

---

# **Licel Transient Recorder and Ethernet-Controller**

## **Programming Manual**

**Licel GmbH**

February 12, 2010

# Contents

<b>1</b>	<b>Program function and structure</b>	<b>4</b>
1.1	Introduction	4
1.1.1	Description	4
1.1.2	Operation principles	5
1.1.3	Hardware requirements	5
1.1.4	Further Literature	5
1.2	Program organization	5
1.2.1	Structure	5
1.2.2	Modules	5
1.2.3	licel_nidaq/licel_re	5
1.2.4	licel_tcpip	6
1.2.5	licel_tr/licel_tr_tcpip	7
1.2.6	licel_util	10
1.2.7	APD functions	11
1.2.8	PMT functions	11
1.2.9	Timing functions	11
1.2.10	Security functions	12
1.2.11	Power Meter	12
1.2.12	Bore Alignment	13
1.2.13	Function arguments	13
1.2.14	Timing Parameter explanation	16
1.2.15	Low Level Commands	17
1.3	Memory organization	19
1.4	Raw Data to Physical Value Conversion	19
1.5	Acquisition Low Level Description	20
1.6	Network security	21
<b>2</b>	<b>Installation</b>	<b>22</b>
2.1	Ethernet software	22
2.1.1	Required software	22
2.2	Windows - DIO-32HS	23
2.2.1	Required software	23
2.2.2	NI-DAQ-Setup	23
2.2.3	Interface card installation	23
2.2.4	Verification	24
2.3	Linux - PCI-DIO-32HS	24
2.3.1	Kernel preparation	24
2.3.2	Necessary Files	24
2.3.3	Driver tests and card installation	25
2.3.4	Changes to /etc/modules.conf	25
2.3.5	Changes to /etc/rc.d/rc.local	25
2.3.6	Directory structure	25
2.3.7	LabView + comedi	25
2.4	Software installation	25
<b>3</b>	<b>C - Example Programs</b>	<b>26</b>
3.1	The script	26
3.2	start	27
3.3	shot	27
3.4	readout	27
3.5	example program configuration file	27
3.6	File format	28
3.7	SampleAcquis for Ethernet Applications	29
3.8	Network management utilities	30
3.8.1	Getting Started	30
3.8.2	Set Fixed IP Address	30
3.8.3	Activate DHCP Mode	30

3.8.4	SecureModeEnable	30
3.8.5	SecureModeDisable	31
<b>4</b>	<b>Appendix VB6/VB.net Programming</b>	<b>31</b>
4.1	Sample applications	31
4.1.1	Control Overview	31
4.1.2	MultipleChannel	31
4.1.3	PushModeDemo	31

# 1 Program function and structure

## 1.1 Introduction

### 1.1.1 Description

Currently the Transient Recorder can be controlled by two ways:

- via the parallel interface cards from National Instruments
- or via a Ethernet interface. The Ethernet interface has then its own parallel bus interface to the transient recorders.

The software to control the Licel Transient Recorder will

- control one or more Transient Recorders
- ensure software portability between different operating systems
- readout the transient recorders at high data transfer rates.

The software for the Ethernet controller will in addition control

- the APD module
- the PMT module
- the Trigger module

The following target systems are supported:

- LabVIEW from National Instruments
  - WinXX, Linux
  - Mac-OS
- NIDAQ-Library from National Instruments
  - WinXX with MS Visual C
  - WinXX with MS Visual Basic
- a COMEDI based Linux driver for gcc
- MS Visual C and gcc for the Ethernet controller software.
- A Visual Basic/VB.net module for the Ethernet controller software. This module is sold separately.

The software is able to:

- configure the Transient Recorders
- start the data acquisition
- stop the data acquisition
- query the Transient Recorder status
- readout the Transient Recorder
- convert the binary data to quantities with physical units.

The Ethernet based software additionally is able to:

- Set the PMT high voltage and read the PMT status.
- Set the APD high voltage, Activate the APD thermo electrical cooler and read the APD Status back.
- Set the delays on the trigger generator and to activate separately the trigger lines.

### 1.1.2 Operation principles

The communication with the Transient Recorder is performed via a parallel bus using a hardware handshake. The parallel bus is based on the digital I/O card family DIO-32HS supplied by National Instruments. For the description of the bus timing refer to the [user manual](#) from National Instruments. The used protocol is Level-Acq. This interface is also implemented internally between the Ethernet controller and the transient recorders.

### 1.1.3 Hardware requirements

Operating environment	Card type	Required Slot
Desktop-PC Win9xx	AT-DIO-32HS	ISA Slot short
Win-NT	PCI-DIO-32HS	PCI slot
Notebook	DAQCard 6533	PCMCIA slot
PXI	PXI 6533	PXI slot
MAC	PCI-DIO-32HS	PCI slot
Linux-PC	PCI-DIO-32HS	PCI slot
PC	Ethernet	Ethernet connector

### 1.1.4 Further Literature

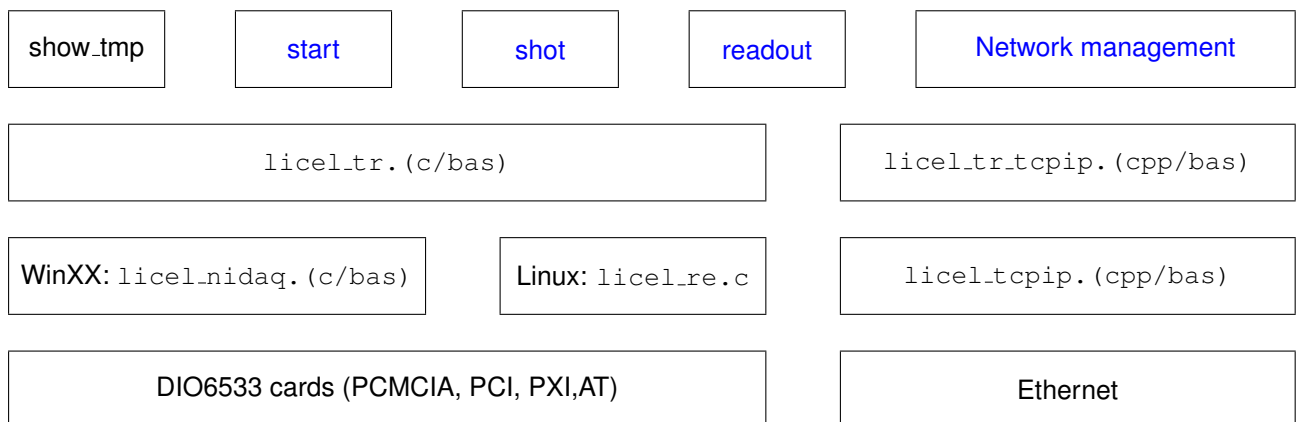
1. [DIO-653x User Manual](#)
2. [NI-DAQ User Manual for PC Compatibles](#)
3. [NI-DAQ Function Reference Manual for PC Compatibles](#)
4. [LabVIEW Measurements Manual](#)

## 1.2 Program organization

### 1.2.1 Structure

The software has layered structure where the low-level routines are encapsulated in `licel_nidaq.c` or `licel_nidaq.bas`. These routines access NI-DAQ. Under Linux the corresponding file is `licel_re.c`. This file calls the corresponding `comedilib` functions. Above this layer are `licel_tr.c` or `licel_tr.bas`. At this layer the functional tasks are translated into low-level commands.

For the Ethernet controller the basic communication routines are inside `licel_tcpip.cpp`. Above this level are the `licel_tr_tcpip.cpp`. At this layer the functional tasks are translated into low-level ASCII commands. Above this layer are the application functions.



### 1.2.2 Modules

### 1.2.3 licel\_nidaq/licel\_re

The `licel_nidaq.(c|bas)/licel_re.c` are used for the National Instruments interface cards and have four routines inside:

```

    /* Setup of the DIO card*/
int InitializeBoard(short int iBoard)
Public Function InitializeBoard(iBoard As Integer) As Integer

    /* Writing the commands to the TR*/
int WriteCommand(short int iBoard, short iCommand)
Public Function WriteCommand(iBoard As Integer, iCommand As Integer)
As Integer

    /* Reading back information from the TR*/
int ReadArray( short int iBoard, unsigned short *piBuffer,
unsigned long ulCount)
Public Function ReadArray(iBoard As Integer, piBuffer() As Integer,
ulCount As Long) As Integer

    /* Selection of TR group*/
int Select(short int iBoard, int hilow)
Public Function Select.TR(iBoard As Integer, iDevice As Integer) As Integer

```

In general these routines do not need a customer tweaking, the only place where customization may be useful is in `InitializeBoard` the `AcqDelay` can be changed between 0 (0ns) and 7 (700ns), which might help if data transfer errors do occur (readout errors usually give variable readout while reading repetitive the same dataset, if this happens it is the right time to contact Licel for help )

#### 1.2.4 licel\_tcpip

**openConnection** open the connection to specified host at a specified port

```

SOCKET openConnection(const char* host, int port);
Public Function openConnection(ByVal sHost As String, ByVal iPort As
Integer) As TcpClient

```

open the connection to specified host at a specified port, stays silent if it fails

```

Public Function openConnection(ByVal sHost As String, ByVal iPort As
Integer, ByVal silent As Boolean) As TcpClient

```

**openSecureConnection** open the connection to specified host at a specified port when the access to the controller is Limited (see [Network Security](#) for details)

```

SOCKET openSecureConnection(const char* sHost, int iPort, const char*
connectionPasswd);

```

**closeConnection** close the specified connection

```

int closeConnection(SOCKET s);
Public Function closeConnection(ByVal client As TcpClient) As Integer

```

**writeCommand** write a string to the tcpconnection, append the terminating CRLF

```

int writeCommand(SOCKET s, const char* command);
Public Function writeCommand(ByVal client As TcpClient, ByVal command
As String) As Integer

```

**readResponse** Read a ASCII response from the controller. Except for binary data transfer the controller response is a short string indicating whether the action could be performed or not. This response is terminated by a CRLF. This routine will read till it encounters a CRLF or if the amount of chars would exceed `maxLength`.

```

int readResponse(SOCKET s, char* response, int maxLength,
int nTimeoutMillisec);
Public Function readResponse(ByVal client As TcpClient, ByRef response
As String, ByVal maxLength As Integer, ByVal nTimeoutMillisec As Integer)
As Integer

```

**ReadArray** Read a binary response from the controller.

```
int ReadArray(SOCKET s, unsigned char *array, unsigned long points,
int nTimeoutMillisec);
Public Function ReadArray(ByVal client As TcpClient, ByRef Data()
As Byte, ByVal points As Long, ByVal nTimeoutMillisec As Integer)
```

### 1.2.5 licel\_tr/licel\_tr\_tcpip

There is a major difference in the programming model between the DIO-32 and the Ethernet version, the Ethernet version first selects the transient recorder and all later issued commands are addressed to this transient recorder. For tasks like starting all transient recorders separate functions are available. Before they can be called a list of transient recorders should be selected, the functions indicated by `Multiple` in the function name.

#### Controller

**Licel\_TCPIP\_ActivateDHCP** activate the DHCP mode on the controller

```
int Licel_TCPIP_ActivateDHCP(SOCKET s, int iPort, const char* passwd);
Public Function Licel_TCPIP_ActivateDHCP(ByVal client As TcpClient,
ByVal iPort As Integer, ByVal passwd As String) As Integer
```

**Licel\_TCPIP\_SetIPParameter** configure the controller for static IP configuration. Set the new IP address, the basic port number, the subnet mask and the gateway

```
int Licel_TCPIP_SetIPParameter(SOCKET s, char* newHost, char* mask, int newPort,
char* gateway, char* passwd);
Public Function Licel_TCPIP_SetIPParameter(ByVal client As TcpClient,
ByVal newHost As String, ByVal mask As String, ByVal newPort As Integer,
ByVal gateway As String, ByVal passwd As String) As Integer
```

**Licel\_TCPIP\_GetID** Get the identification string from the controller

```
int Licel_TCPIP_GetID(SOCKET s, char* buffer, int bufferLength);
Public Function Licel_TCPIP_GetID(ByVal client As TcpClient, ByRef buffer
As String, ByVal bufferLength As Integer) As Integer
```

**Licel\_TCPIP\_GetCapabilities** Get the available subcomponents of the controller like:

TR	for controlling transient recorder
APD	for APD remote control
PMT	for PMT remote control
TIMER	for the trigger timing controller
CLOUD	for transient recorder controller cloud mode
BORE	Boresight alignment system

```
int Licel_TCPIP_GetCapabilities(SOCKET s, char* cap, int bufferLength);
Public Function Licel_TCPIP_GetCapabilities(ByVal client As TcpClient,
ByRef cap() As String, ByVal maxLength As Integer, ByRef validcap As Integer)
As Integer
```

#### Transient recorder

**Licel\_TCPIP\_SelectTR** Select a transient recorder for subsequent communication

```
int Licel_TCPIP_SelectTR(s, int TR)
Public Function Licel_TCPIP_SelectTR(ByVal client As TcpClient,
ByVal TR As Integer) As Integer
```

**Licel\_TCPIP\_SelectMultipleTR** Select a list of transient recorders for subsequent communication

```
int Licel_TCPIP_SelectMultipleTR(SOCKET s, int* TRList, int trNumber);
Public Function Licel_TCPIP_SelectMultipleTR(ByVal client As TcpClient,
ByVal TRList() As Integer, ByVal trNumber As Integer) As Integer
```

**SetDiscriminatorLevel** Set the discriminator level between 0 and 63

```
int HS_Licel_Set_Discriminator_Level(short int iDevice ,int iDiscrLevel);
Public Function HS_Licel_Set_Discriminator_Level(iDevice As Integer,
iDiscrLevel As Integer) As Integer
int Licel_TCPIP_SetDiscriminatorLevel(SOCKET s, int iDiscrLevel);
Public Function Licel_TCPIP_SetDiscriminatorLevel(ByVal client As TcpClient,
ByVal iDiscrLevel As Integer) As Integer
```

**SetRange** Change the input voltage range (20,100,500mV)

```
int HS_Licel_Set_Range(iDevice,int iRange);
Public Function HS_Licel_Set_Range(iDevice As Integer, iRange As Integer) As
Integer
int Licel_TCPIP_SetInputRange(SOCKET s, iRange)
Public Function Licel_TCPIP_SetInputRange(ByVal client As TcpClient,
ByVal iRange As Integer) As Integer
```

**SetThresholdMode** Set the scale of the discriminator level. In the low threshold mode the discriminator level 63 corresponds to 25mV while in the high threshold mode it corresponds to 100mV.

```
int HS_Licel_Set_Threshold_Mode(short int iDevice, int iMode);
Public Function HS_Licel_Set_Threshold_Mode(iDevice As Integer, iMode As
Integer) As Integer
int Licel_TCPIP_SetThresholdMode(SOCKET s, int iMode)
Public Function Licel_TCPIP_SetThresholdMode(ByVal client As TcpClient,
ByVal iMode As Integer) As Integer
```

**Licel\_TCPIP\_SetSlaveMode** Set the slave mode

```
int Licel_TCPIP_SetSlaveMode(SOCKET s);
Public Function Licel_TCPIP_SetSlaveMode(ByVal client As TcpClient)
As Integer
```

**Licel\_TCPIP\_SetPushMode** Activate the push mode for the currently selected transient recorder

```
int Licel_TCPIP_SetPushMode(SOCKET s, int shots,int dataset, int numberToRead,
int memory);
Public Function Licel_TCPIP_SetPushMode(ByVal client As TcpClient,
ByVal shots As Integer, ByVal dataset As Integer, ByVal numberToRead
As Integer, ByVal memory As Integer) As Integer
/*Action Status*/
```

**Licel\_TCPIP\_GetStatus** Return the status information for one transient recorder

```
int HS_Licel_Get_Status(short int iDevice,long int * iCycles, int * iMemory,
int * iAcq_State, int * iRecording);
Public Function HS_Licel_Get_Status(iDevice As Integer, iCycles As Integer,
iMemory As Integer, iAcq_State As Integer, iRecording As Integer) As Integer
int Licel_TCPIP_GetStatus(SOCKET s, long int* iCycles, int* iMemory,
int* iAcq_State, int* iRecording)
Public Function Licel_TCPIP_GetStatus(ByVal client As TcpClient,
ByRef shotNumber As Integer, ByRef lastMemory As Integer,
ByRef acquisitionState As Integer, ByRef recording As Integer) As Integer
```

**ContinueAcquisition** Continue the recording process without a new initialisation of the memory

```
int HS_Licel_ContinueAcquisition(short int iDevice );
Public Function HS_Licel_ContinueAcquisition(iDevice As Integer) As Integer
int Licel_TCPIP_ContinueAcquisition(SOCKET s)
Public Function Licel_TCPIP_ContinueAcquisition(ByVal client As TcpClient)
As Integer
```

**Licel\_TCPIP\_MultipleContinueAcquisition** Continue the recording process for the previously selected transient recorders without a new initialisation of the memory

```
int Licel_TCPIP_MultipleContinueAcquisition(SOCKET s);
Public Function Licel_TCPIP_MultipleContinueAcquisition(ByVal client
As TcpClient) As Integer
```

**StopAcquisition** Stop the recorder after the next received trigger.

```
int HS_Licel_StopAcquisition(short int iDevice );
Public Function HS_Licel_StopAcquisition(iDevice As Integer) As Integer
int Licel_TCPIP_Stop(SOCKET s)
Public Function Licel_TCPIP_StopAcquisition(ByVal client As TcpClient)
As Integer
```

**Licel\_TCPIP\_MultipleStopAcquisition** Stop the acquisition process for the previously selected transient recorders with the next received trigger pulse

```
int Licel_TCPIP_MultipleStopAcquisition(SOCKET s);
Public Function Licel_TCPIP_MultipleStopAcquisition(ByVal client
As TcpClient) As Integer
```

**ClearMemory** Clear both memories (A and B) of the transient recorder

```
int HS_Licel_ClearMemory(short int iDevice);
Public Function HS_Licel_ClearMemory(iDevice As Integer) As Integer
int Licel_TCPIP_ClearMemory(SOCKET s)
Public Function Licel_TCPIP_ClearMemory(ByVal client As TcpClient)
As Integer
```

**Licel\_TCPIP\_MultipleClearMemory** Clear both memories (A and B) of the previously selected transient recorders

```
int Licel_TCPIP_MultipleClearMemory(SOCKET s)
Public Function Licel_TCPIP_MultipleClearMemory(ByVal client As TcpClient)
As Integer
```

**Licel\_TCPIP\_IncreaseShots** Increase the shotnumber of the TR without adding data, this can be used to make a fixed number of acquisitions based on the internal 4094 shot limit.

```
int Licel_TCPIP_IncreaseShots(SOCKET s, int shots )
```

**StartAcquisition** Start the acquisition process with a new initialisation of the memory

```
int HS_Licel_StartAcquisition(short int iDevice)
Public Function HS_Licel_StartAcquisition(iDevice As Integer) As Integer;
int Licel_TCPIP_Start(SOCKET s);
Public Function Licel_TCPIP_StartAcquisition(ByVal client As TcpClient)
As Integer
```

**Licel\_TCPIP\_MultipleStart** Start the acquisition process for the previously selected transient recorders with a new initialisation of the memory

```
int Licel_TCPIP_MultipleStartAcquisition(SOCKET s);
Public Function Licel_TCPIP_MultipleStart(ByVal client As TcpClient)
As Integer
```

**SingleShot** Acquire one shot

```
int HS_Licel_Single_Shot(short int iDevice);
Public Function HS_Licel_Single_Shot(iDevice As Integer) As Integer
int Licel_TCPIP_SingleShot(SOCKET s)
Public Function Licel_TCPIP_SingleShot(ByVal client As TcpClient) As Integer
```

**WaitForReady** Wait for the return of the transient recorder from the armed state. If the waiting time is longer than the time specified by delay, then the transient recorder will return to the idle state with the next reading of binary data

```
int HS_Licel_Wait_For_Ready(short int iDevice, int imDelay);
Public Function HS_Licel_Wait_For_Ready(iDevice As Integer, imDelay As Integer) As Integer
int Licel_TCPIP_WaitForReady(SOCKET s, imDelay)
Public Function Licel_TCPIP_WaitForReady(ByVal client As TcpClient, ByVal delay As Integer) As Integer
```

**Licel\_TCPIP\_MultipleWaitForReady** Wait until all transient recorders return from the armed state

```
int Licel_TCPIP_MultipleWaitForReady(SOCKET s, imDelay);
Public Function Licel_TCPIP_MultipleWaitForReady(ByVal client As TcpClient, ByVal delay As Integer) As Integer
/*Data*/
```

**Licel\_TCPIP\_ReadData** Read binary data into a byte array. Transient recorder data is internally 16bits wide so for every data point two bytes need to be fetched.

```
int Licel_TCPIP_ReadData(SOCKET s, int numberToRead, unsigned char* data);
Public Function Licel_TCPIP_ReadData(ByVal client As TcpClient, ByVal numberToRead As Integer, ByRef data() As Byte) As Integer
```

**GetDatasets/Read16bitwide** Read binary datasets from a transient recorder

```
int HS_Licel_Read_16bit_wide(short int iDevice, int iDataset, int iNumber, int iMemory, unsigned short * uPortData);
Public Function HS_Licel_Read_16bit_wide(iDevice As Integer, iDataset As Integer, iNumber As Integer, iMemory As Integer, uPortData() As Integer)
int Licel_TCPIP_GetDatasets(SOCKET s, int iDevice, int iDataset, int iNumber, int iMemory, unsigned char* data)
Public Function Licel_TCPIP_GetDatasets(ByVal client As TcpClient, ByVal TR As Integer, ByVal dataset As Integer, ByVal numberToRead As Integer, ByVal memory As Integer, ByRef data() As Byte) As Integer
```

**1.2.6 licel\_util**

/\* Converts the LSW and the MSW values into an integer array containing the summed up analog values. The first trash element (due to the data transmission scheme ) are also removed. \*/

```
void Licel_Combine_Analog_Datasets(unsigned short* i_lsw, unsigned short* i_msw, int iNumber, long * lAccumulated, short * iClipping);
Public Sub Licel_Combine_Analog_Datasets(i_lsw() As Integer, i_msw() As Integer, iNumber As Integer, lAccumulated() As Long, iClipping() As Integer)
```

/\* Converts the raw Photon counting data into an integer array containing the summed up photon counting values. The first trash element (due to the data transmission scheme ) are also removed. The clipping information present in the most significant bit is masked out if necessary\*/

```
void Licel_Convert_Photoncounting(unsigned short* photon_raw, int iNumber, long * photon_c, int iPurePhoton);
Public Sub Licel_Convert_Photoncounting(photon_raw() As Integer, iNumber As Integer, photon_c() As Long, iPurePhoton As Boolean)
```

/\* Normalizes the accumulated Data with respect to the number of cycles\*/

```

void Licel_Normalize_Data( long * lAccumulated, int iNumber, int iCycles,
double * dNormalized);
Public Sub Licel_Normalize_Data(lAccumulated() As Long, iNumber As Integer,
iCycles As Integer, dNormalized() As Double)

/* Scales the normalized data with respect to the input range */
void Licel_Scale_Analog_Data(double * dNormalized, int iNumber, int
iRange, double * dmVData);
Public Sub Licel_Scale_Analog_Data(dNormalized() As Double, iNumber As
Integer, iRange As Integer)

```

### 1.2.7 APD functions

**Licel\_TCPIP\_APDGetStatus** Get the status of the APD with the corresponding APD number

```

int Licel_TCPIP_APDGetStatus(SOCKET s, int APD, bool* ThermoCooler,
bool* TempInRange, int* HV, bool* HVControl)
Public Function Licel_TCPIP_APDGetStatus(ByVal client As TcpClient,
ByVal APD As Integer, ByVal ThermoCooler As Boolean, ByRef TempInRange
As Boolean, ByRef HV As Integer, ByRef HVControl As Boolean) As Integer

```

**Licel\_TCPIP\_APDSetCoolingState** Set the cooling mode of the specified APD

```

int Licel_TCPIP_APDSetCoolingState(SOCKET s, int APD, bool ThermoCooler)
Public Function Licel_TCPIP_APDSetCoolingState(ByVal client As TcpClient,
ByVal APD As Integer, ByVal ThermoCooler As Boolean) As Integer

```

**Licel\_TCPIP\_APDSetGain** Set the applied high voltage (gain) of the specified APD

```

int Licel_TCPIP_APDSetGain(SOCKET s, int APD, int HV)
Public Function Licel_TCPIP_APDSetGain(ByVal client As TcpClient,
ByVal APD As Integer, ByVal HV As Integer) As Integer

```

### 1.2.8 PMT functions

**Licel\_TCPIP\_PMTGetStatus** Get the status of the PMT with the corresponding PMT number

```

int Licel_TCPIP_PMTGetStatus(SOCKET s, int PMT, bool* HVOn, float* HV,
bool* HVControl)
Public Function Licel_TCPIP_PMTGetStatus(ByVal client As TcpClient,
ByVal PMT As Integer, ByRef HVOn As Boolean, ByRef HV As Double,
ByRef HVControl As Boolean) As Integer

```

**Licel\_TCPIP\_PMTSetGain** Set the applied high voltage (gain) of the specified PMT

```

int Licel_TCPIP_PMTSetGain(SOCKET s, int PMT, int HV)
Public Function Licel_TCPIP_PMTSetGain(ByVal client As TcpClient,
ByVal PMT As Integer, ByVal HV As Integer) As Integer

```

### 1.2.9 Timing functions

The old API was designed for a single trigger board, if one Ethernet Controller controls more than one trigger board, each board needs to be addressed separately with a board ID. The first board has the default ID 0, which is addressed by the old API, the additional boards have the ID's 1 and 2.

**Licel\_TCPIP\_SetTriggerMode** Enable/Disable the trigger in and outputs

Old API defaults to **boardID**: 0.

```

int Licel_TCPIP_SetTriggerMode(SOCKET s, bool LaserActive, bool PreTriggerActive,
bool QSwitchActive, bool GatingActive, bool MasterTrigger)
Public Function Licel_TCPIP_SetTriggerMode(ByVal client As TcpClient,
ByVal LaserActive As Boolean, ByVal PretriggerActive As Boolean,

```

```
ByVal QSwitchActive As Boolean, ByVal GatingActive As Boolean,
ByVal MasterTrigger As Boolean) As Integer
```

new API for multiple trigger boards.

```
int Licel_TCPIP_SetTriggerMode(SOCKET s, int boardID, bool LaserActive,
bool PreTriggerActive, bool QSwitchActive, bool GatingActive, bool MasterTrigger)
Public Function Licel_TCPIP_SetTriggerModeN(ByVal client As TcpClient,
ByVal boardID As Integer, ByVal LaserActive As Boolean, ByVal PretriggerActive
As Boolean, ByVal QSwitchActive As Boolean, ByVal GatingActive As Boolean,
ByVal MasterTrigger As Boolean) As Integer
```

**Licel\_TCPIP\_SetTriggerTiming** Set the timing parameter, as for the trigger mode there is the old API, which defaults to `boardID: 0` while the ne API supports multiple trigger boards.

Old API:

```
int Licel_TCPIP_SetTriggerTiming(SOCKET s, long repetitionRate, long Pretrigger,
long PretriggerLength, long QSwitch , long QswitchLength)
Public Function Licel_TCPIP_SetTriggerTiming(ByVal client As TcpClient,
ByVal repetitionRate As Integer, ByVal Pretrigger As Integer,
ByVal PretriggerLength As Integer, ByVal QSwitch As Integer,
ByVal QswitchLength As Integer)
```

New API:

```
int Licel_TCPIP_SetTriggerTiming(SOCKET s, int boardID, long repetitionRate,
long Pretrigger, long PretriggerLength, long QSwitch , long QswitchLength)
Public Function Licel_TCPIP_SetTriggerTiming(ByVal client As TcpClient,
ByVal boardID As Integer, ByVal repetitionRate As Integer, ByVal Pretrigger
As Integer, ByVal PretriggerLength As Integer, ByVal QSwitch As Integer,
ByVal QswitchLength As Integer)
```

### 1.2.10 Security functions

**Licel\_TCPIP\_SetAccessLimited** Activate the access limitation, that means only whitelisted hosts can access the controller and need to verify them self by properly encoding with the connectionPasswd a two 8 byte numbers. Make sure that you called `Licel_TCPIP_SetWhiteList` before, otherwise no host will be authorized to access the controller.

```
int Licel_TCPIP_SetAccessLimited(SOCKET s, char* connectionPasswd, char* passwd);
```

**Licel\_TCPIP\_SetAccessUnLimited** Dectivate the access limitation, that means every hosts can access the controller.

```
int Licel_TCPIP_SetAccessUnLimited(SOCKET s, char* passwd);
```

**Licel\_TCPIP\_SetWhiteList** List hosts that be allowed to to access the controller after `Licel_TCPIP_SetAccessLimited` has been called. One can list three different hosts. Specifying a 255 as the last number activates the whole range, e.g. 10.49.234.255 as host will make all hosts from 10.49.234.1 to 10.49.234.254 whitelisted hosts.

```
int Licel_TCPIP_SetWhiteList(SOCKET s, char* whiteHost1, char* whiteHost2,
char* whiteHost3, char* passwd);
```

### 1.2.11 Power Meter

The power meter uses two data sockets the first socket for command transmission and return values. The second which has port number one above the first for continous data transmission once the data acquisition has been started.

**Licel\_TCPIP.PowerSelectChannel** Select one channel of the power meter.

```
int Licel_TCPIP_PowerSelectChannel(SOCKET s, int Channel );
```

**Licel\_TCPIP\_PowerStart** Start the power meter data acquisition.

```
int Licel_TCPIP_PowerStart(SOCKET s);
```

**Licel\_TCPIP\_PowerTrace** Start a single power meter data acquisition, it will return a raw integer array of ADC readings.

```
int Licel_TCPIP_PowerTrace(SOCKET s, int *readings, int *numReadings);
```

**Licel\_TCPIP\_PowerStop** Stop the power meter data acquisition.

```
int Licel_TCPIP_PowerStop(SOCKET s);
```

**Licel\_TCPIP\_PowerGetData** Get the data from the second socket with the port number which is one above the command socket.

```
int Licel_TCPIP_PowerGetData(SOCKET s, int *milliseconds, double *reading);
```

### 1.2.12 Bore Alignment

The bore alignment detector uses two data sockets the first socket for command transmission and return values. The second which has port number one above the first for continuous data transmission once the data acquisition has been started.

**Licel\_TCPIP\_BoreSetRanges** Set the background and the signal region for the bore alignment sensor.

```
int Licel_TCPIP_BoreSetRanges(SOCKET s, int backgroundStart, int backgroundStop, int signalStart, int signalStop);
```

**Licel\_TCPIP\_BoreSign** Toggle the sign of the counter that is transmitted together with the alignment data. The counter will increment with every cycle, however the sign might toggle. This can be used to make sure that the data evaluated has been measured after a certain point, for instance a drive movement.

```
int Licel_TCPIP_BoreSign(SOCKET s);
```

**Licel\_TCPIP\_BoreStart** Start the alignment sensor data acquisition. The data itself will be transmitted over a second socket (see [BoreGetData](#))

```
int Licel_TCPIP_BoreStart(SOCKET s, int shots, int cycles);
```

**Licel\_TCPIP\_BoreStop** Stop the alignment sensor data acquisition.

```
int Licel_TCPIP_BoreStop(SOCKET s);
```

**Licel\_TCPIP\_BoreGetData** Get the data from the second socket with the port number which is one above the command socket. The data is returned in the [countrates](#) array which should hold 8 doubles.

```
int Licel_TCPIP_BoreGetData(SOCKET s, double *countrates, long int *counter);
```

### 1.2.13 Function arguments

<i>shotNumber</i>	<i>number of shots already acquired. This shotnumber has an offset of 2 as the two initial clearing cycles advance the shotnumber to 2</i>
<i>lastMemory</i>	<i>to which summation memory the last acquisition was added.</i>
<i>acquisitionState</i>	<i>is FALSE when the transient returns from the armed state, and is TRUE, when an acquisition is running</i>
<i>recording</i>	<i>TRUE during acquisition-time, e.g. the ADC or the photon counting is acquiring data. Recording is FALSE during summation and when the TR is waiting for a new trigger.</i>
<i>numberToRead</i>	<i>number of 16 bit wide data points</i>
<i>dataset</i>	<i>which part of the raw information should be transferred from the device to the computer. Use the constants PHOTON LSW MSW, !!! PRxx xx recorder need to read LSW and MSW instead of PHOTON</i>

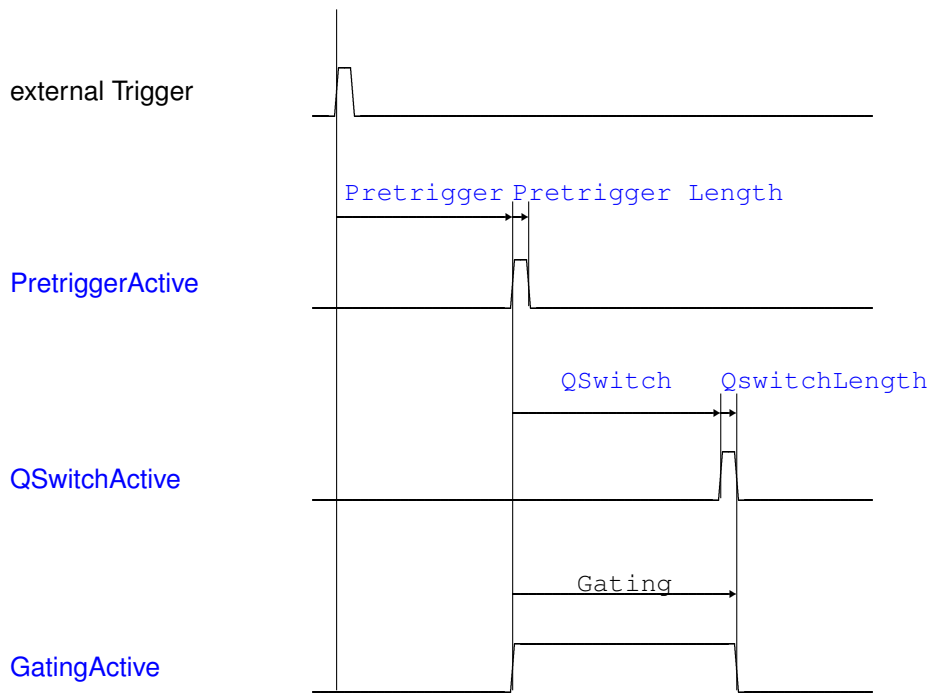
<i>memory</i>	<i>summation memory to be retrieved.</i>						
<i>shots</i>	<i>transfer data every #shots</i>						
<i>TR</i>	<i>Hardware address of the transient recorder Valid values 0:15 all following single device commands for Ethernet controller will access this transient recorder</i>						
<i>TRLlist</i>	<i>List with hardware addresses of transient recorders for subsequent "multiple" commands.</i>						
<i>trNumber</i>	<i>length of the TRLlist.</i>						
<i>iDevice</i>	<i>Hardware address of the transient recorder Valid values 0:7</i>						
<i>s, client</i>	<i>TCPIP socket connection reference</i>						
<i>iDiscrLevel</i>	<i>Photon counting discriminator level Valid values 0:63</i>						
<i>iRange</i>	<i>Analog input range Valid values 0:2, Please use symbolic constants, as defined below.</i> <table border="0" style="margin-left: 40px;"> <tr> <td><i>MILLIVOLT500</i></td> <td><i>0</i></td> </tr> <tr> <td><i>MILLIVOLT100</i></td> <td><i>1</i></td> </tr> <tr> <td><i>MILLIVOLT20</i></td> <td><i>2</i></td> </tr> </table>	<i>MILLIVOLT500</i>	<i>0</i>	<i>MILLIVOLT100</i>	<i>1</i>	<i>MILLIVOLT20</i>	<i>2</i>
<i>MILLIVOLT500</i>	<i>0</i>						
<i>MILLIVOLT100</i>	<i>1</i>						
<i>MILLIVOLT20</i>	<i>2</i>						
<i>iMode</i>	<i>Photon counting threshold mode Valid values 0:1 Please use symbolic constants, as defined below.</i> <table border="0" style="margin-left: 40px;"> <tr> <td><i>THRESHOLD_LOW</i></td> <td><i>1</i></td> </tr> <tr> <td><i>THRESHOLD_HIGH</i></td> <td><i>0</i></td> </tr> </table>	<i>THRESHOLD_LOW</i>	<i>1</i>	<i>THRESHOLD_HIGH</i>	<i>0</i>		
<i>THRESHOLD_LOW</i>	<i>1</i>						
<i>THRESHOLD_HIGH</i>	<i>0</i>						
<i>iCycles</i>	<i>Number of already acquired traces. Please note that there are two cycles used for memory initialization, so that the current shotnumber is iCycle-2 if iCycle &gt;=2. Valid values: 0:4095</i>						
<i>iMemory</i>	<i>Indicates to which memory bank the last shot was added Valid values: 0:1</i>						
<i>iAcq_State</i>	<i>Is 0 if there is an acquisition currently running, and 1 otherwise.</i>						
<i>iRecording</i>	<i>Is 1 if the ADC is acquiring values, when the status command was issued and 0 otherwise.</i>						
<i>imDelay</i>	<i>Delay in milliseconds to wait.</i>						
<i>iDataset</i>	<i>Selects the dataset to be transferred. Valid values: 0:2. Please use symbolic constants, as defined below.</i> <table border="0" style="margin-left: 40px;"> <tr> <td><i>PHOTON</i></td> <td><i>0</i></td> </tr> <tr> <td><i>LSW</i></td> <td><i>1</i></td> </tr> <tr> <td><i>MSW</i></td> <td><i>2</i></td> </tr> </table>	<i>PHOTON</i>	<i>0</i>	<i>LSW</i>	<i>1</i>	<i>MSW</i>	<i>2</i>
<i>PHOTON</i>	<i>0</i>						
<i>LSW</i>	<i>1</i>						
<i>MSW</i>	<i>2</i>						
<i>iNumber</i>	<i>Number of Datapoints. The maximum value depends from the transient recorder type</i> <table border="0" style="margin-left: 40px;"> <tr> <td><i>for TR xx:80</i></td> <td><i>8192</i></td> </tr> <tr> <td><i>for TR xx:160</i></td> <td><i>16380</i></td> </tr> </table>	<i>for TR xx:80</i>	<i>8192</i>	<i>for TR xx:160</i>	<i>16380</i>		
<i>for TR xx:80</i>	<i>8192</i>						
<i>for TR xx:160</i>	<i>16380</i>						
<i>uPortData</i>	<i>array for the datarray.</i>						
<i>i_lsw</i>	<i>array containing the LSW of the analog data</i>						
<i>i_msw</i>	<i>array containing the MSW of the analog data</i>						
<i>lAccumulated</i>	<i>array for the combination of the analog data.</i>						
<i>iClipping</i>	<i>array with the clipping(out of range) information, 1 if there was at least once fulfilled the overange condition, for the specific datapoint, 0 otherwise.</i>						
<i>photon_raw</i>	<i>array containing the rar photon counting data</i>						
<i>photon_c</i>	<i>array with photon counting data without clipping information</i>						
<i>iPurePhoton</i>	<i>If there is no analog there is no need to remove the clipping bit.</i>						

<i>dNormalized</i>	<i>array normalized to the shotnumber.</i>
<i>dmVData</i>	<i>array converted to mV.</i>
<i>sHost</i>	<i>string with the host name</i>
<i>iPort</i>	<i>integer with the host to connect</i>
<i>silent</i>	<i>boolean suppress MsgBox when the connection fails</i>
<i>command</i>	<i>string that will be transfered</i>
<i>response</i>	<i>string containing the response from the controller, the trailing CRLF will be removed</i>
<i>maxlength</i>	<i>max storage capacity of the response string</i>
<i>nTimeoutMillisec</i>	<i>max time to wait for a closing CRLF</i>
<i>Data</i>	<i>byte array to store the data</i>
<i>points</i>	<i>number of bytes to read</i>
<i>nTimeOutMillisec</i>	<i>max time to wait to read the specified amount of data</i>
<i>newPort</i>	<i>new Port after reboot of the controller one should connect to this port</i>
<i>passwd</i>	<i>string containing the current password for the controller</i>
<i>newHost</i>	<i>string The IP address that the controller will be set to</i>
<i>mask</i>	<i>string with the subnet mask that the controller should use for TCPIP communication</i>
<i>gateway</i>	<i>string with the gateway hat should be used by the controller for TCPIP communication</i>
<i>buffer</i>	<i>string to hold the identification information</i>
<i>bufferLength</i>	<i>max capacity of the result string</i>
<i>cap</i>	<i>array with the available capabilities</i>
<i>maxLength</i>	<i>max. capacity of the array</i>
<i>validcap</i>	<i>number of different capabilities</i>
<i>delay</i>	<i>max. time to wait for returning to the idle state in ms</i>
<i>APD</i>	<i>The physical device number of the APD. Valid values are 0-3</i>
<i>TempInRange</i>	<i>is true if the thermocooler is on and the detector temperature is very close to the target temperature</i>
<i>HVControl</i>	<i>is true if remote control of the high voltage is active</i>
<i>HV</i>	<i>the applied HV to the detector</i>
<i>ThermoCooler</i>	<i>true turns thermocooler on, if it is false the detector is only passively cooled</i>
<i>PMT</i>	<i>The physical device number of the PMT. Valid values are 0-7</i>
<i>HVOn</i>	<i>is true is the high voltage is on</i>
<i>remote</i>	<i>is true if remote control is active</i>
<i>boardID</i>	<i>0 for the first timing board (default), 1 and 2 for additional timing boards</i>
<i>LaserActive</i>	<i>if true a trigger for the laser lamp will be generated</i>
<i>PretriggerActive</i>	<i>if true a trigger for the transient recorder will be generated</i>
<i>QSwitchActive</i>	<i>if true a trigger for the laser Q-Switch will be generated</i>

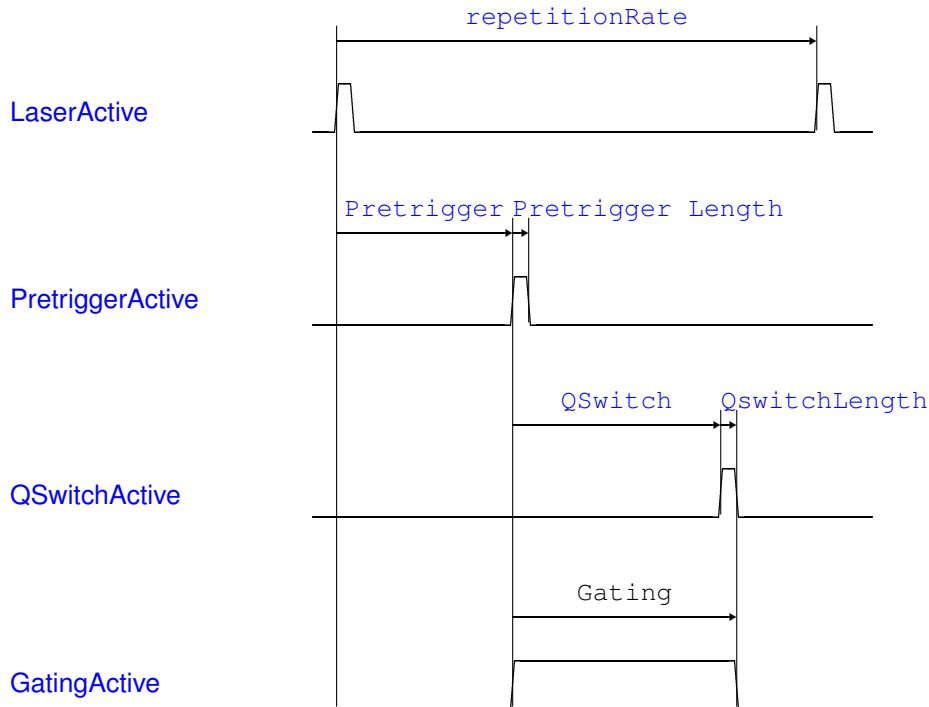
<i>GatingActive</i>	<i>if true a gating pulse will be generated. The gating pulse starts with the raising edge of the pretrigger and ends with the falling edge of the QSwitch Pulse</i>
<i>MasterTrigger</i>	<i>if true an external trigger will be accepted, if false the internal trigger will be used. The internal trigger will be controlled via the repetitionRate</i>
<i>repetitionRate</i>	<i>in internal mode delay between two pulses in ns.</i>
<i>Pretrigger</i>	<i>delay between internal or external trigger and pretrigger in ns</i>
<i>PretriggerLength</i>	<i>length in ns of the pretrigger pulse</i>
<i>QSwitch</i>	<i>delay between pretrigger start and Q-Switch start in ns</i>
<i>QswitchLength</i>	<i>length in ns of the Q-Switch pulse</i>
<i>whiteHost</i>	<i>Host that is allowed to open a connection to the controller in the limited access mode. Specifying a 255 as the last number activates a IP range, e.g. 10.49.234.255 as host will make all hosts from 10.49.234.1 to 10.49.234.254 whitelisted hosts.</i>
<i>connectionPasswd</i>	<i>password used for encrypting the tokens send from the controller initially.</i>
<i>Channel</i>	<i>Power meter detector channel, valid values 0...3.</i>
<i>readings</i>	<i>Power meter raw data for a single trace</i>
<i>numReadings</i>	<i>number of valid data points in the power raw trace</i>
<i>milliSeconds</i>	<i>milliseconds since start</i>
<i>reading</i>	<i>power meter reading</i>
<i>backgroundStart</i>	<i>first background bin</i>
<i>backgroundStop</i>	<i>last background bin</i>
<i>signalStart</i>	<i>first signal bin</i>
<i>signalStop</i>	<i>last signal bin</i>
<i>cycles</i>	<i>number of cycles (data transmissions), -1 for infinite cycles</i>
<i>countrates</i>	<i>array with 8 count rates (4 signal and 4 background)</i>
<i>counter</i>	<i>number of transmitted packages, the sign might be negative</i>

#### 1.2.14 Timing Parameter explanation

**External trigger** MasterTrigger = True



**Internal trigger** MasterTrigger = False



The Laser Lamp pulse has a fixed length of  $5\mu\text{s}$ .

### 1.2.15 Low Level Commands

This applies to the parallel bus communication with the DIO32 cards and the communication of the Ethernet controller itself with the TR. The commands here indicate the low level operating mechanism of the transient recorder.



### 1.3 Memory organization

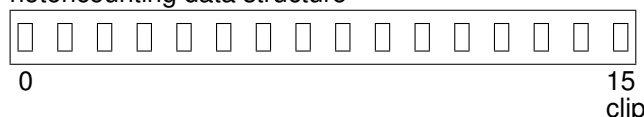
The transient recorder has two separate memories for corresponding to trigger A and B. In each of these memory regions the data is a 40 bit wide vector of accumulated values. The length of these vectors (trace) is defined by the FIFO length and is 8K for the TR-xx-80 and 16k for the TR-xx-160.

The accumulated analog data for each bin is 24 bits wide, the photon counting takes the remaining 16 bits. The photoncounting data can be transferred with one read operation for each bin, the analog data is transferred in two read operations. The analog data has an additional flag indicating that the sum incorporates an overflow value. If for one bin the ADC gives either 0 or 0xFFF the flags is set for the sum and indicates that either an over- or underflow has occurred at this special bin. The sum at these points may not correspond to the physical mean value as the actual ADC value could have been -10 or above 4095. This flag persists when the next trace is added to the previous traces. After accumulating for instance 4094 shots one is able to verify that all mean values do not incorporate out of range values by checking this clip flag. The clip flag is cleared by clearing the memory. The clip flag is transmitted as the 24th bit in the analog dataset or the 15 bit in the photoncounting dataset.

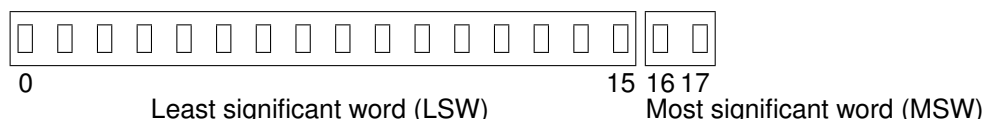
Analog data structure



Photoncounting data structure



**PRxx-xx** recorder differ in the memory layout. The maximum number of counts per bin for a single shot is here 63 which corresponds to 6 bits. Together with 4094 shots the accumulated data can be 18 bits wide. The data is then transferred as for the analog data in a LSW and a MSW dataset. There is no clip bit as it is usefull only for analog acquisitions.



### 1.4 Raw Data to Physical Value Conversion

The [Licel data file format](#) stores the data as raw values and defers the computation of physical values to the display phase.

The conversion starts with a normalization with the shot number. After this step the analog data shows the mean ADC bit values, while the photon counting shows the mean counts per bin per shot (this is the data display used by the Track and Live Display VI's).

The analog data needs then to be scaled by the ADC max value and the input range.

$$phys = norm * \frac{analogRange}{2^{ADCbits} - 1} \quad (1)$$

for a 12 bit ADC and 500mV range this means

$$mVData = norm * \frac{500mV}{4095} \quad (2)$$

The photon counting data can be converted from the counts per bin per shot into MHz if number of bins per  $\mu s$  is given as:

$$MHzData = norm * \frac{bins}{\mu s} \quad (3)$$

If for instance the counts per bin per shots are 1.5 and the number of bins per  $\mu s$  is 20, this would correspond to 30MHz. The transient recorder units share the clock between the ADC and photon counting so the number of bins per microsecond and the sampling rate are equal.

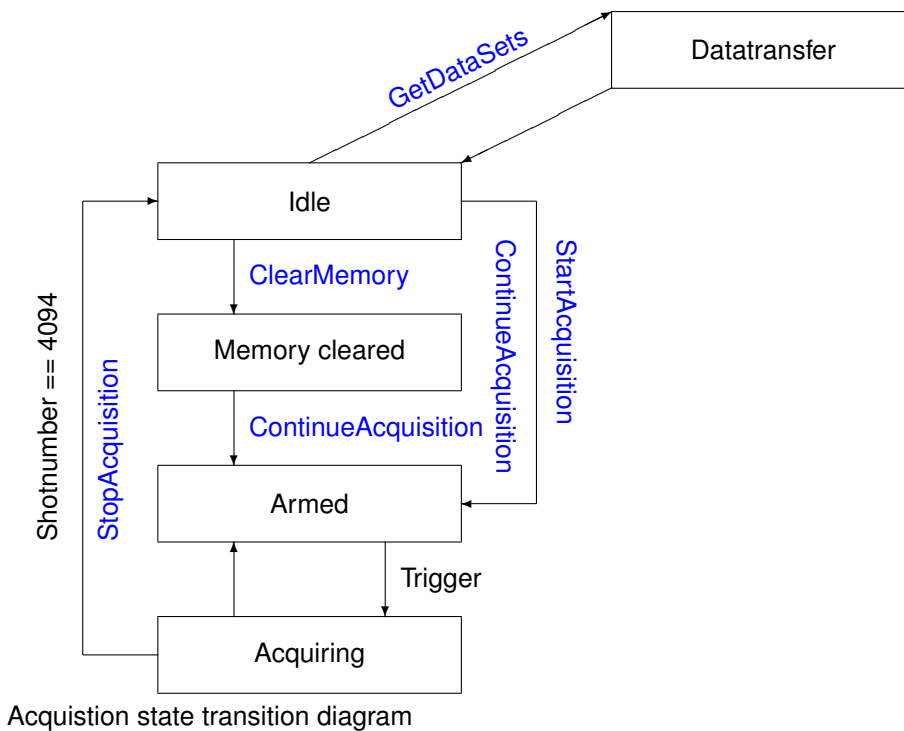
### 1.5 Acquisition Low Level Description

Once the transient recorder is started it will wait for a trigger pulse on either trigger input A or B. The starting consists of two operations which can also be executed separately: clearing the memory and arming the transient recorder. While arming the transient recorder is nearly instantaneous, clearing the memory requires the time a acquisition would take for memory A and B. For a TR20-160 this would be 6ms ( $2 \times 3ms$ ).

Once the TR is armed it waits for next trigger and once it arrives the input where it arrives which summation memory will be used. This will continue till the shotnumber reaches 4094 or a stop command is send. The stop command will not be executed immedately but rather tells the system to return after the next acquisition not to the armed state but to the idle state.

Once the system is in the idle state the acquired data can be transferred to the PC. There is no acquisition possible in paralell with datatransfer. So this will cause another time when the transient recorder cannot average. The typical transfer time for DIO-32HS based system is 20ms for a single dataset with 16k. For a Ethernet based system the ethernet controller will transmit 200 Kbytes/second. A single dataset with 16k will then take  $32/200 = 160ms$ .

Once all the data is transfered the transient recorder can be started again.



**Shot number considerations** The reported shot number is zero after the reset. After the memory has been cleared the TR reports a shot number of two, after the first acquisition a shot number of 3 is reported and so till 4096 is reported. At this stage the above mentioned 4094 shots are acquired and the TR will stop.

To stop the TR at a predefined shot number below this one should wait till the desired `shot number + 1` is reached and issue a stop command. If a trigger is still supplied the TR will return from the armed state with the next trigger and the reported shot number will be `shot number + 2`, which has the two additional cycles from the memory clear and the desired shot number.

#### Calculation of lost shots

1. Stratospheric system, lets assume a system with 4 channels TR20-160, both analog and photon counting data needs to be transferred for the whole range(120km). The Laser frequency is 30Hz. The system will

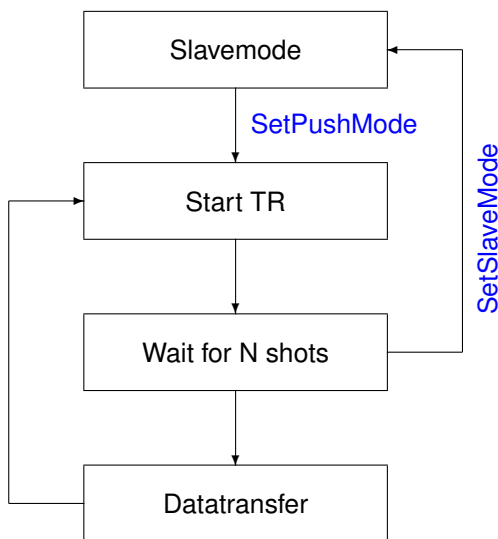
need 6ms for initializing all 4 TR, as this can be done in parallel. Then it will acquire 4094 shots and all data is to be transferred to the PC. So one has the LSW and the MSW for the Analog and the PHOTON dataset for the photon counting. That's 60ms per transient recorder and 240ms overall for the system. So one would lose 250ms for reading and restarting or 7 to 8 shots per 4094 shots.

For a tropospheric system this would be 96k per transient recorder and 400k for the whole system this would require 2sec and one will lose 60 shots.

2. Tropospheric System, assume 1 channel TR20-160, where only the first 1000 bins of analog data are transferred (a 7.5km trace). Then even with the Ethernet controller the dataset would be transmitted within 10ms to the PC, so no shot would be lost.

Please note that these times assume that the timing constraint comes from the transient recorder or the Ethernet controller. In real systems the PC is also limiting factor. For instance the start of the data transfer requires to start a DMA process which consumes also time down to a couple of ms or front panel activity like displaying the data may require significant time.

**Push Mode** For Ethernet based systems sending a lot of small commands can cause delays due to the Nagle Algorithm (see [http://en.wikipedia.org/wiki/Nagle's\\_algorithm](http://en.wikipedia.org/wiki/Nagle's_algorithm) for the details). To overcome this inside the Ethernet controller a mechanism is implemented which waits for a predefined shot number to be acquired, sends the data (push) without further request to the PC and restarts the transient recorder. This will continue till the push mode is revoked. The controller returns then into the normal mode (slave). This transmission scheme is especially useful for analog datasets. As long as the shot number is below 16 the accumulated analog data will not exceed 16bits (12 bit ADC + 4 bits for averaging) so only the LSW needs to be transferred.



Push Mode state transition diagram

## 1.6 Network security

The Licel Ethernet Controller might be the target of an attack. The best protection against this is to run the controller with a private IP address beyond a firewall. Firewalls are designed to protect against various types of attacks that can not be covered by the ethernet controller. Licel strongly recommends the use of a firewall/router combination to prevent unauthorized use of the hardware.

Starting with firmware versions from 2005-02-22 (*state53*) the Licel Ethernet Controller has an additional level of security that can be additionally used.

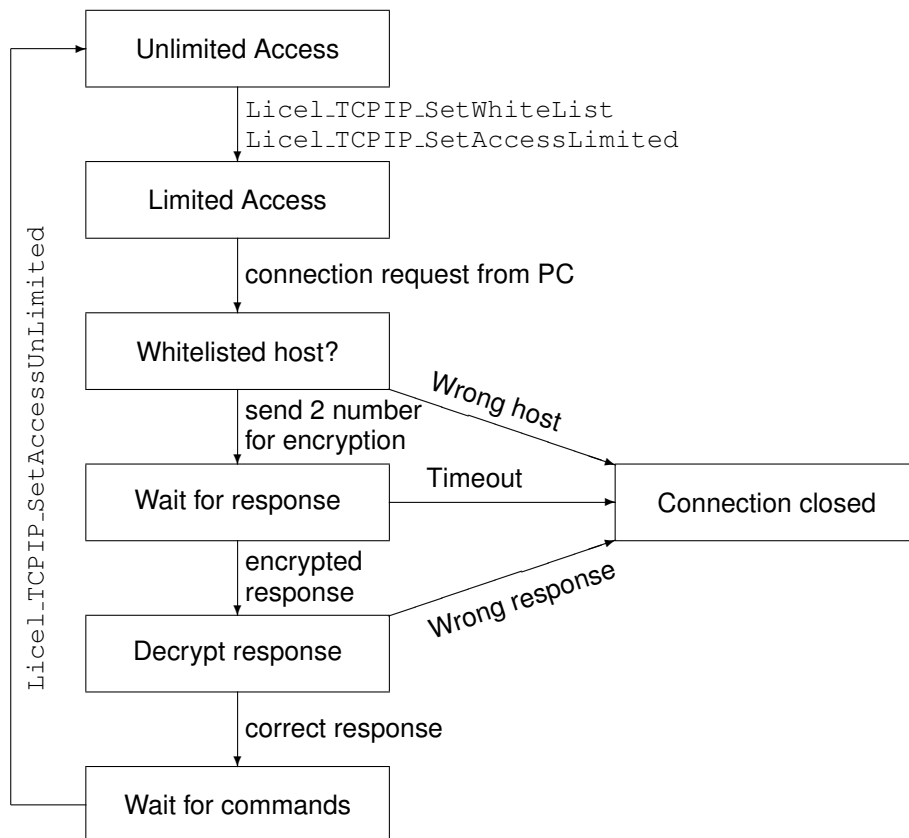
A secure mode combines white listing of allowed hosts with an encrypted password transmission scheme. In order to activate the secure mode,

- one needs to transmit the white listed hosts to the controller, send a connection password when activating the secure mode.
- Once this is done the controller will check whether the host is authorized to access the controller and

- then send a token that the PC needs to encrypt with the connection password and
- send the encrypted response back.
- The controller then decrypts the encrypted token with the previously received connection password and compares it to what was sent for encryption.
- If the response is correct the connection is allowed to proceed otherwise the connection is closed.

The idea behind this is that only once in a secure environment the connection password is transmitted to the controller and later the password is used to encrypt and decrypt a random token.

The algorithm for encryption/decryption is a blowfish algorithm which is a open algorithm without license restrictions. See [Bruce Schneiers Page](#) for the details.



Secure Mode - State transition diagram

The setting of the secure mode will persist during power off and on.

Per default the secure mode is disabled and all hosts can access the controller. The hardware reset will also reset the secure mode and remove all white listed hosts. If during the secure mode activation something goes wrong, like the controlling PC is not white listed, then the only way to get again access to the controller is a hardware reset. Due to this it is highly recommended that the secure mode is only enabled when one has physical access to the controller.

## 2 Installation

### 2.1 Ethernet software

#### 2.1.1 Required software

In order to use the C,Basic or LabView routines under Windows or Linux you will need a working installation of

- Visual C++ 6.0 or gcc(Linux)

- Visual Basic 6.0 .
- LabView 6.x or 7.x

The controlling PC should have a working network connection. The details of the of setting up the Ethernet controller are outlined in the [Ethernet Manual](#).

## 2.2 Windows - DIO-32HS

### 2.2.1 Required software

In order to use the C,Basic or LabVIEW routines under windows you will need a working installation of

- Visual C++ 6.0 or
- Visual Basic 6.0 .
- LabVIEW 7.x or higher

### 2.2.2 NI-DAQ-Setup

- Check first whether you find `Measurement and Automation Explorer` in your National Instruments program group or a `NIDAQ configuration Utility` in your LabVIEW program group.
- If present start them and look for the NIDAQ-Version. We found that the most reliable version are 5.1.1, 6.5.1 and 6.9.3
- The NIDAQ-Versions 6.0 - 6.1 do not work properly with the interface card, so please use 5.1.1 or a higher version if you have these versions already on your PC.
- There have been reports about crashes with NI-DAQ 6.8.x under Win98.
- If not present install the NIDAQ-Software, preferable 6.9.3, as indicated in the NIDAQ Manual.
- In order to compile the corresponding projects the language support for Visual C or Visual Basic should be installed.
- Once the language support is installed there should be a `nidaq.h` in `Program\National Instruments\Ni-daq\Include\` and a `nidaq32.lib` in `Program\National Instruments\Ni-daq\Include\` for a working C installation.
- For Visual Basic you should be able to locate `nidaq32.bas` in the `Program\National Instruments\Ni-daq\Include\` directory.

### 2.2.3 Interface card installation

The DIO-32HS cards are Plug and Play enabled cards. Windows9x detects the plug and play cards during the boot process. There are special considerations with ISA PNP boards under Windows NT.

**Configuring ISA Plug and Play Devices for Windows NT 4.0** If you plan to use ISA Plug and Play DAQ devices on Windows NT 4.0, you must first install the Windows NT 4.0 ISA Plug and Play driver before configuring your device with the NI-DAQ Configuration Utility. This driver is not installed by default. Follow these steps to install the driver:

1. Insert your Windows NT 4.0 CD.
2. Go the `\Drvlib\Pnpisa\X86` directory.
3. Right-click once on the `Pnpisa.inf` file, select the Install option, and follow the instructions.
4. After you have installed the `Pnpisa.inf` file, shut down your computer.
5. Install your ISA Plug and Play DAQ device.
6. Turn on your computer. When Windows NT 4.0 detects your ISA Plug and Play DAQ device, it will specify the necessary driver files. Because this will result in a configuration change, restart your computer.
7. After you have restarted your computer, run the NI-DAQ Configuration Utility to configure your device.

## Configuring PCI Boards

1. PCI boards will be detected during the boot process.
2. Turn the computer off
3. Insert the card and boot the machine
4. The card will be detected during the boot process.
5. The following resources are necessary
  - I/O Ports
  - one Interrupt
  - one DMA Channel
6. The configuration can be changed under the Windows9x in the system manager and under Windows NT with the NIDAQ Configuration utility.

### 2.2.4 Verification

Assuming that you have managed to bring up the machine again without any dirty messages about resource conflicts ..., you have to test whether the NIDAQ software recognizes the DIO-32HS.

Remove all cables from the DIO-32HS.

Open the Measurement and Automation Explorer. Press the Testpanel button. If you have resource conflicts you should go back to the hardware manager and change the configuration there, reboot and repeat the verification. If your configuration is valid, select the same port for input and output. After setting some lines in the output to TRUE you should see the same lines true in the input port, because the card reads back the output lines. If it does not work you have to change the configuration again and repeat the procedure above.

## 2.3 Linux - PCI-DIO-32HS

### 2.3.1 Kernel preparation

The COMEDI driver is compiled versus the kernel sources. In order to load the modules afterward properly, the kernel sources should correspond to the booted kernel. The best way to ensure this is to compile a new kernel and make it bootable. The methods to do this are behind the scope of this manual and are outlined either in the [Kernel HOWTO](#) or in the vendor documentation of your distribution. Please note that every time you update the kernel you will need to recompile comedi, comedilib and all applications.

### 2.3.2 Necessary Files

The modified COMEDI versions are distributed as two archives comedi.tgz and comedilib.tgz. Please create first a directory for instance `com` and copy both archives there and unpack them by

- `tar -xzvf comedi.tgz`
- `tar -xzvf comedilib.tgz`

The following files have been modified with respect to the original version.

- `comedi/comedi/comedi_fops.c`
- `comedi/comedi/kvmmem.h`
- `comedi/comedi/drivers/ni_pcidiio.c`
- `comedi/comedi/include/linux/comedi.h`
- `comedilib/include/comedi.h`
- `comedilib/include/comedilib.h`
- `comedilib/lib/Makefile`
- `comedilib/lib/dio.c`
- `comedilib/lib/dio_licel.c`

### 2.3.3 Driver tests and card installation

Please follow the instructions outlined at `comedi/INSTALL` and `comedilib/INSTALL` to install both packages.

Once you have installed both packages and modified `/etc/modules.conf` you should be able to run `/sbin/modprobe ni_pcidio` and it should not generate an error.

Shutdown the computer and plugin the PCI-DIO-32HS and reboot the machine.

Run again `/sbin/modprobe ni_pcidio` and `/usr/sbin/comedi_config /dev/comedi0 ni_pcidio`. It should not generate an error, please check also `/var/log/messages`

### 2.3.4 Changes to `/etc/modules.conf`

The following to lines should be added to `/etc/modules.conf`:

```
alias char-major-98 comedi
alias char-major-98-0 ni_pcidio
```

### 2.3.5 Changes to `/etc/rc.d/rc.local`

The following to lines start the module at every bootup. Otherwise these commands should be issued by the superuser:

```
/sbin/modprobe ni_pcidio
/usr/sbin/comedi_config /dev/comedi0 ni_pcidio
```

### 2.3.6 Directory structure

The comedi driver is placed below the `com2` directory. The C-Sources for the examples are below `licellinux`. The LabView Libraries are in the `licellinux/labview/`. The code for the interface between LabView and comedi is in the `licellinux/labview/lib` folder.

### 2.3.7 LabView + comedi

Recompiling comedilib may require a recompilation of the glue code. This can be done by

- `make clean`
- `make`

in the `licellinux/labview/lib/config`, `..read` and `..write` directories. Once the CIN code resources are rebuild they should be reloaded into the corresponding CIN's in `licellinux/labview/LV_COMEDI_Interface.llb`. In the following VI's a code resource reload would be necessary:

- `lv_config.vi`
- `lv_read.vi`
- `lv_write.vi`

## 2.4 Software installation

1. Create a subdirectory for the transient recorder software
2. Copy the libraries from the CDROM containing LabVIEW llb-files.
3. Create a subdirectory for the C-Software and copy the files from the C-Sources directory.
4. Change the properties of the LLB-files from write-protected to the unprotected state.

5. Open LabVIEW and select the item `mass compile` from the `File`-menu. Compile the `HS-Track.llb` and `HS-Acquis.llb`. Compiling errors in the vi's indicate a damage of the libraries. Please download a new version of the files from our [ftp-server](#) if you encounter any problems.

## 3 C - Example Programs

This section contains 4 sample program which could be used in a batch file. They demonstrate the basic actions to run the transient recorder. `start.c` configures the transient recorders for an acquisition and starts them. `shot.c` shows the number of shots already acquired. `read_out.c` transfers the data from the transient recorders to the PC and writes them to a data file. `Show_tmp.c` displays the data. The example programs can be compiled by

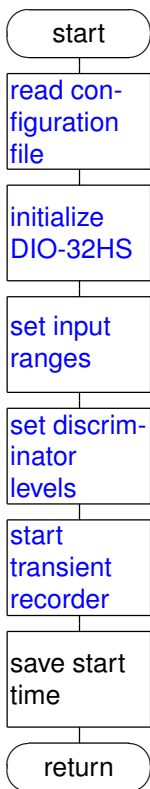
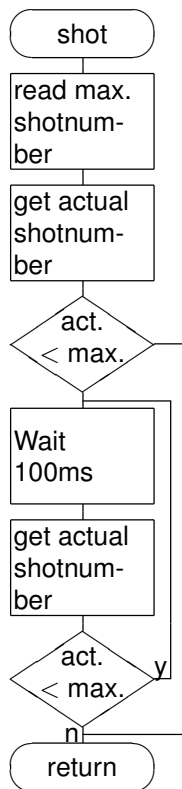
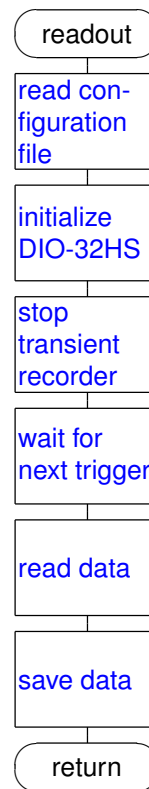
- `make -f start.mk`
- `make -f shot.mk`
- `make -f read_out.mk`
- `make -f show_tmp.mk`

Under Windows you should open the corresponding Visual C Projects. For the DIO32HS version adjust the include path and the path to `nidaq32.lib`. Please note that currently there is no routine for `show_tmp` under WinXX.

### 3.1 The script

```
./start
while true
./shot
./readout
./start
./show_tmp
done;
```

It first starts all transient recorders, then it waits until 4094 shots (the default) are acquired. Then the data is transferred and the transient recorder are restarted for the next acquisition. While the data is acquired the previously acquired data is shown to the user.

**3.2 start****3.3 shot****3.4 readout****3.5 example program configuration file**

The examples use a ASCII based configuration file `standard.cfg`. The information is done in two groups first the measurement situation and then once the number of datasets is specified, the configuration info for each data set is given.

<i>char [8]</i>	<i>measurement site integer altitude above sea level [m]</i>
<i>double</i>	<i>longitude</i>
<i>double</i>	<i>latitude</i>
<i>char</i>	<i>leading letter of filename</i>
<i>char [250]</i>	<i>output directory for data, must be identical with input directory in mega. cfg</i>
<i>bool</i>	<i>does the laser shoot while reading, i.e. is there a trigger pulse while reading (no - 0, yes - 1) : range (0-1)</i>
<i>integer</i>	<i>identification number of the National Instruments board;</i>
<i>integer</i>	<i>number of data sets</i>
<i>integer</i>	<i>Transient Recorder number (range 0-15)</i>
<i>string</i>	<i>Transient Recorder type in the format TRXX-YY, where XX is the sample rate and YY is the memory length</i>
<i>integer</i>	<i>memory MEMORY_A - 0, MEMORY_B - 1 (range 0-1)</i>
<i>integer</i>	<i>signal type: analog - 0, photon counting - 1 (range 0-1)</i>
<i>integer</i>	<i>bins number of data bins in the data file, these bins may incorporate more than original transient recorder bins, if the data reduction below is larger than 0.</i>

<i>integer</i>	<i>signal range</i> <i>analog signal type: 500mV - 0, 100mv - 1,20mv -2 (range 0-2)</i> <i>photon counting signal type: discriminator level (range 0-63) 63-1.25V</i>
<i>integer</i>	<i>show overflow values</i>
<i>integer</i>	<i>voltage at photomultiplier [V]</i>
<i>double</i>	<i>laser frequency [Hz]</i>
<i>double</i>	<i>number of bins</i>
<i>integer</i>	<i>data reduction factor (<math>2^n</math>), this for data reduction of 2 for instance <math>2^2 = 4</math> transient recorder bins will be combined into a single data bin.</i>
<i>integer</i>	<i>polarization none -0, parallel -1, perpendicular -2 double wavelength [nm]</i>
<i>integer</i>	<i>laser source identifier</i>

### 3.6 File format

The example program use the same file form at as the LabVIEW `TCPIP-Acquis.llb`. By this also the files are inter operable between the different platforms. The file format is a mixed ASCII-binary format where the first lines describe the measurement situation, below follow the dataset description and then raw data as 32-bit integer values itself.

#### Sample file header

```
a9981017.204567
Berlin 10/08/1999 17:20:36 10/08/1999 17:20:41 0015 0015.0 0053.0 00
0000000 0010 0002000 0005 02
1 0 2 08000 1 1600 07.5 286.0 0 0 00 000 12 002000 0.100 BT1
1 1 2 08000 1 1600 07.5 286.0 0 0 00 000 00 002000 0.793 BC1
```

#### Line 1

*Filename*                    *string a.*  
*Format: ?yyMddhh.mmssmsms*

- ? - The first letter can be choosen freely.*
- yy - two numbers showing the years in the century*
- M - one number containing the month as a hexadecimal number*  
*(December  $\equiv$  c)*
- dd - two numbers containing the day of month*
- hh - two numbers containing the hours since midnight*
- mm - two numbers containing the minutes*
- s - two number containing the seconds*
- ms - two number containing the milliseconds divided by ten.*

#### Line 2

<i>Location</i>	<i>String with 8 Letters</i>
<i>Start Time</i>	<i>dd/mm/yyyy hh:mm:ss</i>
<i>Stop Time</i>	<i>dd/mm/yyyy hh:mm:ss</i>
<i>Hight asl.</i>	<i>four digits (meter)</i>
<i>Longitude</i>	<i>four digits (including - sign). one digit for decimal grades.</i>
<i>Latitude</i>	<i>four digits (including - sign). one digit for decimal grades.</i>
<i>zenith angle</i>	<i>two digits in degrees</i>

**Line 3**

<i>Laser 1 Number of shots</i>	<i>integer 7 digits</i>
<i>Pulse repetition frequency for Laser 1</i>	<i>integer 5 digits</i>
<i>Laser 2 Number of shots</i>	<i>integer 7 digits</i>
<i>Pulse repetition frequency for Laser 2</i>	<i>integer 5 digits</i>
<i>number of datasets in the file</i>	<i>integer 2 digits</i>

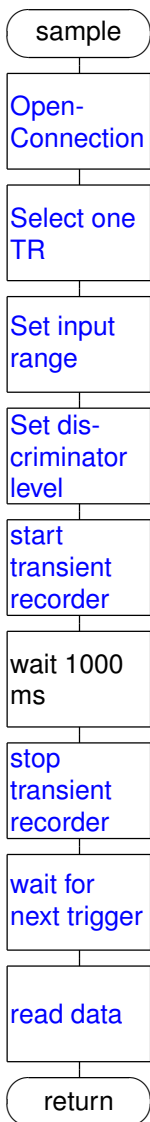
**Dataset description**

<i>Active</i>	<i>1 if dataset is present, 0 otherwise</i>
<i>Analog/Photoncounting</i>	<i>Analog <math>\equiv</math> 0, Photoncounting <math>\equiv</math> 1</i>
<i>Laser source</i>	<i>one digit Laser 1 <math>\equiv</math> 1, Laser 2 <math>\equiv</math> 2.</i>
<i>Number of bins</i>	<i>5 digits</i>
<i>1</i>	
<i>PMT highvoltage</i>	<i>four digits in Volt</i>
<i>binwidth</i>	<i>in meter two digits before . and 2 digits after the dot</i>
<i>Laser wavelength</i>	<i>in nm, three digits dot</i>
<i>Polarisation</i>	<i>one letter, o <math>\equiv</math> no polarisation, s <math>\equiv</math> perpendicular, l <math>\equiv</math> parallel</i>
<i>0 0 00 000</i>	<i>backward compatibility</i>
<i>number of ADC bits</i>	<i>in case of an analog dataset, otherwise 0</i>
<i>number of shots</i>	<i>6 digits</i>
<i>analog input range/discriminator level</i>	
	<i>analog input range in Volt in case of analog dataset , discriminator level in case of photon counting, one digit dot 3 digits.</i>
<i>Dataset descriptor</i>	<i>BT <math>\equiv</math> analog dataset, BC <math>\equiv</math> photon counting, the number is the transient recorder number as a hexadecimal.</i>

The data set description is followed by an extra CRLF. The datasets are 32bit integer values. Datasets are separated by CRLF. The last dataset is followed by a CRLF. These CRLF are used as markers and can be used as check points for file integrity.

### **3.7 SampleAcquis for Ethernet Applications**

This is a skeleton for a simple acquisition with a Ethernet system.



## 3.8 Network management utilities

### 3.8.1 Getting Started

This module shows the basic process of connecting to a Licel Ethernet controller. The controller will return the ID and reveal the capabilities that can be used with this controller.

### 3.8.2 Set Fixed IP Address

This module sets the controller IP address. The new address will be activated after the controller is turned off and on again

### 3.8.3 Activate DHCP Mode

This module activates the DHCP mode. The controller will enter DHCP mode once it is turned off and on again

### 3.8.4 SecureModeEnable

This module enables the secure mode for accessing the Ethernet controller. See the [Network Security](#) section for more details

### 3.8.5 SecureModeDisable

This module disables the secure mode for accessing the Ethernet controller. The controller will be fully accessible for all hosts. See the [Network Security](#) section for more details

## 4 Appendix VB6/VB.net Programming

The Visual Basic driver is not part of our standard distribution and can be ordered from Licel separately. Licel recommends the use of the LabVIEW Modules, which allows a much faster programming and deployment of your applications. The VB.net driver should be used when existing applications are upgraded to communicate with the Licel Ethernet Controller. The VB6 programming is provided for legacy programming when the upgrade to VB.Net is not possible. The programs require the Installation of SocketWrench. The installer is provided as a zip-file in the VB6 directory. The VB6 and the VB.net use a control that ships with LabVIEW to display the data. If you do not LabVIEW you will need to replace the control first before running the applications.

### 4.1 Sample applications

For demonstration purposes some demo modules are available. The main purpose is to show the use of the functions provided in the driver, which are described after the modules.

#### 4.1.1 Control Overview

This module demonstrates the use of the different commands for controlling a single transient recorder, PMT's, APD's and the trigger module. When the module is started and the connection is established the capabilities of the controller are queried. For each supported capability a tab will become available

#### 4.1.2 MultipleChannel

This module demonstrates the use of commands to control multiple transient recorder with single commands. This might be necessary for performance reasons as sending a lot of small commands and waiting every time for the response can yield large delays as the OS tries to delay the sending of TCPIP packages that are almost empty. The starting point is the `Start_Click` function. It opens a connection to the controller and starts the selected TR's then it launches a timer which will every second readout the TR's. If the selected TR's have reached 4094 shots the memory will be cleared and the acquisition will be restarted.

#### 4.1.3 PushModeDemo

This module demonstrates the use of the push mode for fast data transfer The push mode can be use when repetitively data needs to be transfered between the TR and the PC. If the shot number is low this would require a lot of calls to `StartAcquisition`, `GetStatus` and `GetDataSet` In the push mode one sends only one command at the start (see the `Start_Click` routine). Then a timer is started and the data is read as it arrives. Please note that the data is transferred over a second socket connection.