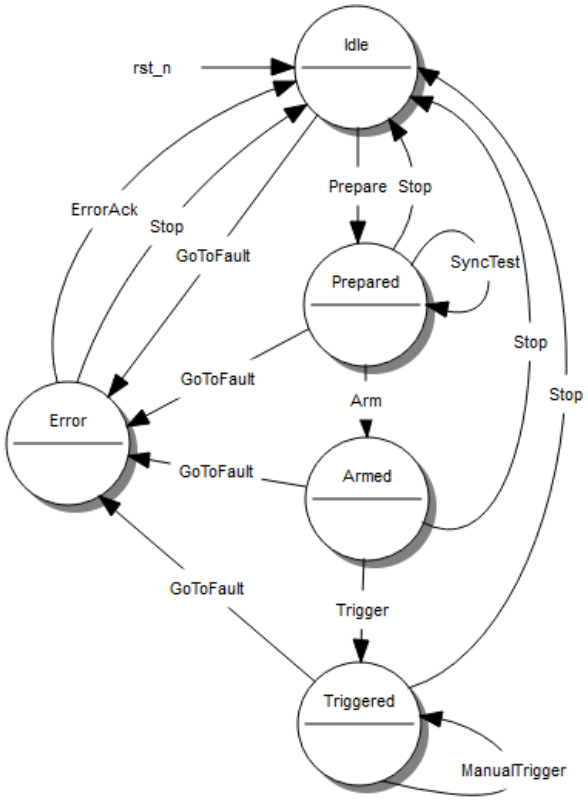
Laser control description

- Laser control is centered around event-driven finite-state machine (FSM).
- FSM is application that runs on LazServ board which is integrated into laser system.
- FSM itself gets commands by remote commands through LAN or RS232 interface (not covered in this manual).

State machine

- State machine representation:



- To describe the behavior of FSM states and transitions are used:
- States:
  - Only one state is allowed at any given time.
  - The FSM can change from one state to another in response to remote command.
  - State itself is a variable, that can be read at any time, but can't be written.
- Transitions
  - Only one transition is allowed at any given time.
  - Events are posted in to a queue while FSM is in transition.
  - Successful transition will lead to planned state, unsuccessful may leave the state unchanged or generate GoToFault event.
  - Transition is composed as linear sequence of commands written in simple script language. Sequences are listed in SEQUENCES table.
  - How long transition lasts is not limited, external application may poll current FSM status.

State machine implementation

- State machine is implemented as executable Linux application.
- Application handles FSM control and communication.
- FSM application is written in LUA.

Communication services

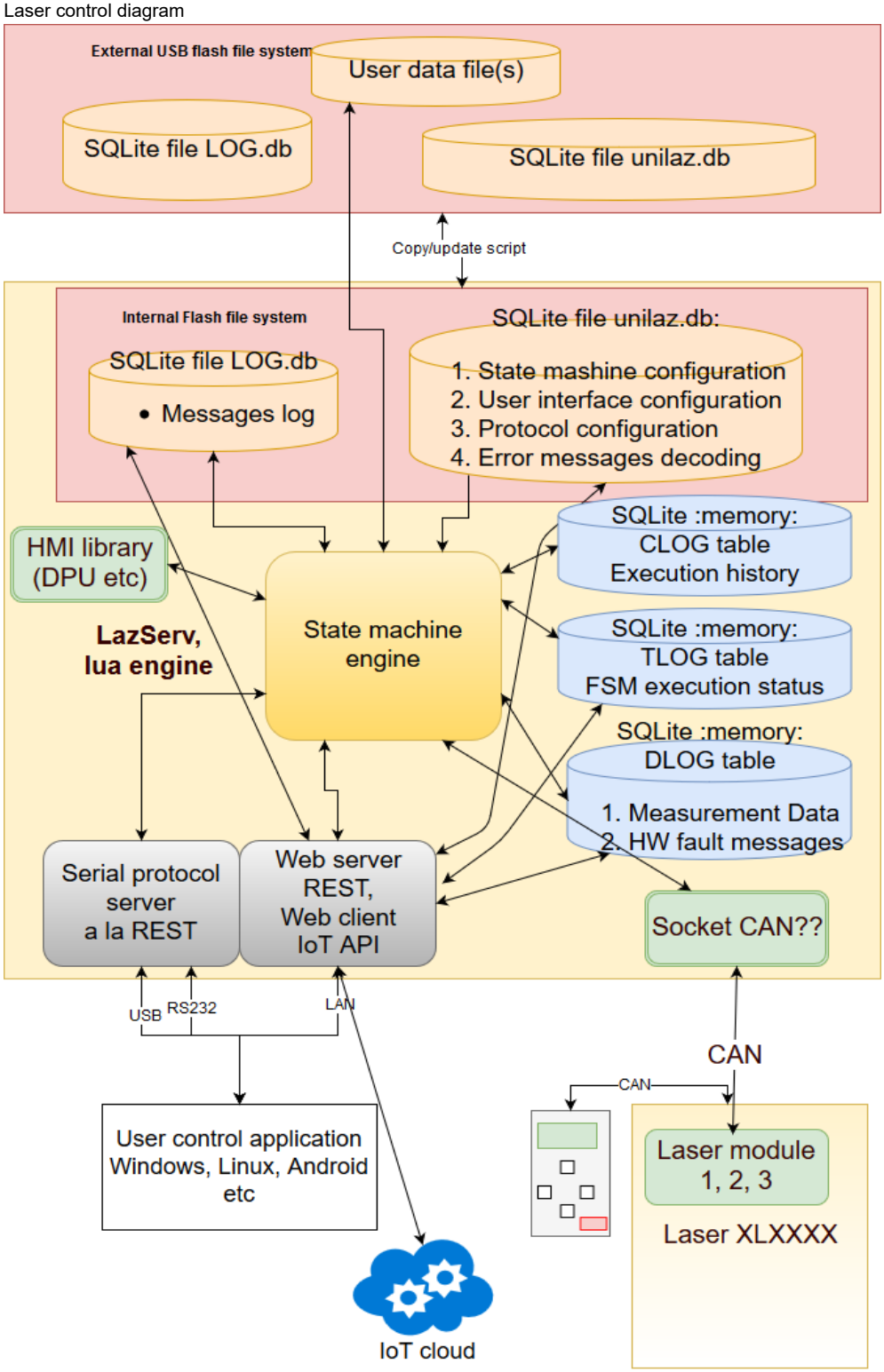
- To handle communication between FSM and remote controller HTTP server is run on 8081 port in controlling board.
- REST API over TCP/IP allows to control FSM remotely.
- REST commands and responses can be entered/viewed by ordinary browser.
- In addition to REST server on controlling PC, additional servers are run on 80 and 8080 ports of the communication controller. Http server on 80

is used for communication setup. Point your browser to IP http://yyy.yyy.yyy.yyy or http://DeviceSerialNr ,e.g http://nll429 to see setup page. See documentation for details.

- Http server on 8080 supports REST API and is used for communication directly to individual system modules. See document "REST API over LAN or WLAN protocol description" for details.
- 80, 8080 servers are using different protocols that are not covered in this manual.
- FSM talks to the laser hardware either through COM port, virtual or genuine, either trough LAN/WLAN.

Laser control diagram

- Laser control diagram



Essential blocks

1. State machine engine. Infinite cycle is employed: receive commands - process events - execute sequences - post results.
2. Web server/REST. Processes REST GET requests and composes return data. All communication between user application and laser FSM go through this server.
3. Configuration database, SQLite file unilaz.db.
4. Session log database, SQLite file log.db. Logs error messages in normal model and all steps in debug mode.
5. Execution status database, SQLite memory file, TLOG table. Holds status updates about execution status of sequences.
6. Data database, SQLite memory file, DLOG table.

- SQLite files are accessible through specialized queries.

Remote/user interface requests:

- FSM starts from executing **Init** sequence. After **Init** sequence is successfully finished, FSM enters wait for event mode.
- Events are generated by remote commands.
- Request/command URL: protocol://host/REST/HTTP_CMD/query
  - Both requests and commands are using http GET protocol, therefore internet browser is sufficient to test communication and control.
  - **protocol**: http
  - **host**: xxx.xxx.xxx.xxx:8081.
  - **REST/HTTP_CMD** - FSM identification string
  - **query** - command or parameter request

query format

- **query** starts with question symbol. In description below '?' is omitted.
- '?' is followed by query o command word and parameters separated by '/' symbol.
- There:
  - **query** - request for execution status, variables or sensor data. Query does not change FSM status.
  - **command** - instructions for FSM that will alter the state.
- Using of commands/queries:

LIST query

- Returns SELECT query results from SQLite database files unilaz.db and log.db.
- The most obvious use of **LIST** command is to read execution message log or to read coded mesages list.
- LIST command use:
  - LIST/XXXX[/YYYY]
  - This will result query to database: **SELECT YYYY FROM XXXX**
  - YYYY are column names separated by commas. YYYY will be replaced by * in case no parameter supplied resulting all columns listed. DISTINCT statement can be used there.
- XXXX should start from the table name, but is not limited to it. GROUP BY, ORDER BY, LIMIT statements can by used. Consult SQLite language manual for details https://sqlite.org/lang_select.html
- Table names:
  - **SEQUENCES** - FSM command sequences
  - **MSG** - coded message decoding table, fields:
    - **ERROR** type INTEGER - message code that application responds to all queries and commands.
    - **ID** type INTEGER - additional code, used to decode hardware faults.
    - **FUNCTION** type STRING, process, owning this message, name.
    - **FSTRING** type TEXT, message body.
    - **COMMENT** type TEXT, additional info, for example, what to do next.
  - **CLOG** - session execution log.
    - **TIME** type REAL, time when message was recorded in seconds lapsed from, 1900 01 01
    - **STEP** type INTEGER, SEQUENCES
    - **FAULT** type INTEGER,
    - **RESULT** type TEXT,
    - **SRC** type TEXT
  - **COM** - protocol response format strings.
- query examples:
  - http://192.168.0.160:8081/REST/HTTP_CMD/?LIST/MSG - coded message list
  - http://192.168.0.160:8081/REST/HTTP_CMD/?LIST/SEQUENCES/DISTINCT SEQUENCE to see all available FSM commands.
  - http://192.168.0.160:8081/REST/HTTP_CMD/?LIST/CLOG to get execution log
  - http://192.168.0.160:8081/REST/HTTP_CMD/?LIST/CLOG WHERE TIME > 3574141740.945 to get execution log from some time

moment.

DATA query.

- Returns SELECT query results from data table SQLite. Data table holds measurement readings from sensors and reports about register value changes. Reports about 'Error Code' register changes are posted to this table.
- Data table is organized in to three columns:
  - **TIME** - data timestamp. This is local timer readings, integer number. One count corresponds to 1us. Timestamp is system depending without meaningful absolute value.
  - **CHAN** - registration channel. Data from different sources are organized in to separate channels. Channel number is assigned during initialization by command **logstart**, see VALUE column of SEQUENCES table

IND	SEQUENCE	COMMAND	ADDRESS	REGISTER	VALUE
11	Init	logstart	SY3PL50M:32	Error Code	1
12	Init	logstart	LDD1A:18	Error Code	2
13	Init	logstart	PHD1K000:3	Data	3
14	Init	logstart	PHD1K000:4	Data	4

  - **DATA** - actual readings in double float format.
- Command format is DATA/ChannelNo[/FromTime]
- Query examples:
  - http://192.168.0.160:8081/REST/HTTP_CMD/?DATA/2 - get all data from channel No 2
  - http://192.168.0.160:8081/REST/HTTP_CMD/?DATA/2/1458866649 data from channel No 2, where TIME > 1458866649
- Data table keeps 5000 records totally. At max registration rate, for example, 5 sensors x 1000Hz frequency buffer keeps data max one second old. Data should be queried more often than max data age in order to maintain consistency.
- Application software that needs consistent data set from the channel should use DATA/ChannelNo query at first. Later **DATA/ChannelNo /FromTime** query should be cycled. There **FromTime** is timestamp from the latest received data record.

EXE command

- Posts FSM command to the execution queue.
- At posting to the queue of command, ticket is assigned. As the ticket serves actual time receiving of command. The ticket is used later for command execution status retrieving.
- Actual command status is not known at the time of posting, therefore positive response means success of the posting, but not the execution itself.
- See **CES** query about getting execution status.
- Command format is EXE/Command[/Parameter]
- Command examples:
  - http://192.168.0.160:8081/REST/HTTP_CMD/?EXE/Stop
  - http://192.168.0.160:8081/REST/HTTP_CMD/?EXE/Fire
  - http://192.168.0.160:8081/REST/HTTP_CMD/?EXE/Fire
- Available commands and parameters are listed in product documentation.

RDVAR query

- Returns process variable value.
- Process variables are used to pass parameters between sequence steps.
- Variables are read-only from remote control point of view.
- There are several predefined variables:
  - 'State' - current FSM sate
  - 'LogBlab' - log extent. 0 - only errors are logged, 2 - all sequence steps are logged.
  - 'x' - variable used to pass command parameter. For example, command EXE/Amplification/50 does the following: sets x variable to 50 and performs

sequence 'Amplification'.

- 'ProductID' - product type
- 'ProductSN' - serial number
- RDVAR query example:
  - http://192.168.0.160:8081/REST/HTTP_CMD/?RDVAR/State - returns current state.
  - http://192.168.0.160:8081/REST/HTTP_CMD/?RDVAR/LogBlab - returns logging extent.

CES query.

- Returns status of command that is being executed or is finished.
- Command format is CES[/Timestamp]
- Timestamp is command marker and is received in EXE command response.

- Command example:
  - http://192.168.0.160:8081/REST/HTTP_CMD/?CES/3574677021.391924
- Command without **Timestamp** returns last received from queue command status.
- Query CES returns status code of query and the following command execution parameters:
  1. Command execution status code:
     - -3 - command is posted to the queue
     - -2 - command is received from queue by FSM
     - -1 - command is being executed
     - 0 - command is finished successful
     - >0 - command is finished with error
  2. Step index (IND). Index (IND column value) of current step in execution or the very last step for finished command. Knowing the IND value allows to understand what FSM is doing now.
  3. Result. Result from the sequence step execution. In long wait cycles seeing of interim result may help understanding of process itself. For example, if system waits for heater temperature to rise above some threshold, it maybe useful to see current temperature. Result keeps important information, about consequences of sequence execution error. The keywords, explaining consequences are following "Next:" string can be:
     - **GoToFault** - sequence executing is terminated at this point and command 'GoToFault' is posted to queue. In general, result will some fault indicating state. Please note, GoToFault will the new command and old ticked will not sho
     - **Skipping rest** - sequence executing is terminated at this point, remaining sequence steps are skipped. In general result will be no changes in state. Guards are generating this error.
     - **Ignoring error** - there will be no consequences, except that the last step was gone wrong.
  4. Source. HTTP_CMD.vi for remote control command.
  5. Last operation timestamp

Responses to queries.

- Responses to queries are formatted according to RES_HTML column information in COM table, file unilaz.db

COM_NAME	FUNCTION	RES_PAR_COUT	RES_HTML	DESCRIPTION
EXE	exeSEQM	2	`%d <a href="?CES/%d">Check status</a>`	2. Command ticket(timestamp)
RDVAR	RetTrueValue	2	`%d %s  %s`	2. Variable value 3. Type
CES	checkID	6	`%d %d %d %s  %s  %f`	2. Command execution status code 3. Step index (IND) 4. Result 5.Source 6. Last operation timestamp
LIST	SQL	2	`%d <code>%s</code>||;|| `	2. SQL query result in list form
DATA	SQLdata	2	`%d <code>%s</code>||;| `	2. SQL query result in list form

There:

- **COM_NAME** - query/command name as it appears in query.
- **FUNCTION** - internal function name. Should be left unchanged, because this is link to the actual function.
- **RES_PAR_COUT** - parameters, that are passed to the format function in response to query.
- **RES_HTML** - format string specifies how you want the function to convert the input arguments into resulting string.
- **DESCRIPTION** - parameter descriptions. The first parameter always is response code, therefore is omitted.

General rules:

- Very first parameter always is query response code. Zero code means, that query/command was accepted and the following results are valid. Non zero means, that query can't return any meaningful results because of error that code is representing.
- The following parameter(s) is(are) actual result(s) of query.
- Query responses have minimal formatting, just to keep readability in modern browsers. By changing format strings there is possibility to comply to many of information encoding standards: XML, JSON, HTML.
- Response formatting is straightforward for EXE, RDVAR, CES queries, there is just single format string.
- Response formatting for LIST and DATA queries is split in two steps:

  1. `%d<br><code>%s</code>`

     - is overall envelope. There %d is response code, %s is response body.

2. '|' symbol splits evenlope from the row formatting.

- Row formatting rules are:
  1. DATA query. There are three strings separated by '|' symbol: 1. row beginning (empty string) 2. separator (semicolon) 3. row terminator (single line break).
  2. LIST query. There are four strings separated by '|' symbol: 1. column beginning (empty string) 2. column separator (semicolon) 3. row beginning (empty string) 3. row terminator (single line break).

Response samples

Query of available commands: http://192.168.0.160:8081/REST/HTTP_CMD/?LIST/SEQUENCES/DISTINCT SEQUENCE

Response

```
0<br><code>SomeEvent;<br>Init;<br>SyncMode;<br>Amplification;<br>Watchdog;<br>GoToFault;<br>Stop;<br>Fire;<br>EnMode;<br></code>
```

Query of chanel 2 data: http://192.168.0.160:8081/REST/HTTP_CMD/?DATA/2

Response

```
0<br><code>1631885132;0.000000;<br>1631885509;0.000000;<br>1631885881;0.000000;<br>1631886255;0.000000;<br>1631886631;0.000000;<br>
```

Command 'Stop': http://192.168.0.160:8081/REST/HTTP_CMD/?EXE/Stop

Response

```
0<br><a href="?CES/3575601570403">Check status</a>
```

Query of 'Fire' command execution status with ticket: http://192.168.0.160:8081/REST/HTTP_CMD/?CES/3574823433.061258

Response, command successful:

```
0<br>0<br>142<br>%22Idle%22 <br>HTTP_CMD.vi <br>09:39:30.694 2017.04.21
```

Response, error happened:

```
0<br>310<br>79<br>Next:%20Skipping%20rest%20 <br>HTTP_CMD.vi <br>09:42:21.460 2017.04.21
```

Query of product serial: http://192.168.0.160:8081/REST/HTTP_CMD/?RDVAR/ProductSN

Response:

```
0<br>"001"
```

Error handling and information system

FSM handles errors from the following sources:

1. FSM executing module
2. CAN Network
3. Http server
4. Hardware modules

- Every single step of sequence have assigned some error handler type. In case of step objective can't be achieved, error handler performs some predefined actions.

Error handler types

Name	Parameter	Action(s)	Typical use
ResetErr	-	Error is cleared, 'Clean completion' is posted as result to the log with success code	Used together with **waitfor** command that is acting as fixed time delay
IgnoreErr	-	Error is cleared, 'Next: Ignore error' is posted as result to the log with fault code	Useful while debugging not to break execution but see error code in the log
SkipRestOnErr	Optional, substitute code	'Next: Skipping rest' is posted as result to the log with original error code or if supplied with substitute code. Execution of the sequence is terminated	Used in guards in beginning of sequence. In case guards are not passed, no further steps are executed and FSM state remains unchanged
FaultOnErr	Optional, substitute code	'GoToFault' event is posted in to queue, 'Next: GoToFault' is posted as result to the log with original error code or if supplied with substitute code. Current sequence is terminated, GoToFault sequence is retrieved from queue and executed	Used to direct FSM to the fault mode

- Substitution code in SkipRestOnErr and FaultOnErr is used to replace standard error message with information that is related to the particular system. For example, in case some bit that represents safety interlock state is checked in sequence, FSM will normally post error that this bit has value x instead of required y. This makes the issue troubleshooting sophisticated.