CSULB Marine Biology Department Software Project

Implementation Plan

This document is subject to change. The latest version may be found at https://github.com/ashawnbandy/cecs491/tree/master/docs

12/12/2012

Contents

STATE OF THE PROJECT

Summary

The back-end and front-end, with the exception of the email notification subsystem, works within the parameters of the Requirements Analysis on the targeted hardware. The email notification subsystem must be ported from Java to Perl to accommodate the csulbsharklab.com server environment.

Back-End Configuration

- VEMCO VR2C receiver with firmware version 0.0.25
- Perle IOLAN DS1 Serial-To-Ethernet Device Server with Trueport device driver
- Virtualbox VM emulating Intel i5-2415M @ 2.3 GHZ. 768 MB RAM running Windows 7 Professional Build 7600,

Front-End Configuration (csulbsharklab.com server)

- Linux Kernel v2.6.18 (i686)
- Apache v2.2.2
- MySQL v5.1.65-cli
- PHP 5.3.15
- crond
- Perl interpreter

Availability

The source code and documentation for our project may be obtained in the following ways:

Repository browser: https://github.com/ashawnbandy/cecs491

Zip file: https://github.com/ashawnbandy/cecs491/archive/master.zip

Git: https://github.com/ashawnbandy/cecs491.git

Additionally, the back-end will be installed on a PC at the Marine Biology Department and a copy of the front-end has been installed on the webserver.

Note: The github repository will have the latest version of the source code but will not necessarily have the latest compiled binaries. The latest compilable version of the source code will be in the *master* branch while the most stable while the latest stable version can be found in the *release* branch.

Licensing

We propose all code written for this project to be licensed under the GPL v3.0 (http://www.gnu.org/licenses/gpl-3.0.txt and a full copy may be found in license.txt in our repository).

In summary:

> *The terms and conditions of the GPL must be made available to anybody receiving a copy of the work that has a GPL applied to it ("the licensee"). Any licensee who adheres to the terms and conditions is given permission to modify the work, as well as to copy and redistribute the work or any derivative version. The licensee is allowed to charge a fee for this service, or do this free of charge. This latter point distinguishes the GPL from software licenses that prohibit commercial redistribution. The FSF argues that free software should not place restrictions on commercial use, and the GPL explicitly states that GPL works may be sold at any price.*
>
> *The GPL additionally states that a distributor may not impose "further restrictions on the rights granted by the GPL". This forbids activities such as distributing of the software under a non-disclosure agreement or contract. Distributors under the GPL also grant a license for any of their patents practiced by the software, to practice those patents in GPL software.*
>
> *…the seventh section of version 3 require that programs distributed as pre-compiled binaries are accompanied by a copy of the source code, a written offer to distribute the source code via the same mechanism as the pre-compiled binary, or the written offer to obtain the source code that you got when you received the pre-compiled binary under the GPL. The second section of version 2 and the fifth section of version 3 also require giving "all recipients a copy of this License along with the Program". Version 3 of the license allows making the source code available in additional ways in fulfillment of the seventh section. These include downloading source code from an adjacent network server or by peer-to-peer transmission, provided that is how the compiled code was available and there are "clear directions" on where to find the source code.*
>
> --http://en.wikipedia.org/wiki/GNU_General_Public_License#Version_3

Continued Support

Although we have reached the end of our semester long project, you are free to contact the individual members of the team for continued support. This does not include the email alert subsystem: we will continue work on the port from Java to Perl with no further arrangements necessary.

Security Considerations

During the course of the project, the Marine Biology Department granted our team access to several systems including the csulbsharklab.com web server control panel, the ftp server, and the MySQL database server as well as the web application itself. As a standard security practice, it is important that these accounts are either removed or their passwords changed at the conclusion of our project. If you would like assistance in making these changes, please feel to ask one of the team members.

MAINTENANCE

In addition to the normal operation of the software, there are some non-routine updates that may need to be made when changes to the overall configuration occur. Examples of these changes include adding or changing VEMCO transceiver hardware, changing authentication credentials on the csulbsharklab.com database, or adding, editing or removing access to the real-time query. Care was taken in the design of the system to allow for these changes to be made as easily as possible.

Back-End

The back-end is coded in C# with .NET framework 4.5 and WPF. There are three project files that require Microsoft Visual Studio 2012 to compile and edit. These are *ServerComponents*, *Modules*, and *ServiceController2*.

The complete documentation for the back-end code is compiled into .CHM files (may be opened on any Windows Vista/7 computer) found in the *docs* folder on the git repository.

The help file for the server components is *docs/Help/Documentation.chm*.

The help file for the code found in the *Modules* solution folder is *docs/Help/Modules.chm*.

The Back-End uses a JSON parser called FridayThe13th licensed under LGPL v2. The source code is available both on our github repository and at the library's github repository: https://github.com/jprichardson/FridayThe13th.

Database Configuration

This file allows for changes to the database module without changing the system. It uses JSON formatting and as such changes should be validated (http://jsonlint.com/). See Appendix A for an example.

log_file:

Specifies the location to try to make the database log file. The log file will save failed insertions into the database until they have been successfully entered. In the event of a system crash, this file should be checked for insertions that had not yet gone into the database before the crash. See Appendix A for an example.

database:

setting "use" to "true" will tell the system to use the host, database name, username, and password entered below. If anything other than "true" is entered, the default database values will be used (csulbsharklab with a superadmin account.)

transmitters:

This section allows sensor transmitters to be entered along with their sensor types and data values. Any sensor not entered here will go into the database with the raw A2D (uncalibrated) value into the database with a type of "A2D". Currently, all types are assumed to be of the linear form AX + B, where A is the first value entered here, B is the second value entered, and X is the uncalibrated value. Values must be entered into the format shown in the examples, or they will not be read:

"A69-9001-50":"m,3.55,2.52",

where "A69-9001-50" is the full serial of the transmitter with the codespace, "m" is the one character type of data ("m" here assumed to be depth in meters), and the two numbers as the aforementioned A and B weights.

Receiver Configuration

This file contains configuration primarily for how to read messages from and send commands to the receiver and send messages back. It uses JSON formatting and changes should be validated (http://jsonlint.com/). The system is designed to allow multiple receivers each with possibly different firmware to run concurrently. As long as the text protocol defined by VEMCO does not change between firmware versions remains stable, a new configuration is not necessary. If the protocol does, however, change, the updates can be placed in a new configuration file in the *config* folder and the back-end restarted.

See Appendix A for an example.

firmware_version:

The firmware version of the receiver being used. Multiple copies of this config file may be made with different firmwares, in order for the system to be able to run many receivers with different firmware versions installed concurrently. For example, if there are five receivers in operation with firmware 0 and two with firmware 1, the first five will all read the config file with 0 for this field and the last two will read the config file with 1 entered for this field.

Note: VEMCO uses xxx.yyy.zzz for the firmware version which is converted by the back-end system to a single integer via

(xxx * 10000) + (yyy * 100) + zzz

discovery_commands:

This lists the commands our system will send while trying to discover new receivers that have been connected to the system. It is unlikely these will ever need to be changed.

encoder:

Specifies how to build the actual command portion of messages to send to the receiver. The word in quotations before the colon in each line is the name of the command, and the latter part in quotations is the format of the command. Things in curly brackets are parameters. {0} should be before each command, as that will represent the prefix of each command containing the receiver ID. Adding new commands to this file alone is not enough to change the behavior of the system. However, simple changes such as the status command changing from "STATUS" to "STS" would be possible.

decoder:

Specifies how to interpret messages received from the Receiver using regular expressions. A tutorial on regular expressions is beyond the scope of this document, however, many tutorials can be found online. The following will assume familiarity with RegEx:

"words" contains every possible part of all messages the receiver will send. The regular expressions for this must be enough to uniquely find the datum of interest within the message. The part of the string returned that is of actual interest to the system must be in group 1. For example:

"lv":"LV=([0-9]+\\.[0-9]+)",

for the line voltage field sent during a STATUS message. This will return the string LV=3.65 or something similar out of a STATUS message, and the part that will be used will be group 1, the part of the regular expression in the outermost parenthesis, in this case "3.65".

"sentences" contains information on each type of possible message sent by the receiver. The two fields for each type is:

"format": This is a regular expression that identifies this type of message. Note that this expression does not need to represent the entire received message, but only enough to uniquely identify the type of message from other types.

"word_order": This contains every "word" from above that is contained in the type of message. For example, a detection message contains the receiver_id, a "detection_counter", the "date", "time", "frequency_codespace", "transmitter_id", and "sensor_value" (whether or not the particular detection had a sensor_value).

Modules

Functionality can be added, changed or removed from the system by including a managed run-time DLL in the *modules* directory of the server and then restarting the server. A module *must* implement the i_Module interface found in the EventSlice.Interfaces namespace and must have a constructor which accepts an EventSlice.Dispatcher object. A module *may* extend the abstract class Module in EventSlice.Interfaces. See Appendix B for these definitions and the source code for a simple example module.

Perle Serial-To-Ethernet Bridge Configuration

The Perle Serial-To-Ethernet Bridge (S2EB) is used to package and transmit serial port traffic to and from the VEMCO VR2C transceivers over the network. The server may run up to 4096 Perle S2EBs and if properly configured, no additional work on the back end must be done to add additional transceivers.

The following configuration steps refer to v4.4 of the IOLAN DS1 firmware. The manual for this version can be found in the *docs/IOLAN_DS-TS_UG_V4.4.pdf* and the manual for this version of the TruePort device driver is at *docs/windows_ug-15.pdf* on the project's github repository. For different hardware/firmware configurations (or other devices) please refer to the correct manuals instead.

Configuration Steps

1. Using an Ethernet cross-over cable connect the S2EB to a local computer with Perle's software installed. Also ensure the device is properly powered.
2. Configure the device to use the *TruePort* profile. See page 64 of the DS1 manual.
3. Configure the TruePort driver (identified by the S2EB's current IP address) for an unused COM port. Note the MAC address for this S2EB device.
4. On the *Connections* configuration page, set the driver to *Initiate Connection to Device Server* and on the *Advanced* page set *Send Keep Alive Packets* to true. See page10 of the TruePort driver manual. Also note the port number for which this device is configured.

Installation Steps

1. After configuring the S2EB device, power it down and remove it from the cross-over cable.
2. At the installation site, connect the device's RS-232 to the VEMCO VR2C transceiver and connect the Ethernet port to the local network. Power both devices. The device will attempt to configure itself via DHCP. It is recommended that your DHCP server assign a consistent IP address based on the MAC address noted during configuration. Note this IP address.
3. Configure the gateway device to forward packets on the port noted during configuration to the IP address now assigned to the device.
4. On the machine which is running the TruePort for this device driver (e.g. the computer running the "back-end" at the Marine Biology Department) , change the IP address to match the one you noted in Step 2.

Post-Installation

Data from a new VEMCO receiver will be added to the csulbsharklab.com database, ceteris paribus, but will not be shown in the "Real-Time Query" on the website until the receiver ID number is added to the receiver table. See *Front-End* maintenance.

Summary

Unfortunately, there are too many network routing devices and DHCP servers to describe the exact steps for configuration. The TruePort driver will connect via TCP to the IP address and port that is specified. Assuming that the connection traverses more than one network (i.e. the Internet), then the IP address must be public and the gateway's firewall must not block traffic to the port. In many consumer grade devices, the router includes Network Address Translation and DHCP facilities. Where NAT and private IP addresses are used, the gateway must be configured to forward traffic arriving on the public IP address to the private IP address for the given port.

Finally, there are a great number of options for TruePort and the S2EB device that are beyond the scope of this document but may be needed for more exotic network environments. These are outlined in the two referenced documents.

Front-End

The Front-End is an extension of the csulbsharklab.com web application and is written in PHP and javascript. Files that were modified or added during the course of our project can be found in the *csulbsharklab-extension* folder of the github repository.

Modifying Data in Database – MySQL Workbench

Warning: This procedure can potentially cause the loss of data. See *Backup Procedure* below for instructions on making a copy of the database.

MySQL Workbench can be downloaded from www.mysql.com/products/workbench.

Creating Connection to Database

Note: This procedure only needs to be done once per database connection on a new copy of MySQL workbench.

1. Add "New Server Instance." This will be instance of actual sharklab database.
2. Select "Remote Host"
3. Address: www.csulbsharklab.com
4. Clock “Next”
5. Give your connection a name, e.g. www.csulbsharklab.com
6. Connection Method: Standard (TCP/IP)
7. Hostname: www.csulbsharklab.com
8. Port: 3306
9. Username: csulbsha_shark
10. Password: <database password>
11. Click “Next”
12. Select "Do not use remote management"
13. Click “Next”
14. Name your instance name, e.g. www.csulbsharklab.com

Add/Remove Data

1. Open connection to database by double clicking **csulbsharklab** under the section “Open Connection to Start Querying.” A new tab will open.
2. Expand “csulbsha_sharktopus” in the object browser.
3. Expand “Tables.” This will list all the tables that are in your database.

Example

Now that we have a list of all the tables, we can manage their contained data.

1. Right click the **projects_members** table and select “Edit Table Data”. All data associated with **projects_members** will be displayed.
2. Lets **ADD** member **5** to the **White_Shark** project. In a new row, enter **5** under the column **members_id**.
3. Enter **White_Shark** under the column **projects_name**.
4. Click the “Apply” button.
5. The changes are now saved to the database. Let’s now **REMOVE** the row we just added to the **projects_members** table.
6. Select the leftmost cell of the row we just added.
7. Right click the cell and select “Delete Row(s)”
8. Click the “Apply” button. Note: Changes made will not be saved unless the “Apply” button is clicked.

The changes are now saved to the database. If you want to edit an existing item in the table, simply select the cell you want to edit and make your changes.

Backup Procedure

Summary

1. Import structure and data from csulbsharklab.com within 30 minutes of the update procedure.
2. Save the import on permanent, detached medium (e.g. CD-ROM).
3. Make changes to the structure and data on csulbsharklab.com

If we encounter an unrecoverable error, then we will reload the backup and, if possible, restart this procedure at step 2.

Creating Connection to Database

Note: This procedure only needs to be done once per database connection on a new copy of MySQL workbench.

1. Add "New Server Instance." This will be instance of actual sharklab database.
2. Select "Remote Host"
3. Address: www.csulbsharklab.com
4. Clock “Next”
5. Give your connection a name, e.g. www.csulbsharklab.com
6. Connection Method: Standard (TCP/IP)
7. Hostname: www.csulbsharklab.com
8. Port: 3306
9. Username: csulbsha_shark
10. Password: <database password>
11. Click “Next”
12. Select "Do not use remote management"
13. Click “Next”
14. Name your instance name, e.g. www.csulbsharklab.com

Exporting

You can save a backup of your current database by exporting the schema and current data to a .sql file.

1. Click "Manage Import / Export”

2. In the browser to the left, select "Data Export"
3. Under 'Database Objects to Export' select the database to export from.
4. Under 'Options' select "Export to Self-Contained File" and enter the file path you want save your .sql file to
5. Click "Start Export"

This .sql file contains the database schema and all the data entries from the database you exported.

Importing

Warning: This procedure can potentially cause the loss of data and should be considered a recovery method of last resort. Always attempt to make a backup of the current data by following the above procedure for exporting data before continuing.

You can revert to a backup by importing the .sql file that contains your backup data. The .sql file you're going to import contains the schema and all the data entries from the database you exported from.

1. Click "Manage Import / Export"
2. In the browser to the left, select "Data Import/Restore"
3. Under 'Options' select "Import From Self-Contained File" and enter the file path to the .sql file to import
4. Click "Start Import"

The database should now contain the schema changes and imported data entries from the .sql file.

APPENDIX A

Examples of the following files may be found in the github repository.

Main Configuration

```
{
   "firmware_version":0,
   "discovery_commands":[
      "*BROADC.A#ST,QUIT",
      "*BROADC.A#ST,DISCOVERY",
      "*DISCOV.E#RY,DISCOVERY"
   ],
   "encoder":{
      "INFO":"{0}INFO",
      "START":"{0}START",
      "STOP":"{0}STOP",
      "ERASE":"{0}ERASE",
      "RTMNOW":"{0}RTMNOW",
      "RTMINFO":"{0}RTMINFO",
      "RTMOFF":"{0}RTMOFF",
      "RTM232":"{0}RTM232",
      "RTM485":"{0}RTM485",
      "RTMPROFILE":"{0}RTMPROFILE={1}",
      "RTMAUTOERASE":"{0}RTMAUTOERASE={1}",
      "STORAGE":"{0}STORAGE",
      "TIME":"{0}TIME={1}-{2}-{3} {4}:{5}:{6} {7}",
      "RESET":"{0}RESET",
      "QUIT":"{0}QUIT",
      "RESETBATTERY":"{0}RESETBATTERY"
   },
   "decoder": {
```

```
		"words": {
			"date":"([0-9]{4}-[0-9]{2}-[0-9]{2})",
			"time":"([0-9]{2}:[0-9]{2}:[0-9]{2})",
			"receivers_id":"([0-9]{6})",
			"frequency_codespace":"([A-Z0-9]{3}-[A-Z0-9]{4})",
			"transmitter_id":"[A-Z0-9]{3}-[A-Z0-9]{4},([0-9]+),"
			"sensor_value":"[A-Z0-9]{3}-[A-Z0-9]{4},[0-9]+,([0-9]+),",
			"detection_counter":"[0-9]{6},([0-9]{3}),[0-9]{4}",
			"hexsum":"#([A-F0-9]{2})",
			"status":"(OK|FAILURE|INVALID)",
			"p":"[0-9]{6}.([0-9])",
			"decimal_sum":"#([0-9]{2})",
			"byte_count":"\\[([0-9]{4})\\]",
			"receiver_model":"([0-9A-Z]+-[0-9A-Z]+):",
			"study_name":"'([0-9A-Z]*)'",
			"map":"',([A-Z0-9]+-[0-9]+)",
			"code_space":"\\[ (([0-9]{4}[ /])*[0-9]{4}) \\]",
			"firmware_version":"FW=(([0-9]+\\.)*[0-9]+)",
			"hardware_version":"HW=([0-9]+)",
			"dc":"DC=([0-9]+)",
			"pc":"PC=([0-9]+)",
			"lv":"LV=([0-9]+\\.[0-9]+)",
			"bv":"BV=([0-9]+\\.[0-9]+)",
			"bu":"BU=([0-9]+\\.[0-9]+)",
			"i":"I=([0-9]+\\.[0-9]+)",
			"t":"T=([0-9]+\\.[0-9]+)",
			"du":"DU=([0-9]+\\.[0-9]+)",
			"ru":"RU=([0-9]+\\.[0-9]+)",
			"xyz":"XYZ=(-?[0-9]+\\.[0-9]+:-?[0-9]+\\.[0-9]+:-?[0-9]+\\.[0-
9]+)",
			"state":":-?[0-9]+\\.[0-9]+,([A-Z]+)",
			"rtm_mode":",(OFF|232|485),",
```

```
                        "si":"SI=(POLL|[0-9]+)",

                        "bl":"BL=(U|[0-9]+)",

                        "bi":"BI=(WFS|[0-9]+)",

                        "ma":"MA=(U|[0-9]+)",

                        "fmt":"FMT=([A-Z_ ])*",

                        "endline":"(\\r\\n)"

                },

                "sentences": {

                        "detection_event": {

                                "format":"([0-9]{6}),([0-9]{3}),([0-9]{4}-[0-9]{2}-[0-
9]{2}) ([0-9]{2}:[0-9]{2}:[0-9]{2}),([A-Z0-9]{3}-[A-Z0-9]{4}),([0-9]+)",


        "word_order":["receivers_id","detection_counter","date","time","frequency_codespac
e","transmitter_id","sensor_value"]

                        },

                        "generic_response": {

                                "format":"\\*([0-9]{6})\\.([0-9])#([0-9]{2})\\[([0-
9]{4})\\],(OK|FAILURE|INVALID),#([A-F0-9]{2})",


        "word_order":["receivers_id","p","decimal_sum","byte_count","status","hexsum"]

                        },

                        "rtm_info_response": {

                                "format":"(OFF|232|485),SI=(POLL|[0-9]+),BL=(U|[0-
9]+),BI=(WFS|[0-9]+),MA=(U|[0-9]+),FMT=([A-Z_ ])*",


        "word_order":["receivers_id","p","decimal_sum","byte_count","rtm_mode","si","bl","
bi","ma","fmt"]

                        },

                        "info_response": {

                                "format":"([0-9A-Z]+-[0-9A-Z]+):([0-9]{6}),'([0-9A-
Z]*)',([A-Z0-9]+-[0-9]+)",


        "word_order":["receiver_model","receivers_id","study_name","map","code_space","fir
mware_version","hardware_version"]

                        },

                        "status_response": {

                                "format":"STS,DC=([0-9]+),PC=([0-9]+)",
```

```
        "word_order":["receivers_id","date","time","dc","pc","lv","bv","bu","i","t","du","
ru","xyz"]

                        }

                }

    }

}
```

Database Config

```
{

"log_file":"database\\databaseErrorLog.txt",

"database":{

       "use":"false",

       "host":"csulbsharklab.com",

          "name":"csulbsha_sharktopus",

          "user":"usename",

          "pass":"password"

    },

    "transmitters":{

       "A69-9001-50":"m,3.55,2.52",

       "A69-9023-623":"p,74.2,130"

    }

}
```

APPENDIX B

Module (Abstract Class)

```
namespace EventSlice.Interfaces
{
    /// <summary>
    /// This class contains the specification for all classes wishing to
participate in the event system
    /// must implement.
    /// </summary>
    public abstract class Module : i_Module
    {
        /// <summary>
        /// A reference to the real time event dispatcher where events will be
dispatched.
        /// </summary>
        public Dispatcher dispatcher { get; set; }



        /// <summary>
        /// The default constructor.
        /// </summary>
        /// <param name="dispatcher">A reference to the event dispatcher from
which real time events will originate.</param>
        public Module(Dispatcher dispatcher)
            { this.dispatcher = dispatcher; }


        /// <summary>
        /// This should return a short, human-readable name for the module.  Note:
Not guaranteed to uniquely identify a
        /// module in a running system.
        /// </summary>
        public abstract string getModuleName();
        /// <summary>
        /// This is the handler each module should implement for events sent from
the event dispatcher.
        /// </summary>
        /// <param name="realTimeEvent">The real time event to be handled.</param>
        public virtual void onRealTimeEvent(Interfaces.RealTimeEvent
realTimeEvent) { }
    }


}
```

i_Module (Interface)

```
namespace EventSlice.Interfaces
{
    public interface i_Module
    {
```

```
        /// <summary>
        /// This should return a short, human-readable name for the module.
        /// Note:  Not guaranteed to uniquely identify a
        /// module in a running system.
        /// </summary>
        string getModuleName();
        /// <summary>
        /// This is the handler each module should implement
        /// for events sent from the event dispatcher.
        /// </summary>
        /// <param name="realTimeEvent">The real time event to be handled.</param>
        void onRealTimeEvent(Interfaces.RealTimeEvent realTimeEvent);
    }
}
```

Module Example: Simple Console Logger

```
using EventSlice.Interfaces;
using EventSlice;

namespace ConsoleLogger
{
    /// <summary>
    /// Reports real time events to the console.
    /// </summary>
    public class ConsoleLogger : Module
    {
        /// <summary>
        /// Constructor
        /// </summary>
        /// <param name="dispatcher">A reference to the running system real time
event dispatcher.</param>
        public ConsoleLogger(Dispatcher dispatcher)
            : base(dispatcher) { }

        /// <summary>
        /// Human-readable text of any real time event dispatched is printed to
the console.
        /// </summary>
        /// <param name="rte">Real time event dispatched</param>
        public override void onRealTimeEvent(RealTimeEvent rte)
        {
            System.Console.WriteLine(rte);
        }

        /// <summary>
        /// For use by UI components to provide a human readable name for this
module.
        /// </summary>
        /// <returns>Name of this module.</returns>
        public override string getModuleName()
        {
            return "Console Logger";
        }
    }
}
```