNKNOWN	The argument is an unknown (but valid) table
LEP$_NOTENABLED	The Table is not authorized to access LEP
LEP$_NOEQUIV	The Table cannot be located inside LEP

READ_ONE_TABLE—Read one table from LEP

The Read one Table routine calls the FgetTable routine to retrieve Table data from the network.

FORMAT

READ_ONE_TABLE *table*

RETURNS

VMS Usage:	**cond_value**
type:	**longword integer**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

table

VMS Usage:	**identifier**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Table identifier. The Table argument is the Table to be read.

DESCRIPTION

The Table must be properly formated and initalized before calling the Read one Table routine. Trying to read a badly formated table results in condition LEP$_BADTABLESTRUC being returned. Trying to read a not initialized table results in condition LEP$_NOTINIT being returned.

The FgetTable routine may fail and return the LEP$_IMPORTFAIL condition.

Once the Table data are obtained, they are checked for validity. If this check fail, invalid data are replaced by boundary values and the LEP$_DATAINV condition is returned.

RETURN VALUES

LEP$_NOERR	Normal sucessful condition
LEP$_BADTABLESTRUC	The argument is not a valid Table
LEP$_NOTINIT	The argument is not an initialized Table
LEP$_IMPORTFAIL	Network error in reading Table
LEP$_DATAINV	The Table contains invalid data, which have been corrected
LEP$_xxx	Many other conditions can be signaled by the Network package, but they are not currently documented

WRITE_ONE_TABLE—Write one table to LEP

The Write one Table routine calls the FputTable routine to send Table data over the network.

FORMAT

WRITE_ONE_TABLE *table*

RETURNS

VMS Usage:	**cond_value**
type:	**longword integer**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

table

VMS Usage:	**identifier**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Table identifier. The Table argument is the Table to be written.

DESCRIPTION

The Table must be properly formated and initalized before calling the Write one Table routine. Trying to write a badly formated table results in condition LEP$_BADTABLESTRUC being returned. Trying to write a not initialized table results in condition LEP$_NOTINIT being returned.

Before sending the Table data, they are checked for validity. If this check fail, invalid data are replaced by boundary values, the LEP$_INVDATA condition is returned and the write operation is aborted. As invalid data have been corrected, a retry will succeed.

The FputTable routine may fail and return the LEP$_EXPORTFAIL condition.

RETURN VALUES

LEP$_NOERR	Normal sucessful condition
LEP$_BADTABLESTRUC	The argument is not a valid Table
LEP$_NOTINIT	The argument is not an initialized Table
LEP$_EXPORTFAIL	Network error in writting Table
LEP$_INVDATA	The Table contains invalid data, which have been corrected
LEP$_xxx	Many other conditions can be signaled by the Network package, but they are not currently documented

COPY_TABLE—Copy data from one Table to another one

FORMAT

COPY_TABLE *mode, source, destination*

RETURNS

VMS Usage:	**cond_value**
type:	**longword integer**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

mode

VMS Usage:	**char_string**
type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

Copy mode. The Mode argument is a code describing the copy operation to perform. It can be either "NOHEADER" if only the data part has to be copied, or "HEADER" if the headers have to be copied.

source

VMS Usage:	**identifier**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Source identifier. The source argument is the Table containing the data to copy.

destination

VMS Usage:	**identifier**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Destination identifier. The destination argument is the Table to be written.

RETURN VALUES

LEP$_NOERR	Normal sucessful condition
LEP$_INVCALL	The arguments for copy are invalid. i.e. bad table structure, different names, different lengths or unsuported mode

GET_TABLE_NAME—Inquiry routine returning the name of the Table

FORMAT **GET_TABLE_NAME** *table, name*

RETURNS

VMS Usage:	**cond_value**
type:	**longword integer**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

table

VMS Usage:	**identifier**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Table identifier. The table argument is the Table whose name is inquired.

name

VMS Usage:	**char_string**
type:	**character string**
access:	**write only**
mechanism:	**by descriptor**

Table name. The Name argument is the string which receives the name of the Table.

RETURN VALUES

LEP$_NOERR	Normal sucessful condition
LEP$_BADTABLESTRUC	The argument is not a valid Table

GET_TABLE_LENGTH—Inquiry routine returning the length of the Table

FORMAT

GET_TABLE_LENGTH *table, length*

RETURNS

VMS Usage:	**cond_value**
type:	**longword integer**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

table

VMS Usage:	**identifier**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Table identifier. The table argument is the Table whose name is inquired.

length

VMS Usage:	**longword**
type:	**longword integer**
access:	**write only**
mechanism:	**by reference**

Table length. The Lengthargument is the longword which receives the size in bytes of the Table.

RETURN VALUES

LEP$_NOERR	Normal sucessful condition
LEP$_BADTABLESTRUC	The argument is not a valid Table

12 LST_ACCESS library

LST_INIT—Initialize the LST Global Section

The routine LST Init maps the LST Global Section into the caller's space and allows access to it in the requested mode.

FORMAT

LST_INIT *lst_addr, access_mode*

RETURNS

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

lst_addr

VMS Usage:	**address**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by reference**

LST address. The Lst_addr argument is the address where the LST Global Section has been mapped.)

access_mode

VMS Usage:	**longword**
type:	**longword(unsigned)**
access:	**read only**
mechanism:	**by reference**

Lock mode that is requested for the LST Global Section. The supported access_mode are defined in a_gbl$src:create_global.inc

DESCRIPTION

This is the initialization routine for accessing the LST Global Section. This call checks the existence and validity of the LST Global Section. It also checks if the area for the Global Section is correct, this should be ensured by the link commands contained in the a_lep$dir:lstlib.opt file. If the LST Global Section is valid, then it is maped to the user's workspace, and can be accessed. The LST Global Section is shared clusterwide by using the GBL package. So, the user should follow the rules for accessing these Global Sections:

- The user may want to write to the Global Section, then he must get the Global Section with an gblacc_write access, this access should be maintained for as short time as possible.
- The user may want to read and never write to the Global Section, then he must get the Global Section with an gblacc_readonly access.
- The user may want to read the Global Section, then he must get the Global Section with an gblacc_read access.

- The user may want to read the Global Section without being interrupted by writers (in order to get an unconditionaly correct view of the Global Section), then he must get the Global Section with an gblacc_prot_read access, this access should be maintained for as short time as possible.

This initialization routine can be called several times, but it is faster to call it only once at the begining, and then to use the LST_UNMAP and LST_REMAP routines to change the access mode.

RETURN VALUES

LEP$_NOERR	Normal sucessful condition
LEP$_BADLSTADDR	The LST area has an invalid address
GBL$_ALIGNERROR	The LST area has an invalid address
GBL$_CHK1FAIL	The LST Global Section is invalid
GBL$_CHK2FAIL	The LST Global Section is invalid
LEP$_BADLSTSTRUC	The LST Global Section is invalid
GBL$xxx	Other system errors for LOCKs and Global Sections

LST_UNMAP—Unmap the LST Global Section

This routine unmaps the LST Global Section from the caller's space, i.e. make the section non accessible.

FORMAT

LST_UNMAP *lst_addr*

RETURNS

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

lst_addr

VMS Usage:	**address**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

LST address. The Lst_addr argument is the address where the LST Global Section has been mapped.)

RETURN VALUES

Same as LST INIT

LST_REMAP—Remap the LST Global Section

The LST Remap routine redefines the access mode to the LST Global Section. It can be used either to change the access mode of the currently mapped section, or to remap the section after an unmap. It is equivalent, but faster than a new call to LST Init.

FORMAT

LST_REMAP *lst_addr, access_mode*

RETURNS

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

lst_addr

VMS Usage:	**address**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

LST address. The Lst_addr argument is the address where the LST Global Section has been mapped.)

access_mode

VMS Usage:	**longword**
type:	**longword(unsigned)**
access:	**read only**
mechanism:	**by reference**

Lock mode that is requested for the LST Global Section. The supported access_mode are defined in a_gbl$src:create_global.inc

RETURN VALUES

Same as LST INIT

LST_READ—Get one Table from the LST Global Section

LST Read retrieves a Table from the LST Global Section and passes it to the user routine provided.

FORMAT

LST_READ *name, user_routine*

RETURNS

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

lst

VMS Usage:	**identifier**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

LST identifier. The Lst argument is the LST Global Section.

user_routine

VMS Usage:	**procedure**
type:	**procedure entry mask**
access:	**call**
mechanism:	

The user routine is called by LST READ with one argument: the requested Table from the LST.

DESCRIPTION

This routine is used to read a table from the LST Global Section. This routine contains a call to LST_INIT, so it is not necessary to use any other routine to read a Table from the Global Section. The table is accessed in protected read, so the user routine should returns as fast as possible. The argument of the user routine is the actual table in the Global Section. The user is responsible for not modifying the data in the Global Section (the table is not write protected by a covention of the GBL package).

RETURN VALUES

Same as LST INIT, plus

LEP$_UNKNOWNTABLE	The requested table is not in the LST
LEP$_BADTABLESTRUC	The requested Table is not valid

LST_WRITE—Copy a Table to the LST Global Section

LST Write copies the data, in the Table received, to the corresponding Table in the LST Global Section.

FORMAT

LST_WRITE *table*

RETURNS

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

table

VMS Usage:	**identifier**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Table identifier. The Table argument is the Table to be copied to the LST Global Section.

DESCRIPTION

This routine is used to write a table to the LST Global Section. This routine contains a call to LST_INIT, so it is not necessary to use any other routine to write a Table to the Global Section. The table provided is copied to the Global Section after a validity check of the data. Invalid data aborts the write. Only the data part of the table is copied to the LST Global Section, it is impossible with this routine to corrupt the Global Section.

RETURN VALUES

Same as LST INIT, plus

LEP$_UNKNOWNTABLE	The requested table is not in the LST
LEP$_BADTABLESTRUC	The requested Table is not valid
LEP$_INVDATA	The Table contains invalid data, which have been corrected, but the copy was aborted
LEP$_INVCALL	The arguments for copy are invalid. Should never occurs

LST_ENTRY_INIT—Initialize one table in the LST Global Section

This routine is used by the program LST Create to initialize the tables in the LST Global Section.

FORMAT

LST_ENTRY_INIT *lst, index, name, table, table_end*

RETURNS

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

lst

VMS Usage:	**identifier**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

LST identifier. The Lst argument is the LST Global Section to be initialized.

index

VMS Usage:	**longword**
type:	**longword integer**
access:	**read only**
mechanism:	**by reference**

The index is the relative table position of the table in the LST Global Section.

name

VMS Usage:	**char_string**
type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

Table name. The Name argument is the string which contains the name of the Table to initialize.

table

VMS Usage:	**identifier**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Table identifier. The Table argument is the Table to be initialized. The Table should be contained entirely into the LST Global Section.

table_end

VMS Usage:	**identifier**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

The Table_end argument is a byte at the end of the free area where the Table is to be created. A convenient value for this argument is the end of the LST Global Section.

GET_LST_MAX_TABLES—Inquire routine returning the number of Tables in the LST Global Section

FORMAT

GET_LST_MAX_TABLES *lst, max_tables*

RETURNS

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

lst

VMS Usage:	**identifier**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

LST identifier. The Lst argument is the LST Global Section whose content is inquired

max_tables

VMS Usage:	**longword**
type:	**longword integer**
access:	**write only**
mechanism:	**by reference**

The max_tables argument is the longword which receives the total number of Tables in the LST Global Section.

RETURN VALUES

LEP$_NOERR	Normal sucessful condition
LEP$_BADLSTSTRUC	The LST Global Section is invalid

GET_LST_TABLE_ADDRESS—Inquire routine returning the address of one Table in the LST Global Section

This routine converts an index for a Table in the LST Global Section, to the address of the Table, to be used in the calls to the LEP Access routines.

FORMAT

GET_LST_TABLE_ADDRESS *lst, index, table_address*

RETURNS

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

ARGUMENTS

lst

VMS Usage:	**identifier**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

LST identifier. The Lst argument is the LST Global Section whose content is inquired

index

VMS Usage:	**longword**
type:	**longword integer**
access:	**read only**
mechanism:	**by reference**

The index is the relative table position of the table in the LST Global Section.

table_addr

VMS Usage:	**address**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by reference**

Table address. The table_addr argument is the address of the Table in the LST Global Section.

RETURN VALUES

LEP$_NOERR	Normal sucessful condition
LEP$_BADLSTSTRUC	The LST Global Section is invalid
LEP$_INVCALL	The index argument is invalid

A Installation and Use

A.1 Installation hints

To install one needs only to copy some directories, and then to run MMS. The recomended directory structure is:

- [...mgr] Installation files
- [...source] Source files
- [...nodeb] .OBJ,.OLB and .EXE (nodebug)
- [...deb] same with debug option
- [...lepsrc] Source files copied from LEP
- [...lepdir] .OBJ and .OLB from LEP

The main files to copy are in the [.mgr] and [.source] directories. One has also to copy the .OPT files from [...nodeb] and [...deb]. The fetch... command files copy the needed files from the LEP reference directories in VXCERN. **Beware of unanounced modifications in LEP files.**
The MMS uses logical names for the directories. These logical names can be set-up with the login command file.

The lepcom.com file drives the execution of the lepcom server. The lepcom server is started by the DAQ server.

A.2 TCP/IP Installation

This paragraph applies to the Wollongong software and the network status as on June 1990.
The installation of the TCP/WIN software is described in the Wollongong documentation. We are concerned here with the specific tuning needed by the LEPCOM package. To understand the requested operations, one need to know the routing rules of TCP/IP.

A.2.1 Routing rules for TCP/IP

TCP/IP network is composed of many Domains, each containing many Hosts. From one Host, one can reach directly all the other Hosts of the same Domain, but one need explicit routes for reaching Hosts on other domains. The explicit route may be either for one specific Host or for a full domain.

At CERN, the TCP/IP network is composed of many Domains:

- 128.141.x.x The Ethernet domain, comprising all the large TCP/IP machines on the main CERN Eternet.
- 128.142.x.x The main LEP domain, on the Token-ring.

- 192.16.155.x The Apollo domain.
- etc...

The management of the CERN TCP/IP network provides routes between the various CERN Domains, but these routes are not enabled on AL0W11, except the LEP-ETHER gateway to the LEP network. The management of the LEP TCP/IP network requests that the LEPCOM trafic transit through a private gateway, located at point 4 and named LSVXAL-GW. It also requests that we use explicit routing to explicit Hosts for security reasons.

A.2.2 Setting the TCP/IP routing for LEPCOM

This must be done on AL0W11.
One has to define the correct route to the LEP Server:

```
ROUTE add eslpcr e-lsvxal-gw 1
```

For debugging purposes, it may also be usefull to define routes to some test computers on the token-ring:

```
ROUTE add lsvxal-gw e-lsvxal-gw 1
ROUTE add dilsr4    e-lsvxal-gw 1
```

One should not forget to put the same commands in the file cluster$manager:startup_tcpip.com.

Then, one should check the result with the command:

```
NETSTAT -r
```

A.2.2.1 How to use ROUTE and NETSTAT

One needs privileges to use ROUTE. To use ROUTE and NETSTAT one needs to define the symbols:

```
route   :== "$sys$sysdevice:[tcpwin.netdist.etc]route route"
netstat :== "$sys$sysdevice:[tcpwin.netdist.user]netstat netstat"
```

Then one has just to type "ROUTE command" or "NETSTAT -option".

A.3 Files used by MMS

A.3.1 Directory .MGR

- DESCRIP.MMS MMS file to create LEPCOM.

A.3.2 Directory .SRC

A.3.2.1 Include files

- LEP_HEADER LEP software Table header
- TABLE_HEADER LEP_access header
- TABLE_HEADS Table_header + LEP_header
- LST_HEADER Header of LST Global Section
- LST_GBL The LST Global Section

- LST_SERVER Communications inside Server

A.3.2.2 Programs/Libraries sources in FORTRAN

- CONVERT_ERRORS Program to create error handling sources.
- LST_CREATE LST_GBL creation.
- LST_SERVER
- READ_ONE_TABLE LEP Access library, independent of Tables.
- INIT_ONE_TABLE LEP Access library, Table dispatchers.
- CHECK_ONE_TABLE LEP Access library, Table verification.
- DISPLAY_ONE_TABLE LEP Access library, Table UPI display.
- LST_INIT LST Access library, dependent of LST_GBL.
- LST_ACCESS LST Access library, independent of LST_GBL.
- LST_WRITE_MESSAGE Lst_Write_Message with UPI_Write_Message.
- LST_WR_MESS_NOUPI Lst_Write_Message with PRINT statements.

A.3.2.3 Examples source

- READPAGE.C
- DISPLAY.FOR
- WRITE_ALEPH1
- LST_PRINT_TIMES
- LST_DISPLAY.FOR
- LST_RW_DISPLAY.FOR
- LST_WRITE_ALEPH1.FOR
- LST_PRINT_ALEPH1.FOR

A.3.3 Directory .NODEB

- LSTLIB.OPT OPT file for LST and LEP_access libaries.

A.3.4 Directory .DEB

- LSTLIB.OPT OPT file for LST and LEP_access libaries.

A.3.5 Directory .LEPDIR

- LEPLIB.OPT OPT file for LEP software.

A.4 Source files created by MMS in Directory .LEPSRC

- CONVERT_LEP_CODES.FOR Translate LEP errors to VMS codes.
- LEP_ERRORS.INC Errors definition Include file

A.5 Other useful files

A.5.1 Directory .MGR

- FETCH_LEP_FILES.COM Command file to copy LEP software.
- A_LEP$xxx.FETCH Fetch files for LEP software.
- CREATE_FULLLIBNET.COM File used to make part of LEP software.
- LOGIN.COM Definition of Logical names.
- LEPCOM.COM Command file for LST Server.

A.6 Documentation files

A.6.1 Directory .MGR

- LONGWRITEUP.TXT LEP documentation.
- LEPCOM.SDML LEPCOM documentation.

A.7 How to Link User Programs

See the examples in the .MMS file.

B Debugging guide

B.1 Introduction

As far as the LEP communications failures are concerned, the system can be decomposed in three pieces:

- The LST server running on AL0W11.
- TCP/IP
- The LEP server running on PCs in the LEP control room.

Eaxch of these can be the origin of a loss of LEP communication. In order to reestablish the communications, one has to locate the defective part and to repair it.

In the following, we will review the three pieces in turn. For each piece there is a diagnostic part, explaining the tools and the procedures to follow in order to determine if that part is operational or defective, and an action part describing the possible corrective actions.

It is strongly recomended to perform the diagnostics in the order presented.

B.2 Problems with the LST Server

B.2.1 Diagnostic procedure

The LST Server has to maintain the data in the LST Global Section identical to the data in the LEP Tables. If the LST is not properly updated, while the LEP informations can be correctly obtained, then the LST Server has failed.

On AL0W11 one can display the LEP Tables with two very similar programs:

- a_lep$dir:display reading the LEP Tables through TCP/IP
- a_lep$dir:lst_display reading the tables from the LST

These two programs should produce the same results. If the LST is badly out of date, then LEPCOM_0 is not running correctly.

B.2.2 Corection actions

The first test is to see if LEPCOM_0 is running by looking at the log file: a_s$log:lepcom_0.log. If it is correctly running, one can try to kill and restart it but this will not work in general (it is restarted automatically every 7 hours). If everything is apparently working, but the LST is not updated, one can suspect a mild overwrite of the LST Global Section.

Action to be taken in case of an LST Global Section overwrite.
If a careless program overwrites the control words of the LST Global Section, this section may become unusable. In that case all the programs using LST cannot run. The only way out of this situation is to delete the corresponding global section file a_lep$dat:lst_gbl.gbl and to recreate it by calling the program LST_Create.

B.3 Problems with TCP/IP

B.3.1 Diagnostic procedure

AL0W11 is unable to communicate over TCP/IP with the PC in the LEP control room. Both from the LST_Server and the display.exe program produce error messages like: "LEP-E-IMPORTFAIL, Import service has failed" or "unknown LEP error code status = 1".

This diagnostic has to be confirmed and precised with the TCP/IP tool PING. This tool is sending a probe message to the target, the target recognises the probe message and returns it to the sender. If the TCP/IP link is working between the sender and the target, one get the average trip time, if not the PING program waits for an answer. After one minute of wait, one can be sure that the link is broken.

To probe the status of TCP/IP, one has to ping various computers on the road from ALEPH to the LEP control room. Currently the road is: Ethernet from ALEPH cluster to the ALEPH-LEP gateway (LSVXAL-GW), and then token ring to ESLPCR. It is suggested to PING the following machines in that order: VXCRNA, CERNVM, AL1W00 to check comunication with machines on the Ethernet. Then the gatway itself E-LSVXAL-GW (eternet side) and LSVXAL-GW (token ring side). Then a machine on the local token ring SVLSR4 (ALEPH alarm computer), and last the communication PC ESLPCR.

B.3.1.1 How to use PING

One needs privilege SYSPRV to use PING. To use PING one needs to define the symbol:

```
ping :== "$sys$sysdevice:[tcpwin.netdist.etc]ping ping"
```

Then one has just to type "PING nodename". One has to exit PING with ctrl-Y.

B.3.2 Corection actions

If no host on the ethernet can be accessed (VXCRNA, CERNVM, AL1W00...) then TCP/IP is not working on AL0W11. This machine has to be rebooted.

If the machines on the ethernet are accessible, but E-LSVXAL-GW or LSVXAL-GW don't respond, then the gateway is off and should be restarted. Ask the LEP operator to restart it.

If LSVXAL-GW is accessible there are two cases: if SVLSR4 don't respond, there is a problem on the token ring, If only ESLPCR does not respond, this machine should be rebooted. In any case, asks the LEP operator to fix the problem.

B.4 Problems with the LEP Server

B.4.1 Diagnostic procedure

If the LST_server and TCP/IP are working, but LEP data are not correct, there are three possibilities:

- Pagel on the TV screen is also wrong.
- Pagel is correct, but the other LEP experiments have also a comunication problem (i.e. all magnetic fields are ****).
- Only ALEPH has a comunication problem.

B.4.2 Corection actions

In any case, one has to rely on the LEP operators to diagnose and correct the problem. The first case is the simpler: the program in the PC is dead. The second case is related, but it is more subtle and the operator has curently no mean to check the comunications. The third case should never occurs (except if we miss an upgrade of the comunication software).