## Introduction

The intent behind my investigations that this report describes, was to examine how and if, different calendaring clients (essentially e-mail clients with calendaring functionality) handle notifications regarding the changes of state – *create*, *update* and *cancel* – pertaining to the same underlying calendar event. This could have uses in various WebLearn tools, but in the first instance, it is thought that this will have relevance to the signup tool, more specifically, sending notifications by tutorials they have signed up for by e-mail that can be easily handled, understood and reflected in their personal (Nexus) calendar via their use of a CUA (‘calendar user agent’).

Almost by accident, it was discovered that there are some existing standards that many CUAs already implement for this very purpose. In the rest of this document, they will go by the abbreviations *PUBLISH* – simple notifications with no attendees, or expectations of a response (RSVP) and *REQUEST* (also referred to as ‘group events) ’– more complex messages often associated with scheduling, which have scope for attendees and requests for RSVP, etc. Section 4.1 of iTIP describes 'Published Event Examples' (<http://tools.ietf.org/html/rfc5546#section-4.1>). Section 4.2 of iTIP describes 'Group Event Examples' (<http://tools.ietf.org/html/rfc5546#section-4.2>). These document how the messages are formed in a ‘transport-independent’ way; iMIP describes how to transport these via e-mail, in particular I followed Section 2 of iMIP 'MIME Message Format Binding' (<http://tools.ietf.org/html/rfc6047#section-2>), to compose the e-mails themselves.

## Methodology

### Clients

A number of core CUAs were identified that were felt to be represent of clients are most typically used for calendaring at Oxford. The list below captures those clients and abbreviations for them that are used in the rest of this document:

* **O2010** – Microsoft Outlook 2010.
* **O2007** - Microsoft Outlook 2007.
* **TL** – Mozilla Thunderbird with the Lightning add-on (v6.0.2/v1.0b5 respectively).
* **OWA** – Microsoft Outlook Anywhere (Internet Explorer only).
* **OLITE** – Microsoft Outlook Anywhere Lite (for all web-browsers).
* **IPHONE** – iPhone mail & calendar client (running iOS v4.3.5).

Preliminary investigations revealed that there was not obvious candidate software for generating arbitrary iMIP messages in isolation, so the approach taken was to write a framework in software using Java, to do this (this would not be wasted effort as WebLearn itself is written in Java so this could be used in any actual implementation). This basically took the form of a command-line application that enabled me to send simple iMIP messages.

This gives an idea of the parameters that are possible to pass to it:

Usage: SendNotification [-from from] [-to to] [-smtp smtpHost]

[-type (create|update|cancel)] [-summary (1-line) summary]

[-style (publish|request)] [-debug (\"optional\")]

Here is an actual example:

$> java SendNotification -summary "A test event for a Friday."

-smtp smtp.ox.ac.uk -to alexis.oconnor@oucs.ox.ac.uk -from [lxsocon@gmail.com](mailto:lxsocon@gmail.com) -style publish -type create

(The source code for this is available in the Nexus Subversion repository).

### Message Format

The overall format of the messages was actually ‘borrowed’ by reverse-engineering the format used by Google Calendar (<http://www.google.com/calendar>). Essentially this has the concept of catering for so-called lo-fidelity clients, i.e. e-mail clients without calendaring functionality, so the same underlying calendar information is also sent in plain text and HTML in a human-readable form (so if a client can’t handle the HTML, it should be able to handle the plain text). For the purposes of my testing the plain text and HTML components consisted of placeholder content (this can be seen in Appendix A: Placeholder Message Content), but included to be more realistically representative of actual messages. The message also has the calendar event in iCal format as a file attachment, as this may allow for some users to get the event into a calendar by, for example, saving it to their desktop first.

### Process

The basic process was to send *create*, *update* and *cancel* notifications for the same calendar event to each CUA in quick succession for each style (*PUBLISH* & *REQUEST*). Whilst the test framework accepted certain arguments, in order to circumvent the lack of persistence between runs, issues related to time-stamps of the events were handled internally; events were created 1 hour in duration and 24 hours into the future (rounded down to the start of the hour of day). In practice, “updates” can take many forms, e.g. change of time, venue, attendees, etc. For our purposes, I restricted updates to a change of time, specifically, moving the event forward by one hour.

## Results

### Explanation

The points below capture for each kind of notification related to a calendar event what was required by each CUA, in order to achieve a ‘PASS’:

* *Create* – Present suitable controls in the UI, e.g. accept / decline. Use of an appropriate control adds the event to the calendar.
* *Update* – Present suitable controls in the UI, e.g. accept / decline. Use of an appropriate control correctly updates the event to the calendar, e.g. *does not* duplicate it.
* *Cancel* – Present suitable controls in the UI, e.g. accept / decline. Use of an appropriate control removes the event from the calendar.

|  |  |  |  |
| --- | --- | --- | --- |
| PUBLISH | | | |
|  | **Create** | **Update** | **Cancel** |
| **O2010** | PASS | PASS | PASS |
| **O2007** | PASS | PASS | PASS |
| **TL** | PASS | *FAIL* (1) | PASS |
| **OWA** | PASS | PASS | PASS |
| **OLITE** | PASS | PASS | PASS |
| **IPHONE** | PASS | PASS | PASS |

|  |  |  |  |
| --- | --- | --- | --- |
| REQUEST | | | |
|  | **Create** | **Update** | **Cancel** |
| **O2010** | PASS | PASS | PASS |
| **O2007** | PASS | PASS | PASS |
| **TL** | PASS | PASS | PASS |
| **OWA** | PASS | PASS | PASS |
| **OLITE** | PASS | PASS | PASS |
| **IPHONE** | PASS | PASS | PASS |

*Notes*

1. The actual rendering of the message was to annotate it with the text ‘This message contains an event’, but to include no UI controls for you to act upon it. I have reported this as a bug to Mozilla:

*Updating an existing event (where METHOD:PUBLISH) is not handled correctly.*

https://bugzilla.mozilla.org/show\_bug.cgi?id=686444

### General Observations

* CUAs handled the *update* notification (in particular) with varying degrees of richness. Several clients did not make it particularly obvious that it was an update to an event that already existed; arguably some did this by ‘comparative inference’, i.e. it was more explicit when the notification was a *create* (via the use of text annotations and UI controls)*,* so from previous experience, one could assume a more subtle notification was actually an *update*! At the other end of the scale, O2007 and O2010 had explicit messages about it being an update to an existing event and showed the original time crossed out.
* CUAs differed with how (or indeed if) they displayed the event within the calendar context and indeed whether this was from a pre- or post- acceptance point of view. Certain clients, e.g. iPhone had a ‘Show in Calendar’ button so you could the event in the context of the calendar (annotated with a dashed border in order to indicate its ‘tentative’ status). O2010 showed the event from a (tentative) post-acceptance point of view, “inline” within the message pane, without needing to press a button to bring up a view of the calendar in another window.
* There was variance amongst the CUAs as to how they handle the messages “post-acceptance”. Microsoft clients delete the messages by default (although this is configurable). TL tends to keep the message in-place, but alters any text annotation, e.g. for a *create* notification, the annotation ‘This message contains an event.’ changes to ‘Event added to calendar.’

## Summary

All the CUAs tested handled the (more complicated) *REQUEST* style in a satisfactory way. An added benefit is that pursuing this style of notification would allow for possible richer augmentation to the associated calendaring functionality in future. On the other hand, the only CUA to fail the *PUBLISH* suite, was (specifically) Mozilla Lightning, an open-source product with a public issue-tracker, so it is conceivable that that particular deficiency may get fixed within the lifetime of this project.

## Appendix A: Placeholder Message Content.

This section captures the content of the plain text and HTML sections of the test iMIP messages. (This can help to distinguish between annotations made by the CUA and that which represents actual “content”, when viewing the associated screenshots).

PUBLISH + REQUEST:

Create

Plain Text: This could be the event in plain text form.

HTML: This could be the event in HTML form. (NOTE: italicized).

Update

Plain Text: An update to an event in plain text form.

HTML: An update to an event in HTML form. (NOTE: italicized).

Cancel

Plain Text: Cancellation of an event in plain text form.

HTML: Cancellation of an event in HTML form. (NOTE: italicized).

## Appendix B: Screenshots.

The raw screenshots are to be currently found in the zipped archive imip-clients-images.zip, which is to be found in the *Shared Documents* document library, in the same SharePoint site as this document. (Inside, each folder pertains to a particular CUA).