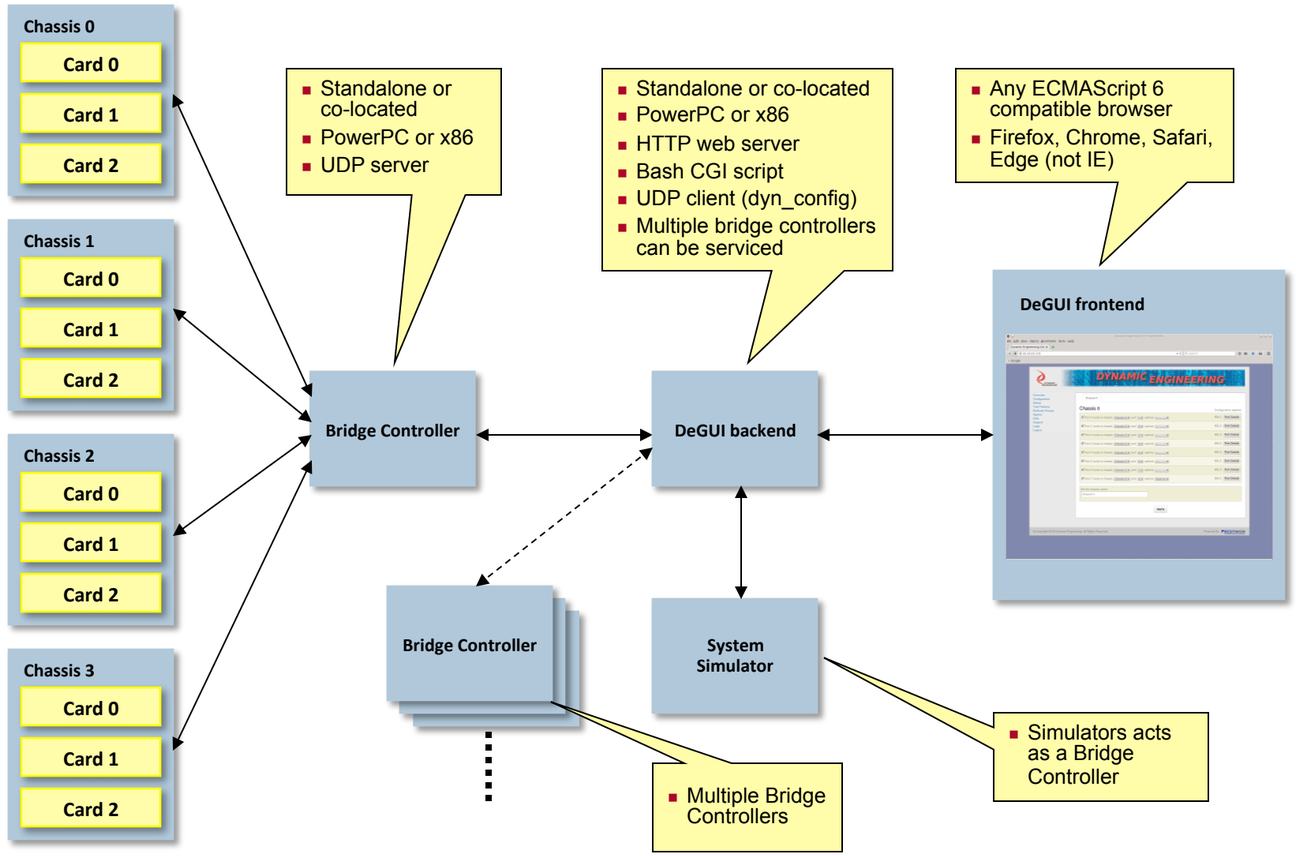
DeGUI

DYNAMIC ENGINEERING

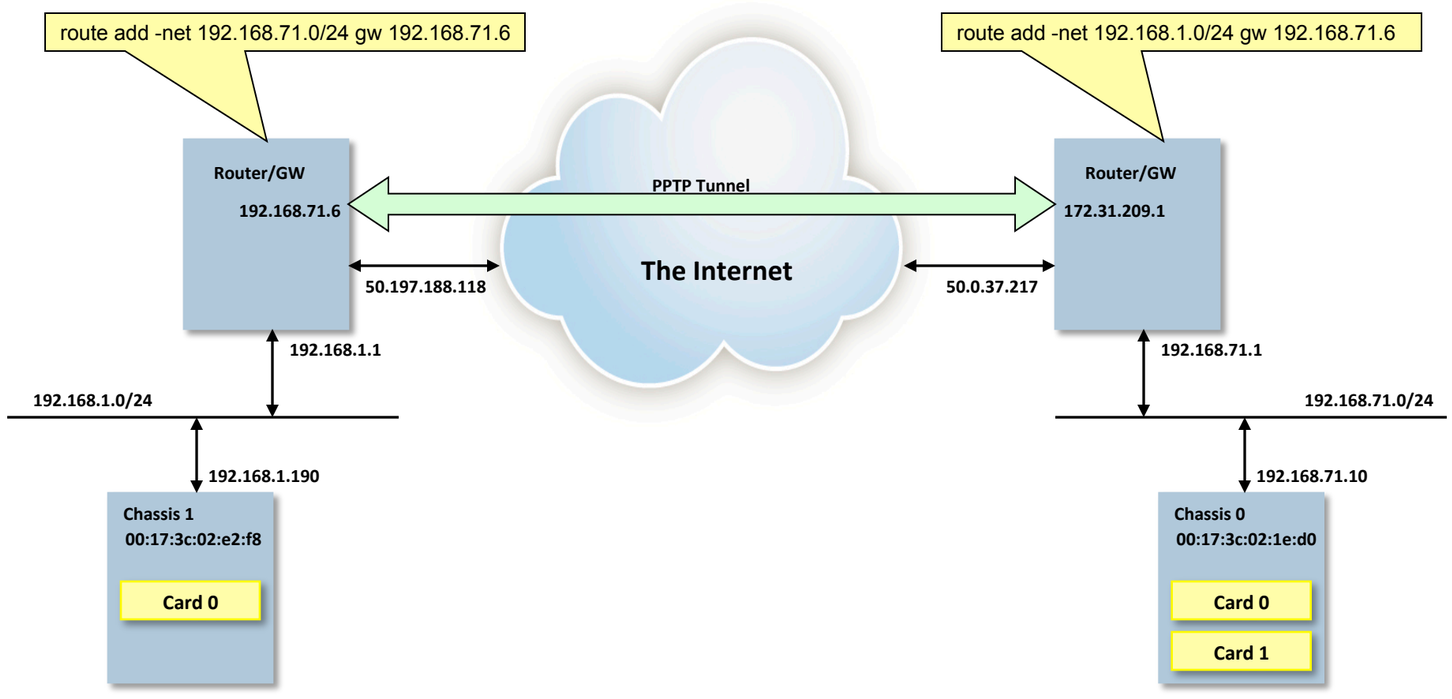
Agenda

- System Architecture
- Multi-chassis test setup
- DeGUI frontend
- DeGUI backend
- Authentication
- Install structure
- RESTful Web API
- Build System

System Architecture



Multi-chassis Test Setup

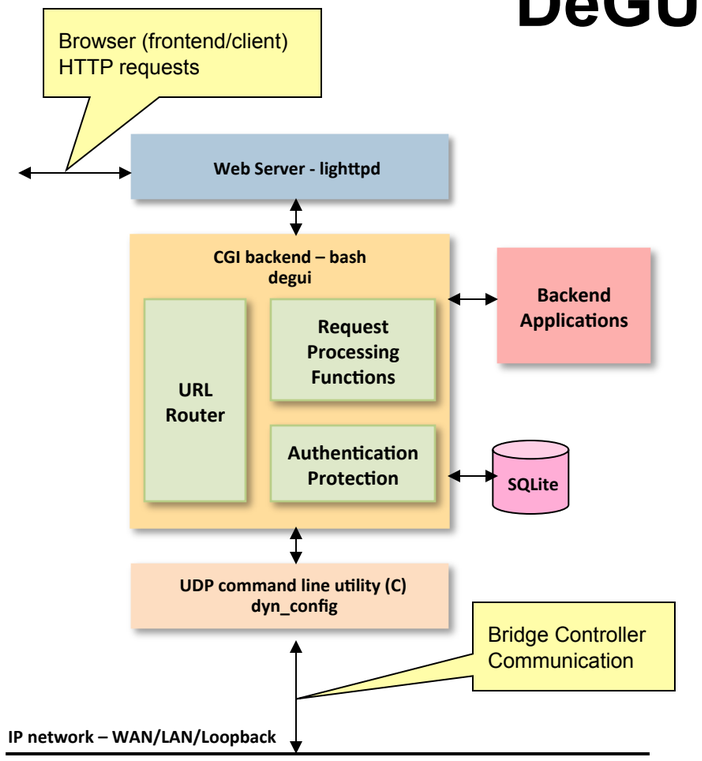


DeGUI frontend

- Dynamically created by using JavaScript
  - Very small initial HTML page.
- main.js – The GUI implementation
  - Fully loaded by initial HTML page
  - JavaScript in conformance with ECMAScript 6 - http://www.ecma-international.org/ecma-262/6.0
  - Browser compatibility described in https://kangax.github.io/compat-table/es6
  - Testing and development with Firefox 38.0.05
- style.css – The Cascading Style Sheet for the GUI
- AJAX calls are used to dynamically update GUI elements from the backend.

```
1
2  <!doctype html>
3  <html>
4  <head>
5  <meta HTTP-EQUIV='Content-Type' CONTENT='text/html; charset=iso-8859-1'>
6  <title>Dynamic Engineering GUI</title>
7  <link rel='stylesheet' type='text/css' href='/css/style.css'>
8  <link rel='stylesheet' type='text/css' href='/css/bootstrap.min.css'>
9  <script type='text/javascript' src='/js/jquery-1.12.4.min.js'></script>
0  <script type='text/javascript' src='/js/bootstrap.min.js'></script>
1  <script type='text/javascript' src='/js/main.js'></script>
2  <script type='text/javascript'>
3  var startme = function()
4  {
5      if(typeof PRG.start === 'function')
6      {
7          PRG.start();
8          return;
9      }
0      window.setTimeout(startme, 200);
1  };
2  </script>
3  </head>
4  <body id='mainbody' class='html not-front not-logged-in one-sidebar sidebar-first page-node page-node- page-node-8 node-type-page' style='margin:Opx;' onload='startme()'>
5  </body>
6  </html>
7
```

DeGUI backend



Design Goals

- Minimize policies in the frontend/client side GUI – Just act as a conduit for data from the backend.
- Dynamically create GUI element based on data queried from the backend.
- Confirmation and error messages driven by the backend.
- System policies and control pushed to the bridge controller as much as possible.
- Automatic system self initialization for easy installation and upgrades.

Backend Control

- Control files in “/usr/local/degui/etc”
- bctrl_ipaddr.txt
  - IP address of bridge controller. If the file is not present, “127.0.0.1” is used.
- bctrl_ipaddr_sim.txt
  - The IP address of the simulator. If the file is not present, the simulator is not considered available.
- droptables
  - Control file to force deletion of configuration database tables. Can contain the following keywords.
    - mulicastgroups – Drop the multicast groups configuration table.
    - sdlcparams – Drop the SDLC port configuration parameters table.
    - spwrparams - Drop the SpaceWire port configuration parameters table.
    - chassis – Drop the chassis table.
    - users – Drop the users table with authentication parameters.
    - chassisports – Drop all the chassis port configuration tables.

Backend Database

- SQL database in /usr/local/degui/db/degui.db
- SQLite - https://sqlite.org
- Five fixed tables
  - chassis – Chassis display names
  - users – Authentication information including valid tokens
  - multicastgroups – All configured multicast groups and their members
  - sdlcparams – SDLC port parameters
  - spwrparams – SpaceWire port parameters
- One table per chassis with chassis ID as the name (MAC address)
  - Port routing information.

Backend Logfiles

- Logfiles are stored in the /usr/local/degui/logs directory
  - degui.log – Main logfile for the GUI backend
  - cgi.log – HTTP server log file
  - error.log – HTTP server error log
- The size of log files are evaluated on each backend access and rotated if they exceed 10485760 (10MB) bytes (degui.log => degui.log.1 and the old degui.log.1 is deleted).
- The HTTP server logfiles are also rotated but since they are still open by the web server, the filesystem is actually not affected (server needs to be restarted).

Backend Applications

- dyn_config
  - Connects the backend to the bridge controller using UDP communication
- dyn_json
  - Parsing JSON data for the backend
- dyn_lock
  - Provides backend locking to serialize requests
- dyn_sim
  - System multi-chassis simulator (bridge controller functionality)
- dyn_stream
  - Streaming server for live captures
- dyn_upload
  - Handles file uploads to the backend

DeGUI System Operations

- System Reset
  - Resets the system by invoking the script/program /usr/local/degui/bin/sysreset if it exists.
- Configuration Resets
  - Creates the file /usr/local/degui/etc/droptables forcing a drop of all database tables.
- Configuration Chassis Resets
  - Creates the file /usr/local/degui/etc/droptables forcing a drop of port configuration tables.
- Reset Backend Lock
  - Tries to reset an out of state lock by invoking “dyn_lock –r”
- Enable Simulator
  - Enable the simulator if the file /usr/local/degui/etc/bctrl_ipaddr_sim.txt exists
- Disable Simulator
  - Disables the simulator if it is active
- Simulator Single Chassis
  - Requests a single chassis simulation by writing to the /tmp/simctrl control file.
- Simulator Multi Chassis
  - Requests a multiple chassis simulation by writing to the /tmp/simctrl control file.
- Reset Authentication
  - Browser local (client side) only
- Password Reset
  - Reset the password by dropping the “user” table through the /usr/local/degui/etc/droptables interface.

DeGUI Authentication

- Login is performed by communicate the SHA-1 hash of the entered credentials to the backend. Username and password is never transmitted directly.
- HTTPS (HTTP over secure TLS) is not used since encryption is not deemed necessary in light of the implications on user experience with the need of server side certificates. It can always be adopted later.
- The authentication uses two tokens, the authentication token and the CSRF token (Cross Site Request Forgery - https://www.owasp.org/index.php/Cross-Site_Request_Forgery_%28CSRF%29).
- The CSRF token is based on the SHA-1 hash of 32 bytes of random data from /dev/urandom.
- Each remote system has a signature based on HTTP headers
  - HTTP_ORIGIN + HTTP_REFERER + HTTP_HOST + REMOTE_ADDR + SHA-1 hash of HTTP_USER_AGENT.
- The authentication token is based on the SHA-1 hash of a string containing the remote system signature, timestamp, 5 bytes of random hex data, username and password (SHA-1 hashed).
- The Authentication token is established at login and is accompanied each protected request to validate the request. This acts as a session token.
- The CSRF token is initiated on login and updated on each protected transaction. It acts as transaction token (one time ticket) that accompanies each protected request.
- Each request is validated based on tokens and various HTTP headers
  - If the HTTP_REFERER is present, it should match HTTP_HOST
  - If the HTTP_ORIGIN is present should match HTTP_HOST
  - The remote system signature must match the stored remote system signature.
  - The both tokens much match the tokens stored in the database

DeGUI Install Structure

```
/usr/local/degui
├── bin
│   ├── bash
│   ├── bashbug
│   ├── dyn_config
│   ├── dyn_json
│   ├── dyn_lock
│   ├── dyn_sim
│   ├── dyn_stream
│   ├── dyn_thread
│   ├── dyn_upload
│   ├── pcre-config
│   ├── pcregrep
│   ├── pcretest
│   └── sqlite3
├── captures
│   └── chassis_0_board_0_port_0.bin
├── config
│   ├── config_00:17:3c:02:e2:f8.txt
│   ├── config_sdlc_param_00:17:3c:02:e2:f8.txt
│   ├── config_spwr_param_00:17:3c:02:e2:f8.txt
│   ├── restore.cfg
│   └── restore.cfg.tmp
├── db
│   └── degui.db
├── etc
│   ├── bctrl_ipaddr_sim.txt
│   ├── bctrl_ipaddr.txt
│   ├── bctrl_ipaddr.txt.org
│   ├── degui.cfg
│   └── lighttpd.conf
├── files
│   ├── ramp.pkt
│   └── test.pkt
├── gui
│   └── forms.sh
├── include (*)
├── lib (*)
├── logs
│   ├── cgi.log
│   ├── cgi.log.1
│   ├── degui.log
│   ├── degui.log.1
│   ├── error.log
│   └── error.log.1
├── sbin
│   ├── lighttpd
│   └── lighttpd-angel
├── share (*)
└── www (**)
```

*

Installed software

- bash-4.4-rc1.tar.gz - BASH scripting
- lighttpd-1.4.39.tar.gz - Web server
- pcre-8.39.tar.gz - Perl-compatible regular expressions
- sqlite-src-3130000.zip - SQL database

Utility JavaScript libraries

- bootstrap-3.3.6-dist.zip
- jsSHA-2.2.0.tar.gz
- jquery-1.12.4.js
- jquery-1.12.4.min.js

**

```
/usr/local/degui
└── www
    ├── css
    │   ├── bootstrap.min.css
    │   ├── bootstrap.min.css.map
    │   └── style.css
    ├── degui
    ├── images
    │   ├── blueruta.jpg
    │   ├── brickwall.png
    │   ├── contract.png
    │   ├── de_home_page.jpg
    │   ├── DE_logo_main.png
    │   ├── DE_logo_main_small.png
    │   ├── de_top_image.png
    │   ├── expand.png
    │   ├── greenball.gif
    │   ├── green_check_25.png
    │   ├── prestinion_logo_small.gif
    │   ├── redball.gif
    │   ├── red_cross_25.png
    │   ├── SecondaryHeader.png
    │   ├── spiffygif_42x42.png
    │   └── spinner.gif
    └── js
        ├── bootstrap.min.js
        ├── jquery-1.12.4.js
        ├── jquery-1.12.4.min.js
        ├── main.js
        └── sha1.js
```

RESTful web API

- Representational State Transfer (REST)
  - An architectural style providing a uniform interface (API) using stateless (application state) client-server messages over HTTP to manipulate recourses represented by the server.
- The DeGUI API uses GET and POST HTTP messages carrying information in both the URI (including query-string parameters) and in the body (JSON formatted data).
- Five classes of API calls
  - Data requests - http://<ip address>/degui/data
  - Core API calls - http://<ip address>/degui/api
  - Apply calls - http://<ip address>/degui/apply
  - Static File download requests - http://<ip address>/degui/get
  - Static File upload requests - http://<ip address>/degui/upload

Request

```
POST /degui/api/sysop HTTP/1.1\r\n
Host: 10.10.10.115\r\n
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:34.0) Gecko/20100101
      Firefox/34.0\r\n
Accept: */*\r\n
Accept-Language: en-US,en;q=0.5\r\n
Accept-Encoding: gzip, deflate\r\n
Content-Type: application/x-www-form-urlencoded; charset=UTF-8\r\n
X-Requested-With: XMLHttpRequest\r\n
Referer: http://10.10.10.115/\r\n
Content-Length: 165\r\n
Connection: keep-alive\r\n
Pragma: no-cache\r\n
Cache-Control: no-cache\r\n
 \r\n
data={“op”:”7”,”auth_token”:”e031c18b003daeb88473124013e1bdd31e28d
      736”,”csrf_token”:5d6d92f69c79e8778ef72ef5710e05b248e84b64”}
```

Response

```
HTTP/1.1 200 OK\r\n
Content-Type: application/json\r\n
Transfer-Encoding: chunked\r\n
Date: Sun, 27 Nov 2016 23:12:45 GMT\r\n
Server: lighttpd/1.4.39\r\n
\r\n
{ "status“ : "OK", "message" : "Simulator Single Chassis Performed.",
      "csrf_token" : "988ee31caf3a156a345bede5242f5cdd3f90a26e" }
```

Data Requests

- Requests various data to help dynamically populate the GUI.
- Uses HTTP GET. Replies are in JSON format.
- Parameters for the requests are in the URI.
- Not protected by authentication (dose not change state on the server).

/degui/data?request=multicastgroups
Requests a list of configured multicast groups and their members
/degui/data?request=capturefilelist
Request the list of data capture files available in "/usr/local/degui/captures"
/degui/data?request=chassislist
Request a list of discovered chassis – queries the bridge controller
/degui/data?request=filelist
Request the list of test pattern files available in "/usr/local/degui/files"
/degui/data?request=type<type>&chassis=<id>&numports=<num>
Request the list of ports for all discovered chassis and their configuration
/degui/data?request=stats&type=<type>&chassis=<id>&numports=<num>
Request RX/TX throughput statistics for all ports in the system
/degui/data?request=sysinfo
Request system information (session timeout)

Core API calls

- Perform various actions, often a request to the bridge controller.
- Uses HTTP POST with data in JSON format . Replies (status) are in JSON format.
- Protected by authentication.

/degui/api/starttx
Inject the selected transmit test pattern (file) into the specified chassis and port.

/degui/api/txdeletefile
Delete the specified transmit test pattern file from “/usr/local/degui/files”.

/degui/api/rxdeletefile
Delete the specified capture file from “/usr/local/degui/captures”.

/degui/api/startrx
Request the start of a file download from “/usr/local/degui/catures”.

/degui/api/livestream
Request start or stop of live streaming to the external streaming server.

/degui/api/capture
Request the start of receive captures. Files will be stored in “/usr/local/degui/catures”.

/degui/api/saveconfig
Allows the user to download and save the configuration (a compressed dump of the database).

Core API calls (cont.)

/degui/api/sysop
Request various system operations to be performed
/degui/api/login
Perform user authentication and creating the session authentication and CSRF tokens
/degui/api/logout
Delete user credentials (authentication token and CSRF token) for a logout
/degui/api/authsave
Save user authentication data i.e. new password for the admin user

Apply Calls

- Apply configuration changes and save changes in the SQL database
- Uses HTTP POST with data in JSON format. Replies (status) are in JSON format.
- Protected by authentication.

/degui/apply/configuration
Saves and applies the port routing configuration with all port configuration parameters

/degui/apply/multicastgroups
Saves and applies the multicast group configuration (groups and members)

Static File Download requests

- Request static binary files.
- Uses HTTP GET. Replies are in “application/octet-stream” format.
- Protected by authentication.

/degui/get/configdata
Saves the configuration (dumps the database) and provides the compressed result in the reply.

Static File Upload requests

- Perform various actions, often a request to the bridge controller.
- Uses HTTP POST with file data. Replies (status) are in JSON format.
- Protected by authentication.

/degui/upload/starttxpkt
Handles the file upload of the transmit test pattern file data.

/degui/upload/restoreconfig
Handles the upload of the configuration and applies the configuration (restores the database tables).

dyn_config

- This application is the link between the GUI backend and the bridge controller. It implements the client side of the UDP based messaging protocol.
- The usage is rather customized with respect to input parameters provided on the command line and in various input files. See the “degui” backend”.
- It creates JSON data for the GUI client side requests for some invocations.

```
Usage: dyn_config [OPTIONS] ...

        OPTIONS are some of the following:

        -r <request>              Specify the request.
           Valid requests:
           chassis - request chassis information.
           ports - request port information.
           config - send configuration information.
           starttx - send a start transmit pattern.
           stats - send a request for statistics.
           livestream - send a request to start a live stream.
      -f <file>                   File(s) with configuration parameters.
      -c <command>                Database command for lookups.
      -t <chassis>                The table (chassis) for lookups.
      -i <chassis index>          The chassis index to use.
      -v <io port>                The I/O port to use.
      -g <group num>              The multicast group number to use.
      -x                          Increase debug level.
      -z                          Reset statistics.
      -p <port>                   Port number to use.
      -m <xxx.xxx.xxx.xxx>        IP address to bind to.
      -h                          Show this message.
```

dyn_json

- This application facilitates parsing of received JSON data from the client side JavaScript GUI.
- It is not a general purpose JSON parser since functionality is limited to what is needed by the “degui” backend.

```
Usage: dyn_json [OPTIONS] ...

        OPTIONS are some of the following:

        -i <num>                The index into an array.
        -f <type>               The input data format.
        -o <type>               The output data format.
        -h                      Show this message.
```

dyn_lock

- This application provides locking in order to limit concurrency to the “degui” backend.
- It is based around POSIX semaphores, allowing for fast wakeup of waiting processes/requests without any polling.
- The semaphore name is “/dyneng_degui”.

```
Usage: dyn_json [OPTIONS] ...

        OPTIONS are some of the following:

        -i <num>                The index into an array.
        -f <type>               The input data format.
        -o <type>               The output data format.
        -h                      Show this message.
```

dyn_sim

- The simulator acts like a bridge controller and responds to the dequi backend messaging as if it is a real system.
- The simulator allows for rapid development of multi chassis GUI features.
- The simulator can simulate a simple one chassis one board system or a fully populated multi-chassis system.
- The simulator is controlled by creating the file “simctrl” with commands in the control directory monitored by the simulator. Below are the understood commands.
  - emptychassis – Respond as if no chassis where discovered.
  - fullchassis – Respond with a maximum populated system.
  - singlechassis – Respond with a single chassis system.
  - errorreply – Respond with an error code on the next message request.
  - noreply – Do not respond to the next message request.

```
Usage: dyn_sim [OPTIONS] ...

        OPTIONS are some of the following:

        -d <directory>            The control directory to use.
        -x                        Increase debug level.
        -p <port>                 Port number to use.
        -m <xxx.xxx.xxx.xxx>      IP address to bind to.
        -h                        Show this message.
```

dyn_stream

- UDP streaming server - Stores data from the bridge controller
- Data is sorted into files with the following names
  - chassis_<chassis id>_board_<board num>_port_<port num>.bin
- Each UDP packet data is prepended with 4 bytes of sequence number (in network byte order) followed by 4 bytes of data length (also in network byte order).

```
Usage: dyn_stream [OPTIONS] ...

        OPTIONS are some of the following:

        -d <directory>            Directory to store files in.
        -b                        Run in the background as a daemon.
        -e                        Exit on error.
        -x                        Increase debug level.
        -l <log file>             Specify a log file for stderr.
        -p <port>                 Port number to use.
        -m <xxx.xxx.xxx.xxx>      IP address to bind to.
        -h                        Show this message.
```

dyn_upload

- This applications supports file uploads to the degui backend.
- The application extracts the data portion of an HTTP multipart message (only one data part is supported) and stores it in a file.

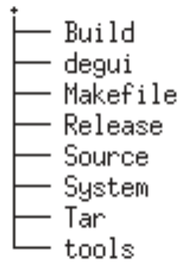
```
Usage: dyn_upload [OPTIONS] ...

        OPTIONS are some of the following:

        -d <directory>            The directory to store files into.
        -f <name>                 Override filename.
        -h                        Show this message.
```

Build System

- The build system support native and cross compilation concurrently
  - “make switch” will alternate between native and cross compiling
  - /usr/local/degui_cross is the cross installation (to be deployed on the target)
  - /usr/local/degui_native is the native installation
  - /usr/local/degui is a link to the active build environment
- Makefile based
- Directory structure
  - Build – Generated directory
  - degui – All GUI files
  - Release – Generated releases and updates
  - Source – The source code for all backend applications
  - Tar – Source archives used in the installation
  - tools – PowerPC cross compile toolchian (from Mike)



Makefile targets

- make buildall – Builds the complete target installation (including release/update)
- Make build – build the 3rd party tools and utilities and installs them
- make progs – Builds the backend applications
- make switch – Switches between native and cross build environments
- make clean – Removes the generated Build directory
- make release – Generates a full release to be deployed on the target. Just untar in /usr/local to install.
- make update – Generates an update with just the GUI and backend components to be deployed on the target. Just untar in /usr/local to install.
- make start – Starts the Lighttpd HTTP server
- make stop – Stops the Lighttpd HTTP server
- make db – Connect to the native SQL database on the development host
- make tables – Dump the tables in the native SQL database
- make sim – start the simulator
- make unlock – unlocks any locks if there are any

Install Instructions

http://filelink.prestinion.com/dyneng/

Username: dyneng

Password: dyn16eng20

Download degui_sdk_113016.tar.gz

Build with:

1. tar xvf degui_sdk_113016.tar.gz
2. cd degui_sdk_113016
3. tar xvf ../tools.tar
4. make buildall
5. make switch
6. make buildall
7. make start
8. make sim

Contact Information

Dynamic Engineering
150 DuBois Street, Ste. C
Santa Cruz, CA 95060
USA

Sales@dyneng.com
Phone: +1 831.457.8891

www.dyneng.com

Thank You