ICS314 iCalendar Testing

Alex Shum

August 9, 2014

Our implementation of the iCalendar specification centers around the idea of modularity and encapsulation. The Calendar class is the main class responsible for calendar events and will store the individual parts of the event: version, classification, time-start/time-end, geographic position, priority level, timezone and recurrence rules. Each of these individual event-parts is represented by its own class that will handle input validation and will output the event parts as a string following RFC 5545 specifications. Once each of the event-parts has been correctly specified, the Calendar class will combine each part along with start and end tags together into a valid event. The Calendar class will take input from the UI and output an *ics* file after each event-part has been validated by the individual classes.

The reason for this design was to ease parallel development and testing. Once the outline of the Calendar class was created, separate classes for each of individual event-parts were worked on separately. Additionally, the UI was also developed without worrying about calling any of the individual event-parts correctly as the UI would only need to call the methods in the Event class. Due to the modular design, testing was done in three phases: unit testing, integration testing and final testing. Unit testing is explained in detail below.

Each event-part was contained in its own class and this was by design to simplify testing. Since many of these classes were written before the UI, they were designed to accept different types of input from the UI. For example, the DateTime class, responsible for correctly formatting event start/end dates and time, can take a date string from the UI or take a series of integers for year, month, day, hour, minute, second. For unit test, all the input formats were tested and the output was checked against RFC5545. The following is a description of event-part classes and unit tests:

- Classification: Takes string and returns either "PUBLIC", "PRIVATE", or "CONFIDENTIAL" or exception thrown. Unit tests input "public", "private", "confidential" and other non-recognized strings in various capitalization and with nuisance characters. Unit tests check if the output is one of the correct classifications or if an exception is correctly thrown.
- Timezone: Takes string and returns string: "region/subregion", "GMT+/-offset" or exception thrown. Unit tests input all commonly acceptable time zone checking for valid GMT offsets, valid regions and subregions with at least one example from each geographic region and one example with unusual formatting (underscores or dashes). Unit test check if output consists of valid offset, or region/subregion combination otherwise it will check if exception is thrown.
- Priority: Takes either an integer as a string string or integer and outputs 0-9 or exception thrown. Unit tests input valid numbers, 0 - 9, invalid numbers, random characters and a combination of numbers with characters. Unit tests check if the output is a correct integer 0 - 9 or if exception is correctly thrown.
- DateTime: Takes either date/time string or integers for date. Unit tests input date/time string as the following formats: YYYYMMDD, YYYYMMDDTHH, YYYYMMDDTHHMM, YYYYMMDDTHHMMss or as series of integers. Input includes valid time periods, invalid time periods (negative days, January 33, etc.) and uncommon dates such as leap years. Unit tests check if output is a valid date/time string formatted as YYYYMMDDTHHMMss or if exception thrown for unknown formats. Note that this class will accept invalid time periods and format them correctly – ex: January 33 becomes February 2.
- GeoPosition: Takes string of floats separated by ";". Unit tests input valid coordinates, invalid coordinates (outside valid range of lat/long) and non numeric characters. Unit test checks if output is a string of floats separated by ";" or if exception is correctly thrown.

For integration testing we wrote a tester class and entered in each event-part through the Calendar class and checked that the generated ics file contain the correct event-parts in a valid order. We generated multiple examples and uploaded these ics files into google calendar. For final testing, a similar procedure was done using the finished UI.