CSULB Marine Biology Department Software Project

Design Specification (rev 1)

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

11/26/2012

Contents

Subsystem Description

The front-end is one cohesive web application, and does not have subsystems.

The back-end system is divided into three major subsystems:

- Serial/Receiver: This represents, unsurprisingly, the physical receiver and the serial ports (virtual or not) which they are connected to, and contains methods to connect to and control those.
  Comprised classes: SerialPortService, Receiver, Encoder
- Events: The events system is the mechanism in which our system communicates with all other parts of itself, by receiving any of various types of RealTimeEvents and sending them along to the other classes.
  Comprised classes: Dispatcher, RealTimeEvents
- Modules: All other parts of the system are modular in nature, and are loaded if and only if their DLL is located in the proper folder. The system can operate without any of these, though with less functionality.
  Comprised classes: ConsoleLogger, Database, Decoder, UI

Back-End Design

Subsystem Communication

Information is passed between subsystems by the way of RealTimeEvents. All modules must contain a reference to the Dispatcher object. Classes may enqueue any type of RealTimeEvent object into the event queue located inside the Dispatcher, which will send a copy of that event to every module in the system. What each individual module does with the event is determined inside the module, and in many cases it simply ignores the event.

Flow of Control

When the service is started, it will periodically poll the computer's serial ports for new receivers. When one is found, it will acquire the firmware version and serial number for that receiver and add it to the list of receivers which the user can view in the UI. The user can turn these receivers on and off.

Error Handling

Noncritical errors will be caught and printed to the console. Database insertion errors will be logged to the local computer in a file specified by the user.

Persistent Data

The database for storing information our system receives is offsite and independent of our system. The only data stored by our system is error log files.

Hardware

The main pieces of hardware for this system are the receiver itself and the serial-to-ethernet bridge. The receivers are the basis for the entire project, and the system is thus designed to connect to and communicate with them. The serial-to-ethernet bridge allows the receiver to be connected to from an off-site computer (where the system is installed). This device creates a virtual serial port that interacts transparently with our system.

Front-End Design

Flow of Control

Email Setup

The control flow for a user starts in the login page. Once the user's credentials are verified, the user will be redirected to the home page. To change the email settings associated with the user, the user must click the "Email Setup" hyperlink located in the main menu. On this page the user can change his/her email address as well as their email preference. After completing the form, the user must click the "Submit" button in order to insert the new email address and preference into the database. If the email address is valid, the submission is successful and the user will be redirected to the home page.

Real-Time Display

The control flow for a user starts in the login page. Once the user's credentials are verified, the user will be redirected to the home page. To access the real-time data associated with the user, the user must click on the "Real-Time Query" hyperlink located in the main menu. The page will display all detections for the past 2 days from all receivers that are associated with the current user. The page refreshes and queries the database for new data every 5 minutes. If new data is found in the database, the page will notify the user and color code the new data in red.

Error Handling

Email

After the user submits the form with their email address, a script runs and checks if the email address is syntactically correct (using regular expression, although it's not perfect). If the email address is invalid, the user will be prompted to reenter their email address.

Persistent Data

All of the persistent data will be stored in a mysql database.

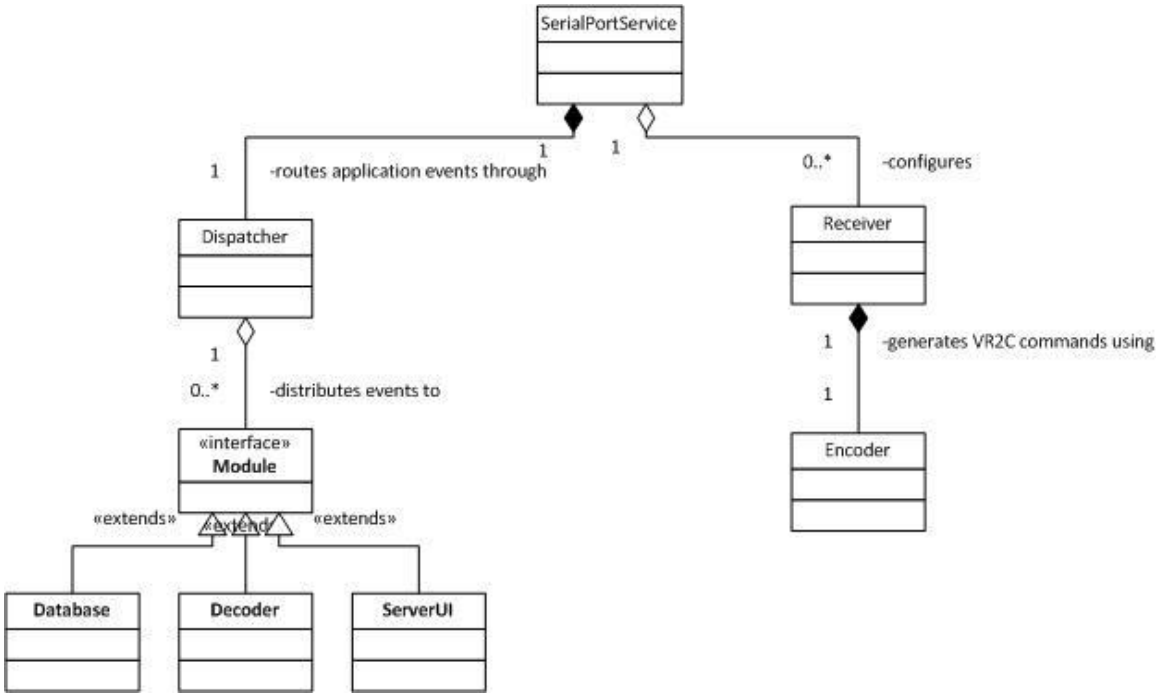
The persistent data will include information about:

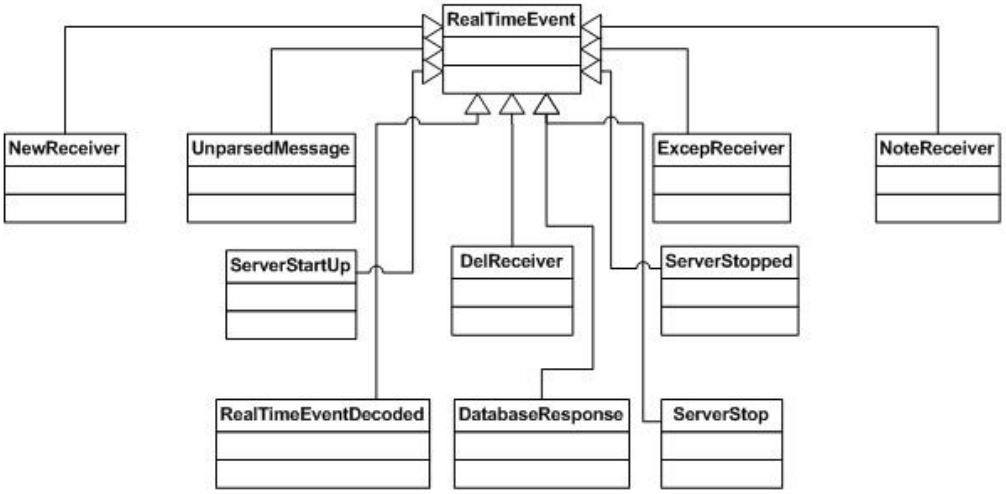
- Members
- Projects
- Stations
- Receivers
- Fish

Each item has its own table in the database. The query pages within the website create a query to fetch and display the desired contents which correspond to each table. Our website will provide a simple interface to add/replace the email address corresponding to a particular member.

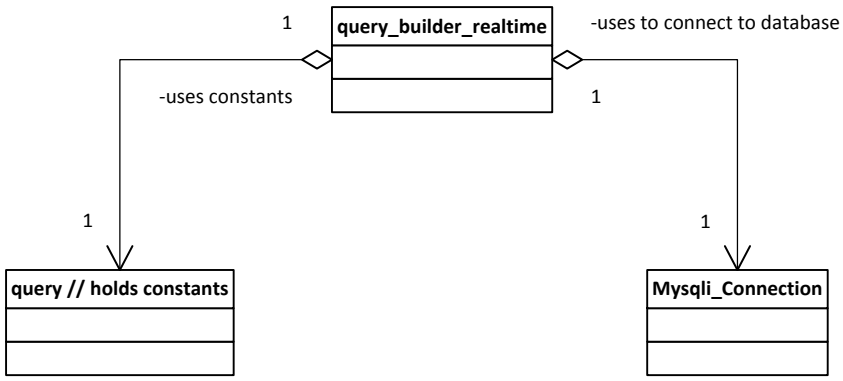
CLASS DIAGRAMS

Back-End





Front-End


1
query_builder_realtime
-uses to connect to database
-uses constants
1
1
1
query // holds constants
Mysqli_Connection

METHOD DOCUMENTATION

Back-End

See ***Service.chm*** and ***Module.chm***

Front-End

Query_Builder_Realtime Class

```
class Query_Builder_Realtime
{
      //
      // This class provides functionality to generates sql
      // queries to send to database.


      // PreConditions:
      //       none
      // PostConditions:
      //    A string containing an insert query for inserting a
      //       user's email address and email preference into
      //       database is returned.
      public static function GenerateEmailQuery( $email,
                                                      $emailPreference);


      // PreConditions:
      //       none
      // PostConditions:
      //       A string containing a Count(*) query to count how
      //       many entries are currently in the database is
      //       returned.
      public static function GenerateCountQuery();



      // PreConditions:
      //       none
      // PostConditions:
      //       A string containing the complete realtime data query
      //       is returned.
      public static function GenerateQuery();


      // PreConditions:
      //       none
      // PostConditions:
      //       A string containing the FROM clause for realtime
      //       data query is returned.
      protected static function BuildFromClause();


      // PreConditions:
      //       none
      // PostConditions:
      //       A string containing the WHERE clause for realtime
```

```
        //      data query is returned.
        protected static function BuildWhereClause();


        // PreConditions:
        //      none
        // PostConditions:
        //      A string containing the full realtime data query
        //      with the ORDERBY clause is returned.
        protected static function BuildFullClause( $select, $from, $where,
$sort = '' );

   };
```

Query_Process Class

```
class Query_Process
   {
        //
        // This class provides functionality to send queries to
        // the database and display results on the query web
        // pages.


        // PreConditions:
        //      none
        // PostConditions:
        //    Returns total count from the query results.
        public static function GetCount($sql)


        // PreConditions:
        //      none
        // PostConditions:
        //    A table containing the realtime data results
        //      is displayed
        public static function GenerateRealTimeTable($sql,
                                $numRows, $page, $sort, $order,
                                $newDataCount, $isSensor = false)




        // PreConditions:
        //      none
        // PostConditions:
        //    Divides the query results into pages and indexes
        //      them for easy access.
        public static function GeneratePagination($page, $limit,
                                                        $totalCount)


        // PreConditions:
        //      none
        // PostConditions:
        //    Updates the database with new email address using
        //      UPDATE query.
      public static function UpdateEmail($updateEmailQuery)

};
```