1. Introduction

The Domino calendar service is now included in the extension library. The calendar service represents calendar data in both JSON and iCalendar formats. The calendar service lets you send HTTP requests to:

- Discover a list of calendar resources for the authenticated user.
- Read summary data for events in a given date range.
- Create, read, update and delete an individual event.
- Read summary data for new invitations.
- Read an individual invitation or other notice.
- Process an invitation or other notice. Participant actions include accept, decline, counter, and delegate.

The calendar service is the latest addition to the family of REST services collectively called Domino Access Services. Like the Domino data service, the calendar service needs to be enabled by a Domino administrator. See section 2 for details.

If you are familiar with the data service, you should find the calendar service easy to work with. It shares some of the same URL parameters and other conventions used by the data service. The following sections help you get started with the calendar service. As the calendar service evolves, we will publish a more complete reference document.

2. Enabling the Calendar Service

After you install the 9.0 version of the extension library, the calendar service is loaded whenever the Domino HTTP task is started. However, an administrator typically doesn’t want the calendar service to handle requests on every Domino server. It only makes sense to enable the calendar service on a messaging server (a server that contains user mail files).

You need to deliberately enable the calendar service in the appropriate Internet Site document. To enable the calendar service:

- Use a Notes client to open the server’s public address book.
- In the Domino Directory navigator, select Configuration – Web – Internet Sites.
- Open the Internet Site document for your server.
- Click the Edit Web Site action.
- Select the Configuration tab.
- At the bottom of the form, look for a section labeled Domino Access Services.
- In the Enabled services field, add the Calendar keyword:

<table>
<thead>
<tr>
<th>Domino Access Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>The following setting is a placeholder for services provided by an external plugin. See the IBM Notes and Domino wiki for more information.</td>
</tr>
<tr>
<td>Enabled services: Data, Calendar, Mail</td>
</tr>
</tbody>
</table>

- Save your changes and restart the HTTP task.

NOTE: The above instructions assume you are using Internet Sites. If you are not using Internet Sites, you can enable the calendar service in the server document. See the Domino Data Service documentation for more information on enabling DAS services.
3. Calendar Service Reference

3.1 Discovering Event Feed URLs

To read and create events, you need the URL for a specific event feed. An event feed is really just a reference to the calendar view in a specific user’s mail file. Usually, this is the currently authenticated user's mail file. If the user manages another person's calendar, he can also make requests to the URL representing the delegated calendar.

To get the event feed URLs for the currently authenticated user, you send an HTTP GET request to the following URL:

http://{host}/api/calendar

For example, consider a GET request to:

http://zealand.xyz.com/api/calendar

This returns a response in JSON format:

```
{
  "calendars": [
    {
      "owner": {
        "displayName": "Duke Lawson\Peaks",
        "email": "DukeLawson@xyz.com"
      },
      "links": [
        {
          "rel": "events",
          "href": "http://\zealand.xyz.com/mail/dlawson.nsf/api/calendar/events"
        },
        {
          "rel": "invitations",
          "href": "http://\zealand.xyz.com/mail/dlawson.nsf/api/calendar/invitations"
        }
      ]
    }
  ]
}
```

In this example, the authenticated user, Duke Lawson, has access to one calendar. The event feed URL for his primary calendar is http://zealand.xyz.com/mail/dlawson.nsf/api/calendar/events. The feed for unprocessed invitations is http://zealand.xyz.com/mail/dlawson.nsf/api/calendar/invitations (see Section 3.7.2). If Duke Lawson has access to other calendars, there would be additional entries in the calendars array. (Listing delegated event feeds is not yet implemented.)

All of the above assumes the request is an authenticated request. This could mean the request includes an Authorization header with basic authentication credentials or the request could include a Cookie header with session authentication credentials. The Domino web engine might be configured for other authentication schemes including SAML and SPNEGO. For more information on authenticating with the Domino web engine see the relevant IBM documentation.

If a client sends an unauthenticated request, the service cannot establish the user’s identity and will return an authentication challenge.

3.2 Reading Events

3.2.1 Introduction to Reading Events

To read the events in a feed, you send an HTTP GET request to the feed URL. For example:

http://zealand.xyz.com/mail/dlawson.nsf/api/calendar/events

This returns events in the default representation -- JSON format -- sorted by date. By default, the calendar service returns the first 50 events with a start date between now (the time of the request) and...
one year from now. (See section 3.2.2 for information on specifying a different date range. See section 3.2.4 for information on paging through large event feeds.) Since the data comes from the calendar view, only summary data is returned. See section A.1 for a sample response in JSON format.

To read the events in iCalendar format, you send an HTTP GET request and append `format=icalendar` to the URL. For example:

```
http://zealand.xyz.com/mail/dlawson.nsf/api/calendar/events?format=icalendar
```

See section A.2 for a sample response in iCalendar format.

### 3.2.2 Reading a Range of Events

To read a list of events in a date range, you send an HTTP GET request and append the `since` and `before` parameters to the feed URL. For example:

```
http://zealand.xyz.com/mail/dlawson.nsf/api/calendar/events?since=2010-12-01T00:00:00Z&before=2010-12-31T00:00:00Z
```

This returns the first 50 events that start between 01-Dec-2010 and 31-Dec-2010 inclusive. Date/time values are specified in ISO8601 format. You can also specify just `since` or just `before` and you can specify the `format` parameter as described in section 3.2.1.

If there are no events in the specified range, the service returns HTTP 200 (OK) without a Content-Type header or a response body. If the specified range is illegal (e.g. `before < since`), the service returns HTTP 400 (Bad request). For more information on error responses, see section 3.8.

### 3.2.3 Reading a Single Event

To read a single event, you send an HTTP GET request to the event URL. The format of the event URL is similar to the event feed URL (see section 3.2.1). It just has the event's UID appended to the path. For example:

```
http://zealand.xyz.com/mail/dlawson.nsf/api/calendar/events/839355989EC30000852577BB0050F06A
```

This returns the event in the default representation -- JSON format. The entire event is returned including both summary and non-summary data. For example, for a meeting, the description and entire attendee list is included.

To read a single event in iCalendar format, you can include the format parameter as described in section 3.2.1.

### 3.2.4 Advanced Event Feed Parameters

Section 3.2.2 describes the `since` and `before` parameters used when reading a range of events from the event feed. There are additional parameters you can use to page through large event feeds, search for events matching specific criteria, filter the fields returned for each event, and more. Table 3.2.1 lists these advanced parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Examples</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>Integer &gt; 0</td>
<td>count=25</td>
<td>Maximum number of events to return. The default count is 50 events. The <code>count</code> and <code>ps</code> parameters are mutually exclusive.</td>
</tr>
<tr>
<td>fields</td>
<td>See table 3.2.2</td>
<td>fields=[&quot;start&quot;,&quot;end&quot;] fields=[&quot;location&quot;]</td>
<td>Filters the fields (or properties) returned for each event. By default,</td>
</tr>
</tbody>
</table>
Table 3.2.2 lists the values that can be used with the fields parameter. By default, the fields parameter filters the JSON properties in the second column. When the fields parameter is combined with format=icalendar, it affects the iCalendar properties in the third column.

<table>
<thead>
<tr>
<th>Value</th>
<th>JSON Property</th>
<th>iCalendar Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>categories</td>
<td>categories</td>
<td>CATEGORIES</td>
</tr>
<tr>
<td>class</td>
<td>class</td>
<td>CLASS</td>
</tr>
<tr>
<td>end</td>
<td>end</td>
<td>DTEND</td>
</tr>
<tr>
<td>location</td>
<td>location</td>
<td>LOCATION</td>
</tr>
<tr>
<td>priority</td>
<td>priority</td>
<td>PRIORITY</td>
</tr>
<tr>
<td>recurrenceid</td>
<td>recurrenceid</td>
<td>RECURRENCE-ID</td>
</tr>
<tr>
<td>sequence</td>
<td>sequence</td>
<td>SEQUENCE</td>
</tr>
<tr>
<td>start</td>
<td>start</td>
<td>DTSTART</td>
</tr>
<tr>
<td>status</td>
<td>status</td>
<td>STATUS</td>
</tr>
<tr>
<td>summary</td>
<td>summary</td>
<td>SUMMARY</td>
</tr>
<tr>
<td>transparency</td>
<td>transparency</td>
<td>TRANSP</td>
</tr>
<tr>
<td>x-lotus-apptype</td>
<td>x-lotus-apptype</td>
<td>X-LOTUS-APPTYPE</td>
</tr>
<tr>
<td>x-lotus-noticetype</td>
<td>x-lotus-noticetype</td>
<td>X-LOTUS-NOTICE-TYPE</td>
</tr>
<tr>
<td>x-lotus-onlinemeeting-url</td>
<td>x-lotus-onlinemeeting-url</td>
<td>X-LOTUS-ONLINEMEETING-URL</td>
</tr>
<tr>
<td>x-lotus-organizer</td>
<td>x-lotus-organizer</td>
<td>X-LOTUS-ORGANIZER</td>
</tr>
<tr>
<td>x-lotus-room</td>
<td>x-lotus-room</td>
<td>X-LOTUS-ROOM</td>
</tr>
</tbody>
</table>

When working with large event feeds, you can use the count, page, ps and si parameters described in table 3.2.1. Often you will also want to inspect the Content-Range header in the HTTP response. Let’s say there are 135 events between January 1 and June 30. Your first request is for:

http://zealand.xyz.com/mail/dlawson.nsf/api/calendar/events?since=2012-01-01T00:00:00Z&before=2012-06-30T00:00:00Z

By default the calendar service returns 50 events per page, so the response includes a Content-Range header like this:
In other words, the response includes events 0 through 49. To get the next 50 events, you could send a request for the second page:

```
http://zealand.xyz.com/mail/dlawson.nsf/api/calendar/events?since=2012-01-01T00:00:00Z&before=2012-06-30T00:00:00Z&page=1
```

Now the calendar service returns the next 50 events with a Content-Range header like this:

```
Content-Range: 50-99
```

Finally, you request the third page:

```
http://zealand.xyz.com/mail/dlawson.nsf/api/calendar/events?since=2012-01-01T00:00:00Z&before=2012-06-30T00:00:00Z&page=2
```

This time the calendar service returns the 35 remaining events with a Content-Range header like this:

```
Content-Range: 100-134
```

### 3.3 Creating Events

To create a new event, you send an HTTP POST request to the event feed URL. The body of the request must include a valid representation of the event. To indicate how the event is represented, you must also include a `Content-Type` header. For example:

```
POST /mail/dlawson.nsf/api/calendar/events HTTP/1.1
Host: zealand.xyz.com
Content-Type: application/json

{
   "events": [
      {
         "summary": "Appointment 1",
         "location": "Location 1",
         "start": {
            "date": "2010-12-01",
            "time": "13:00:00",
            "utc": true
         },
         "end": {
            "date": "2010-12-01",
            "time": "14:00:00",
            "utc": true
         }
      }
   ]
}
```

To create the same event in iCalendar format, you send a request like this:

```
POST /mail/dlawson.nsf/api/calendar/events HTTP/1.1
Host: zealand.xyz.com
Content-Type: text/calendar

BEGIN:VCALENDAR
VERSION:2.0
METHOD:PUBLISH
BEGIN:VEVENT
SUMMARY:Appointment 1
LOCATION:Location 1
```

April 28, 2013 - 6 - Version 0.01
If the event is successfully created, the service returns HTTP 201 (Created). The response body includes a representation of the event including any fields added by the service (like UID). The response also includes an HTTP Location header indicating the URL of the newly created event.

If the event cannot be created, the service returns a different HTTP status -- for example, 400 (Bad request). For more information on error responses, see section 3.8.

3.4 Updating Events

To change an existing event, you send an HTTP PUT request to the event URL. The body of the request must include a valid representation of the event. To indicate how the event is represented, you must also include a Content-Type header. For example:

```plaintext
PUT /mail/dlawson.nsf/api/calendar/events/83935598...0050F06A HTTP/1.1
Host: zealand.xyz.com
Content-Type: text/calendar
BEGIN:VCALENDAR
VERSION:2.0
METHOD:PUBLISH
BEGIN:VEVENT
UID:839355989EC30000852577BB0050F06A
SUMMARY:Appointment 1
LOCATION:Now we are over here
DESCRIPTION:Don’t forget the bagels
DTSTART:20101201T130000Z
DTEND:20101201T140000Z
END:VEVENT
END:VCALENDAR
```

If the event is successfully updated the service returns HTTP 200 (OK). The response body includes a representation of the event including any fields omitted in the original request. If the event cannot be updated, the service returns a different HTTP status -- for example, 400 (Bad request).

When you use PUT to update an event and you do not include a DESCRIPTION property, the calendar service preserves the existing description including attachments. You can override this behavior by including the URL parameter literally=true.

For example, consider this representation of an event. Notice it does not include a DESCRIPTION property:

```plaintext
BEGIN:VCALENDAR
VERSION:2.0
METHOD:PUBLISH
BEGIN:VEVENT
UID:839355989EC30000852577BB0050F06A
SUMMARY:Appointment 1
LOCATION:Now we are over here
DTSTART:20101201T130000Z
DTEND:20101201T140000Z
END:VEVENT
END:VCALENDAR
```
If you PUT this representation without the \texttt{literally} parameter, the calendar service preserves the existing description. On the other hand, you can force removal of the description by putting the same iCalendar data to this URL:

\begin{verbatim}
http://zealand.xyz.com/mail/dlawson.nsf/api/calendar/events/839355989EC30000852577BB0050F06A?literally=true
\end{verbatim}

\textbf{WARNING:} When you specify \texttt{literally=true}, the calendar service also removes attachments.

### 3.5 Deleting Events

To delete an event, you send an HTTP DELETE request to the event URL. If the event is successfully deleted, the service returns HTTP 200 (OK).

### 3.6 Recurring Events

To introduce you to the calendar service, the preceding sections have limited the discussion to non-recurring events. There are special considerations for recurring events. For example, sometimes you want operate on a single instance of a recurring event; other times you want to operate on multiple instances or the event as a whole. This section discusses how the service handles recurring events.

#### 3.6.1 Reading a Recurring Event

Consider what happens when you send an HTTP GET request to the following URL:

\begin{verbatim}
http://zealand.xyz.com/mail/dlawson.nsf/api/calendar/events/74509C08C8DECCB48525788800640D3D-Lotus_Notes_Generated?format=icalendar
\end{verbatim}

If the URL refers to a recurring event, the service returns a response like this (some time zone properties have been removed for clarity):

\begin{verbatim}
BEGIN:VCALENDAR
X-LOTUS-CHARSET:UTF-8
VERSION:2.0
PRODID:-//Lotus Development Corporation//NONSGML Notes 8.5.3//EN_S
BEGIN:VTIMEZONE
TZID:Eastern
BEGIN:STANDARD
...
END:STANDARD
BEGIN:DAYLIGHT
...
END:STANDARD
BEGIN:DAYLIGHT
...
BEGIN:VEVENT
DTSTART;TZID="Eastern":20110101T090000
DTEND;TZID="Eastern":20110101T100000
TRANSP:OPAQUE
RRULE:FREQ=WEEKLY;COUNT=4;BYDAY=SA
DTSTAMP:20110506T181710Z
CLASS:PUBLIC
SUMMARY:Recurring appointment
LOCATION:Undisclosed location
UID:74509C08C8DECCB48525788800640D3D-Lotus_Notes_Generated
X-LOTUS-NOTESVERSION:2
X-LOTUS-APPTTYPE:0
END:VEVENT
END:VCALENDAR
\end{verbatim}
In other words, the iCalendar representation is a single event with an \texttt{RRULE} property that specifies the recurrence rule. Sometimes a recurring event has exceptions – event instances that have been modified or removed. In this case the \texttt{VCALENDAR} component will contain more than one \texttt{VEVENT}:

```
BEGIN:VCALENDAR
X-LOTUS-CHARSET:UTF-8
VERSION:2.0
PRODID:-//Lotus Development Corporation//NONSGML Notes 8.5.3//EN_S
BEGIN:VTIMEZONE
TZID:Eastern
BEGIN:STANDARD
...
END:STANDARD
...
END:DST
END:VTIMEZONE
BEGIN:VEVENT
DTSTART;TZID="Eastern":20110107T090000
DTEND;TZID="Eastern":20110107T100000
TRANSP:OPAQUE
RRULE:FREQ=WEEKLY;COUNT=4;BYDAY=FR
DTSTAMP:20110509T140941Z
CLASS:PUBLIC
SUMMARY:Another recurring appointment
LOCATION:Situation room
UID:6365E29A99F8CE658525788B004D2718-Lotus_Notes_Generated
X-LOTUS-NOTESVERSION:2
X-LOTUS-APPTTYPE:0
END:VEVENT
BEGIN:VEVENT
DTSTART:20110114T150000Z
DTEND:20110114T160000Z
TRANSP:OPAQUE
RECURRENCE-ID:20110114T140000Z
DTSTAMP:20110509T140941Z
CLASS:PUBLIC
DESCRIPTION:This is the exception
SUMMARY:Another recurring appointment
LOCATION:Situation room
UID:6365E29A99F8CE658525788B004D2718-Lotus_Notes_Generated
X-LOTUS-NOTESVERSION:2
X-LOTUS-APPTTYPE:0
END:VEVENT
END:VCALENDAR
```

The second event includes a \texttt{RECURRENCE-ID} property to indicate which instance is an exception. See the iCalendar specification for more information on recurrence rules and exceptions. See Appendix A to compare how the same data is represented in JSON format.

### 3.6.2 Reading an Instance of a Recurring Event

To read a single instance of recurring event, you send an HTTP GET request to the \texttt{event instance} URL. The format of the event instance URL is:

```
http://{host}/(filenamepath)/api/calendar/events/(eventid)/(recurrenceid)
```
where `{recurrenceid}` identifies the specific instance you are reading. The value of `{recurrenceid}` matches the value of the `RECURRENCE-ID` property in iCalendar.

For example, you can send a GET request to the following URL:

```
http://zealand.xyz.com/mail/dlawson.nsf/api/calendar/events/74509C08C8DECCB48525788800640D3D-Lotus_Notes_Generated/20110108T140000Z?format=icalendar
```

The service returns a response like this:

```
BEGIN:VCALENDAR
X-LOTUS-CHARSET:UTF-8
VERSION:2.0
PRODID:-//Lotus Development Corporation//NONSGML Notes 8.5.3//EN_S
BEGIN:VTIMEZONE
TZID:Eastern
BEGIN:STANDARD
...
END:STANDARD
BEGIN:DAYLIGHT
...
END:DAYLIGHT
END:VTIMEZONE
BEGIN:VEVENT
DTSTART:20110108T140000Z
DTEND:20110108T150000Z
TRANSP:OPAQUE
RECURRENCE-ID:20110108T140000Z
DTSTAMP:20110506T182840Z
CLASS:PUBLIC
SUMMARY:Recurring appointment
LOCATION:Undisclosed location
UID:74509C08C8DECCB48525788800640D3D-Lotus_Notes_Generated
X-LOTUS-NOTESVERSION:2
X-LOTUS-APPTTYPE:0
END:VEVENT
END:VCALENDAR
```

Compare this with the response in section 3.6.1. It includes the same `UID`, but the `RECURRENCE-ID` property uniquely identifies this instance. Also, notice the response includes `DTSTART` and `DTEND` properties corresponding to the specific event instance.

### 3.6.3 Creating a Recurring Event

To create a recurring event, you send an HTTP POST request to the event feed URL. The process is essentially the same as described in section 3.3. The only difference is the body of the request needs to include recurrence information -- for example:

```
POST /mail/dlawson.nsf/api/calendar/events HTTP/1.1
Host: zealand.xyz.com
Content-Type: text/calendar

BEGIN:VCALENDAR
VERSION:2.0
METHOD:PUBLISH
BEGIN:VEVENT
SUMMARY:Recurring appointment
LOCATION:Location 4
DTSTART:20110101T180000Z
```

April 28, 2013

- 10 -

Version 0.01
If the event is successfully created, the service returns HTTP 201 (Created). The response body includes a representation of the event including any fields added by the service (like UID). The response also includes an HTTP Location header indicating the URL of the newly created event.

3.6.4 Updating an Instance of a Recurring Event

To change a single instance of a recurring event, you send an HTTP PUT request to the event instance URL (see section 3.6.2). The request format is the same as described in section 3.4 -- for example:

```
PUT /.../events/815EBF5F786FB79D852577BB0045ECA1/20110101T130000Z HTTP/1.1
Host: zealand.xyz.com
Content-Type: text/calendar

BEGIN:VCALENDAR
VERSION:2.0
METHOD:PUBLISH
BEGIN:VEVENT
UID:815EBF5F786FB79D852577BB0045ECA1
RECURRENCE-ID:20110101T130000Z
SUMMARY:Recurring appointment
LOCATION:Now we are over here
DTSTART:20110101T130000Z
DTEND:20110101T140000Z
END:VEVENT
END:VCALENDAR
```

If the event instance is successfully updated the service returns HTTP 200 (OK). The response body includes a representation of the event instance including any fields omitted in the original request.

3.6.5 Deleting an Instance of a Recurring Event

To delete an instance of a recurring event, you send an HTTP DELETE request to the event instance URL. If the instance is successfully deleted, the service returns HTTP 200 (OK).

You can also delete more than one instance of a recurring event. To do so, you add a recurrenceRange parameter to the URL -- for example:

```
http://zealand.xyz.com/mail/dlawson.nsf/api/calendar/events/815EBF5F786FB79D852577BB0045ECA1/20110101T130000Z?recurrenceRange=future
```

When you send a DELETE request to this URL, the service deletes the specified instance and all future instances. The following are the legal values for the recurrenceRange parameter:

- **this** - updates just the specified instance (this is the default value)
- **all** - updates all instances in the recurrence set
- **future** - updates this and all future instances
- **previous** - updates this and all previous instances

3.7 Scheduling Workflow

The preceding sections have covered the basics of creating, reading, updating and deleting events with the calendar service. Until now we have omitted any mention of scheduling workflow. This is fine for appointments, reminders, all day events and anniversaries. There is no scheduling workflow associated
these event types, but meetings are different. If you are the organizer of a meeting, you need to control what notices are sent as you create, update and delete the meeting. If you are an attendee of a meeting, you need to be able to send workflow requests including accept, decline and counter. The calendar service supports scheduling workflow for both organizers and attendees.

3.7.1 Reading a Calendar Notice

An event is an object that is already on a calendar. A calendar notice is a separate object that contains some information relevant to an event. For example, an invitation is one type of notice. When the participant accepts an invitation, an event is added to his calendar. A reschedule is another type of notice. When the participant accepts a reschedule, the event on his calendar is moved to a new date or time.

Some notices are intended for the organizer of an event. For example, a counter-proposal is one type of notice sent to an organizer. When the organizer accepts a counter-proposal, the event on his calendar is moved to a new date or time and reschedule notices are sent to all attendees.

To read a calendar notice, you send an HTTP GET request to the notice URL. The format of a notice URL is:

http://{host}/{filenamepath}/api/calendar/notices/{noticeid}

For example, the following is a valid notice URL:

http://zealand.xyz.com/mail/dlawson.nsf/api/calendar/notices/4004A308157DD3EB852578F7005DFB75

When you send a GET request to a notice URL, the service returns a representation of the notice in JSON format – for example:

```json
{
  "scheduleMethod": "request",
  "timezones": [
    {
      "tzid": "Eastern",
      "standard": {
        "start": {
          "date": "1950-11-05",
          "time": "02:00:00"
        },
        "offsetFrom": "-0400",
        "offsetTo": "-0500",
        "recurrenceRule": "FREQ=YEARLY;BYMONTH=11;BYDAY=1SU;BYHOUR=2;BYMINUTE=0"
      },
      "daylight": {
        "start": {
          "date": "1950-03-12",
          "time": "02:00:00"
        },
        "offsetFrom": "-0500",
        "offsetTo": "-0400",
        "recurrenceRule": "FREQ=YEARLY;BYMONTH=3;BYDAY=2SU;BYHOUR=2;BYMINUTE=0"
      }
    }
  ],
  "events": [
    {
      "id": "4004A308157DD3EB852578F7005DFB75-Lotus_Notes_Generated",
      "summary": "Staff meeting",
      "location": "Room 202",
      "description": "Let's get together for a quick status update",
      "start": {
        "date": "2011-09-09",
        "time": "13:00:00",
        "tzid": "Eastern"
      },
      "end": {
```
The JSON representations of notices and events are very similar. However, a notice includes two additional properties: `scheduleMethod` and `x-lotus-noticetype`.

The `scheduleMethod` property corresponds to the iCalendar `METHOD` property. It tells you roughly what kind of notice follows. For example:

- **request** indicates the notice is an invitation or reschedule.
- **reply** indicates the notice is a response to a request from the organizer (e.g. accept, decline, tentative).
- **cancel** indicates the notice is a cancellation from the organizer.

For the complete list of method types, see RFC 2446 (http://www.rfc-editor.org/rfc/pdfrfc/rfc2446.txt.pdf).

The `x-lotus-noticetype` property is a more precise indicator of the notice type. For example:

- **I** indicates the notice is an invitation.
- **A** indicates an accept notice.
- **R** indicates a decline notice
- **P** indicates a tentative acceptance.

For the complete list of notice types, see Notes and Domino Calendaring & Scheduling Schema.

To read the notice in iCalendar format, you send an HTTP GET request and append `format=icalendar` to the URL. For example:

http://zealand.xyz.com/mail/dlawson.nsf/api/calendar/notices/4004A308157DD3EB852578F7005DFB75?format=icalendar

The following is an example of a notice represented in iCalendar format:

```
BEGIN:VCALENDAR
X-LOTUS-CHARSET:UTF-8
VERSION:2.0
```
NOTE: As defined by the calendar service, notices are immutable resources. You can read them, but unlike events, you cannot create, update and delete them. On the other hand, the service does let you can take an action on a notice. For example, you can accept a reschedule. See section 3.7.4 for more information on actions.

3.7.2 Reading the List of New Invitations

To read the list of new invitations, you send an HTTP GET request to the invitations URL. The format of the invitations URL is:

http://{host}/{filenamepath}/api/calendar/invitations

The calendar service responds with a list of invitations in JSON format:

```json
{  "notices": [    {      "summary":"All hands meeting",      "scheduleMethod":"request",      "href":"http://xyz.com/.../api/calendar/notices/C6245540BD606AA8525788500627308",    }  ]}
```
In other words, the service returns an array of JSON objects. Each object in the array corresponds to a single unprocessed invitation. Invitations include notices of delegated meetings. They do not include reschedules.

You can think of each object in the array as a summary of the complete notice. To read the complete notice, use the notice URL as defined in the `href` property and follow the instructions in section 3.7.1.

**NOTE:** Unlike other resources, you cannot ask for an iCalendar representation of the list of new invitations. The iCalendar standard doesn't include such a concept.

When you send a GET request to the invitations URL, the calendar service by default returns only the new invitations that are considered current (at least one instance starts within the last day or in the future). You can include URL parameters for finer grained control of the result set. Table 3.7.1 lists these additional parameters.

### Table 3.7.1. Invitations Feed Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Event id</td>
<td>id=4004A308157DD3EB852578F7005DFB75</td>
<td>Returns only the invitations associated with the given event ID.</td>
</tr>
<tr>
<td>receivedsince</td>
<td>ISO8601 date</td>
<td>receivedsince=2010-12-31T00:00:00Z</td>
<td>Returns only the invitations received since the given date.</td>
</tr>
<tr>
<td>since</td>
<td>ISO8601 date</td>
<td>since=2010-12-31T00:00:00Z</td>
<td>Returns only the invitations for events with a start date greater than the given date.</td>
</tr>
</tbody>
</table>

### 3.7.3 Reading a List of Unapplied Notices

When a calendar application opens an existing event, it often wants to know whether there are any unapplied notices related to the event. For example, when opening a meeting on the participant’s calendar, the application wants to know if there are any reschedule notices pending for the meeting. To read a list of unapplied notices, you send a GET request to the event notices URL:

```
http://{host}/{filenamepath}/api/calendar/events/{eventid}/notices
```

The calendar service responds with a list of notices in JSON format as described in section 3.7.2. In this case, the notices apply only to the single event specified by `{eventid}`.

### 3.7.4 Actions on Notices

The calendar service defines a special resource for taking actions on notices. To take an action on a notice you send an HTTP PUT request to a URL in this format:

```
http://{host}/{filenamepath}/api/calendar/notices/{noticeid}/action?type={action}
```

The list of legal values for `{action}` are:

- accept
- cancel
- confirm
• counter
• decline
• delegate
• delete
• processall
• requestInfo
• tentative

NOTE: Some of the above actions apply only to certain notices. For example, you cannot cancel an invitation. Also, some actions apply only to events – not notices. See section 3.7.6 for more information on each of the above actions.

The request can also include extra data in the response body. To send extra data, you include a Content-Type header and a body in JSON format – for example:

```plaintext
PUT /mail/dlawson.nsf/api/calendar/notices/E476...12F8/action?type=accept
Host: zealand.xyz.com
Content-Type: application/json

{
   "comments":"I’ll bring the bagels."
}
```

The response body should be a single JSON object with one or more of the following properties:

• comments specifies the comments that should be included with any outbound notices associated with the action.

• delegateTo specifies the person a meeting is being delegated to. This only applies to the delegate action.

• counterStart specifies the start date/time for a counter proposal. This only applies to the counter action.

• counterEnd specifies the end date/time for a counter proposal. This only applies to the counter action.

• keepInformed is a Boolean value that specifies whether the attendee should be kept informed of updates to the corresponding event. This only applies to the decline and delegate actions.

If the request succeeds, the calendar service returns an HTTP status code of 200. At this point the notice has been applied. It should no longer appear in the list of new invitations (section 3.7.2) or unapplied notices for the corresponding event (section 3.7.3). You may read the notice as described in section 3.7.1, but you should not attempt to send another action for the notice. This can produce unexpected results.

NOTE: The notice action resource accepts only PUT requests. You cannot GET or DELETE a notice action.

3.7.5 Actions on Events

The calendar service also defines a special resource for taking actions on existing events. To take an action on an event you send an HTTP PUT request to a URL in this format:

http://{host}/{filenamemxpath}/api/calendar/events/{eventid}/action?type={action}

The request can also include a Content-Type header and a body in JSON format – for example:

```plaintext
PUT /mail/dlawson.nsf/api/calendar/events/F576...12E9/action?type=counter
Host: zealand.xyz.com
```
The response body should be a single JSON object as described in the section 3.7.4.

You can also take an action on a single instance of a recurring event. To do this you send an HTTP PUT request to a URL in this format:

```
http://{host}/{filenamepath}/api/calendar/events/{eventid}/{recurrenceid}/action?type={action}
```

For example, as the participant of a recurring meeting you can delegate just one instance to a different person. You can also specify a recurrence range as defined in section 3.6.5. For example, you can delegate this and all future instances, by appending `recurrenceRange=future` to the URL:

```
http://{host}/{filenamepath}/api/calendar/events/{eventid}/{recurrenceid}/action?type=delegate&recurrenceRange=future
```

### 3.7.6 Action Cross Reference

The calendar service supports the actions listed in table 3.7.2.

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
<th>Relevant Notice Types</th>
<th>Works for Events</th>
<th>Relevant JSON Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>accept</td>
<td>Applies the notice and sends relevant workflow.</td>
<td>Confirmation, counter, information request, invitation, reschedule</td>
<td>No</td>
<td>comments</td>
</tr>
<tr>
<td>cancel</td>
<td>Cancels a meeting and sends relevant workflow.</td>
<td>none</td>
<td>Yes</td>
<td>comments</td>
</tr>
<tr>
<td>counter</td>
<td>Proposes a new time for a meeting.</td>
<td>Invitation, reschedule</td>
<td>Yes</td>
<td>comments, counterStart, counterEnd</td>
</tr>
<tr>
<td>decline</td>
<td>Declines the meeting or counterproposal and sends relevant workflow</td>
<td>Counter, invitation, reschedule</td>
<td>Yes</td>
<td>comments, keepInformed</td>
</tr>
<tr>
<td>delegate</td>
<td>Delegates a meeting to another person</td>
<td>Invitation, reschedule</td>
<td>Yes</td>
<td>comments, delegateTo, keepInformed</td>
</tr>
<tr>
<td>delete</td>
<td>Deletes the object without sending any workflow.</td>
<td>none</td>
<td>Yes</td>
<td>none</td>
</tr>
<tr>
<td>tentative</td>
<td>Tentatively accepts a</td>
<td>Invitation, reschedule</td>
<td>No</td>
<td>comments</td>
</tr>
</tbody>
</table>
### 3.7.7 Controlling Implicit Scheduling

By default, the calendar service sends out workflow notices automatically for certain operations. For example:

- When you create an event with attendees, it automatically sends invitations to all attendees.
- When you update an event in the organizer’s calendar, it automatically sends out either a reschedule or an update notice.
- When you delete an event in the organizer’s calendar, it automatically sends a cancel notice.
- When you delete an event in an attendee’s calendar, it automatically sends a decline notice.

In each of these cases, you can disable implicit scheduling, by simply adding a `workflow=false` parameter to the request URL.

As an example, Section 3.3 describes how to create a new event, by sending a POST request to an event feed URL. Assuming the event has attendees, you can disable implicit workflow by using the following URL in your POST request:

```
http://zealand.xyz.com/mail/dlawson.nsf/api/calendar/events
```

If you include `workflow=false` in the URL and the operation does not result in any workflow, the parameter is simply ignored.

When updating an existing event, you often want the implied workflow notice to be sent but you want a comment to be included with the notice. For example, if you reschedule an event on the organizer’s calendar, you might want to add a comment to the reschedule notice. To do this you add `workflowcomment` parameter to the request URL. For example:

```
http://{host}/mail/dlawson.nsf/api/calendar/events/{eventid}?workflowcomment='Sorry for the last minute time change.'
```

Sending a PUT request to the above URL would update the event, send out the implied notices, and include the specified workflow comment. If you include `workflowcomment` in the URL and the operation does not result in any workflow, the parameter is simply ignored.

### 3.8 Error Responses

When you send a request to the calendar service it may return an error response. The HTTP status code of the response is the primary indicator an error has occurred. A normal response is indicated by HTTP status code 200, or in some cases, HTTP status code 201. An error response is indicated by one of the following codes:

- 400 (Bad request) – the calendar service cannot interpret your request. This could indicate there was an error parsing a URL parameter or the JSON object in the request body.
- 401 (Not authorized) – the request doesn’t include an authorization or the authenticated user doesn’t have access to resource (event, event instance, or notice).
- 403 (Forbidden) – might indicate the calendar service is not enabled on this server.
- 404 (Not found) – the calendar service couldn’t find the resource (event, event instance, or notice).
- 409 (Conflict) – the calendar service couldn’t complete the request because there is a conflict between the data on the calendar and the data you sent.
- 500 (Internal server error) – there was an internal error processing your request.

Often the calendar service includes a JSON object in the response body. For example, the following JSON object is returned when you attempt to read an event that doesn’t exist:

```json
{
    "code": 404,
    "text": "Not Found",
    "message": "Error reading event",
    "cserror": 1031,
    "type": "text",
    "data": "com.ibm.domino.calendar.store.StoreException: Error reading event\n\ncom.ibm.domino.calendar.dbstore.NotesCalendarStore.getEvent(NotesCalendarStore.java:130) ..."
}
```

The following JSON properties are included in the error response:

- **code** – the HTTP status code
- **text** – a description of the HTTP status code
- **cserror** – a calendar service error code. See table 3.8.1 for a list of valid error codes.
- **message** – a description of the calendar service error
- **data** – more detailed information about the error
- **type** – the type of the data property

**Table 3.8.1. Calendar service error codes**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Applies to</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>General calendar store error</td>
<td>Any request</td>
</tr>
<tr>
<td>1025</td>
<td>Invalid Recurrence-ID</td>
<td>A request that includes a recurrence ID in the URL, but the recurrence ID is not in the right format</td>
</tr>
<tr>
<td>1026</td>
<td>Error sending notices</td>
<td>A request that results in outgoing notices, but the notices cannot be sent</td>
</tr>
<tr>
<td>1027</td>
<td>Newer version exists</td>
<td>A request that updates an entry, but the update would overwrite a newer version.</td>
</tr>
<tr>
<td>1028</td>
<td>Unsupported action</td>
<td>A request that applies an action to an event or notice, but the action is not supported for the particular entry.</td>
</tr>
<tr>
<td>1029</td>
<td>The invitation for this meeting has not been accepted</td>
<td>A request that attempts to apply an action to an event before it has been accepted</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
<td>Message</td>
</tr>
<tr>
<td>-------</td>
<td>-------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>1030</td>
<td>Action cannot be performed because it would overwrite personal changes.</td>
<td>A request that attempts to apply an action to an event, but the action would overwrite personal changes.</td>
</tr>
<tr>
<td>1031</td>
<td>Identifier not found</td>
<td>A request that includes an ID or recurrence ID that doesn’t exist.</td>
</tr>
<tr>
<td>1032</td>
<td>Entry already exists</td>
<td>A request that creates a new event with a UID, but the UID already exists</td>
</tr>
</tbody>
</table>
Appendix A. Event Representations Cross Reference

A.1 JSON Representation

The calendar service uses a JSON "schema" designed to preserve all the properties of iCalendar. It is also designed to be easy to use from web applications implemented in JavaScript.

This section provides several examples of events in JSON format. Each example has an equivalent representation in iCalendar format as described in section A.2. For example, sections A.1.1 and A.2.1 show two different representations for the same data.

A.1.1 Single Event in JSON Format

The following is an example of a single event in JSON format:

```json
{
    "timezones": [
        {
            "tzid": "Eastern",
            "standard": {
                "start": {
                    "date": "1950-11-05",
                    "time": "02:00:00"
                },
                "offsetFrom": "-0400",
                "offsetTo": "-0500",
                "recurrenceRule": "FREQ=YEARLY;BYMONTH=11;BYDAY=1SU;BYHOUR=2;BYMINUTE=0"
            },
            "daylight": {
                "start": {
                    "date": "1950-03-12",
                    "time": "02:00:00"
                },
                "offsetFrom": "-0500",
                "offsetTo": "-0400",
                "recurrenceRule": "FREQ=YEARLY;BYMONTH=3;BYDAY=2SU;BYHOUR=2;BYMINUTE=0"
            }
        }
    ],
    "events": [
        {
            "id": "6ABE2BD358613EAD85257888006A5C6F-Lotus_Notes_Generated",
            "summary": "Appointment example",
            "location": "Area 51",
            "description": "This is a sample description",
            "start": {
                "date": "2011-01-03",
                "time": "09:00:00",
                "tzid": "Eastern"
            },
            "end": {
                "date": "2011-01-03",
                "time": "10:00:00",
                "tzid": "Eastern"
            },
            "class": "public",
            "transparency": "opaque"
        }
    ]
}
```

The above example shows a timezone and event array both of which contain a single object. The event's start and end properties refer to the time zone by tzid.
The same event can be represented without the timezone array. In this case all date/time values are represented in UTC:

```
{  "events": [  {
      "id":"6ABE2BD358613EAD85257888006A5C6F-Lotus_Notes_Generated",
      "summary":"Appointment example",
      "location":"Area 51",
      "description":"This is a sample description",
      "start": {  
          "date":"2011-01-03",
          "time":"14:00:00",
          "utc":true
      },
      "end": {  
          "date":"2011-01-03",
          "time":"15:00:00",
          "utc":true
      },
      "class":"public",
      "transparency":"opaque"
  }]}
```

### A.1.2 Multiple Events in JSON Format

The following is an example of multiple events in JSON format. This example includes only summary properties for each event. Non-summary properties like description are omitted:

```
{  "events": [  {
      "id":"FD0219D2B25EF916852578880064531E_3-Lotus_ReadRange_Generated",
      "summary":"Recurring appointment",
      "location":"Undisclosed location",
      "start": {  
          "date":"2011-01-01",
          "time":"14:00:00",
          "utc":true
      },
      "end": {  
          "date":"2011-01-01",
          "time":"15:00:00",
          "utc":true
      },
      "class":"public",
      "sequence":0
  },  
  {
      "id":"6ABE2BD358613EAD85257888006A5C6F_2-Lotus_ReadRange_Generated",
      "summary":"Appointment example",
      "location":"Area 51",
      "start": {  
          "date":"2011-01-03",
          "time":"14:00:00",
          "utc":true
      },
      "end": {  
          "date":"2011-01-03",
          "time":"15:00:00",
          "utc":true
      },
      "class":"public",
      "sequence":0
  },  
  {
      "id":"906CEF2CB4DCCB2C85257888006A9059_1-Lotus_ReadRange_Generated",
      "summary":"Meeting example",
      "location":
```
A.1.3 Event with Attendees in JSON Format

The following example shows a single event with an organizer property and an array of attendee properties:

```json
{
  "timezones": [
    {
      "tzid": "Eastern",
      "standard": {
        "start": {
          "date": "1950-11-05",
          "time": "02:00:00"
        },
        "offsetFrom": "-0400",
        "offsetTo": "-0500",
        "recurrenceRule": "FREQ=YEARLY;BYMONTH=11;BYDAY=1SU;BYHOUR=2;BYMINUTE=0"
      },
      "daylight": {
        "start": {
          "date": "1950-03-12",
          "time": "02:00:00"
        },
        "offsetFrom": "-0500",
        "offsetTo": "-0400",
        "recurrenceRule": "FREQ=YEARLY;BYMONTH=3;BYDAY=2SU;BYHOUR=2;BYMINUTE=0"
      }
    }
  ],
  "events": [
    {
      "id": "906CEF2CB4DCCB2C8525788806A9059-Lotus_Notes_Generated",
      "summary": "Meeting example",
      "location": "Your office",
      "description": "Review project plan",
      "start": {
        "date": "2011-01-04",
        "time": "09:00:00",
        "tzid": "Eastern"
      },
      "end": {
        "date": "2011-01-04",
        "time": "10:00:00",
        "tzid": "Eastern"
      },
      "class": "public",
      "transparency": "opaque",
      "sequence": 0,
      "attendees": [{
        "displayName": "Duke Lawson\Peaks",
        "email": "DukeLawson@xyz.com",
        "status": "accepted"
      }]
    }
  ]
}
```
A.1.4 Recurring Event in JSON Format

The following is an example of a recurring event in JSON format. The `start` and `end` properties are for the first instance of the recurring event. The `recurrenceRule` property specifies the recurrence rule. In this case the event recurs weekly on Saturday for four weeks:

```json
{
    "timezones": [
        {
            "tzid": "Eastern",
            "standard": {
                "start": {
                    "date": "1950-11-05",
                    "time": "02:00:00"
                },
                "offsetFrom": "-0400",
                "offsetTo": "-0500",
                "recurrenceRule": "FREQ=YEARLY;BYMONTH=11;BYDAY=1SU;BYHOUR=2;BYMINUTE=0"
            },
            "daylight": {
                "start": {
                    "date": "1950-03-12",
                    "time": "02:00:00"
                },
                "offsetFrom": "-0500",
                "offsetTo": "-0400",
                "recurrenceRule": "FREQ=YEARLY;BYMONTH=3;BYDAY=2SU;BYHOUR=2;BYMINUTE=0"
            }
        }
    ],
    "events": [
        {
            "id": "74509C08C8DECCB48526788800640D3D-Lotus_Notes_Generated",
            "summary": "Recurring appointment",
            "location": "Undisclosed location",
            "start": {
                "date": "2011-01-01",
                "time": "09:00:00",
                "tzid": "Eastern"
            },
            "end": {
                "date": "2011-01-01",
                "time": "10:00:00",
                "tzid": "Eastern"
            },
            "recurrenceRule": "FREQ=WEEKLY;COUNT=4;BYDAY=SA",
            "class": "public",
            "transparency": "opaque"
        }
    ]
}
```
A.1.5 Recurring Event with an Exception in JSON Format

Sometimes a recurring event has exceptions — event instances that have been modified or removed. In this case the events array will contain more than one element:

```json
{
  "timezones": [
    {
      "tzid":"Eastern",
      "standard": {
        "start": {
          "date":"1950-11-05",
          "time":"02:00:00"
        },
        "offsetFrom":"-0400",
        "offsetTo":"-0500",
        "recurrenceRule":"FREQ=YEARLY;BYMONTH=11;BYDAY=1SU;BYHOUR=2;BYMINUTE=0"
      },
      "daylight": {
        "start": {
          "date":"1950-03-12",
          "time":"02:00:00"
        },
        "offsetFrom":"-0500",
        "offsetTo":"-0400",
        "recurrenceRule":"FREQ=YEARLY;BYMONTH=3;BYDAY=2SU;BYHOUR=2;BYMINUTE=0"
      }
    }
  ],
  "events": [
    {
      "id":"6365E29A99F8CE658525788B004D2718-Lotus_Notes_Generated",
      "summary":"Another recurring appointment",
      "location":"Situation room",
      "start": {
        "date":"2011-01-07",
        "time":"09:00:00",
        "tzid":"Eastern"
      },
      "end": {
        "date":"2011-01-07",
        "time":"10:00:00",
        "tzid":"Eastern"
      },
      "recurrenceRule":"FREQ=WEEKLY;COUNT=4;BYDAY=FR",
      "class":"public",
      "transparency":"opaque"
    },
    {
      "id":"6365E29A99F8CE658525788B004D2718-Lotus_Notes_Generated",
      "summary":"Another recurring appointment",
      "location":"Situation room",
      "description":"This is the exception",
      "start": {
        "date":"2011-01-14",
        "time":"15:00:00",
        "utc":true
      },
      "end": {
        "date":"2011-01-14",
        "time":"16:00:00",
        "utc":true
      },
      "recurrenceId":"20110114T140000Z",
      "class":"public",
      "transparency":"opaque"
    }
  ]
}
```
In the above example, the first element in the `events` array describes the overall event including its recurrence rule. The second element describes an exception that occurs January 14. The `start`, `end` and `description` properties are different for the exception.

A.1.6 An Instance of a Recurring Event in JSON Format

As described in sections A.1.4 and A.1.5, the service often represents a recurring event as a whole (all event instances). It can also represent a single instance of a recurring event. The example below shows an instance of the same event from section A.1.5. The instance occurs on January 21. It has the same `id` property, but it also has a `recurrenceId` property. The `recurrenceId` property uniquely identifies the instance:

```json
{
    "timezones": [
        {
            "tzid":"Eastern",
            "standard": {
                "date":"1950-11-05",
                "time":"02:00:00"
            },
            "offsetFrom":"-0400",
            "offsetTo":"-0500",
            "recurrenceRule":"FREQ=YEARLY;BYMONTH=11;BYDAY=1SU;BYHOUR=2;BYMINUTE=0"
        },
        {
            "daylight": {
                "start": {
                    "date":"1950-03-12",
                    "time":"02:00:00"
                },
                "offsetFrom":"-0500",
                "offsetTo":"-0400",
                "recurrenceRule":"FREQ=YEARLY;BYMONTH=3;BYDAY=2SU;BYHOUR=2;BYMINUTE=0"
            }
        }
    ],
    "events": [
        {
            "id":"6365E29A99F8CE658525788B004D2718-Lotus_Notes_Generated",
            "summary":"Another recurring appointment",
            "location":"Situation room",
            "start": {
                "date":"2011-01-21",
                "time":"14:00:00",
                "utc":true
            },
            "end": {
                "date":"2011-01-21",
                "time":"15:00:00",
                "utc":true
            },
            "recurrenceId":"20110121T140000Z",
            "class":"public",
            "transparency":"opaque"
        }
    ]
}
```
A.2 iCalendar Representation

This section includes several examples of events represented in iCalendar format. For a detailed description of iCalendar components and properties, see the iCalendar specification.

Each example in this section has an equivalent representation in JSON format as described in section A.1. For example, sections A.1.1 and A.2.1 show two different representations for the same data.

A.2.1 Single Event in iCalendar Format

The following is an example of a single event in iCalendar format:

```
BEGIN:VCALENDAR
X-LOTUS-CHARSET:UTF-8
VERSION:2.0
PRODID:;-//Lotus Development Corporation//NONSGML Notes 8.5.3//EN_S
BEGIN:VTIMEZONE
TZID:Eastern
BEGIN:STANDARD
DTSTART:19501105T020000
TZOFFSETFROM:-0400
TZOFFSETTO:-0500
RRULE:FREQ=YEARLY;BYMINUTE=0;BYHOUR=2;BYDAY=1SU;BYMONTH=11
END:STANDARD
BEGIN:DAYLIGHT
DTSTART:19500312T020000
TZOFFSETFROM:-0500
TZOFFSETTO:-0400
RRULE:FREQ=YEARLY;BYMINUTE=0;BYHOUR=2;BYDAY=2SU;BYMONTH=3
END:DAYLIGHT
END:VTIMEZONE
BEGIN:VEVENT
DTSTART;TZID="Eastern":20110103T090000
DTEND;TZID="Eastern":20110103T100000
TRANSP:OPAQUE
DTSTAMP:20110506T192710Z
CLASS:PUBLIC
DESCRIPTION:This is a sample description
SUMMARY:Appointment example
LOCATION:Area 51
UID:6ABE2BD358613EAD85257888006A5C6F-Lotus_Notes_Generated
X-LOTUS-NOTESVERSION:2
X-LOTUS-APPTTYPE:0
END:VEVENT
END:VCALENDAR
```

The above example shows a VCALENDAR component containing a VTIMEZONE and a VEVENT. The event's DTSTART and DTEND properties refer to the time zone by TZID.

The same event can be represented without the VTIMEZONE component. In this case all date/time values are represented in UTC:

```
BEGIN:VCALENDAR
X-LOTUS-CHARSET:UTF-8
VERSION:2.0
PRODID:;-//Lotus Development Corporation//NONSGML Notes 8.5.3//EN_S
BEGIN:VEVENT
DTSTART;TZID="Eastern":20110103T090000
DTEND;TZID="Eastern":20110103T100000
TRANSP:OPAQUE
DTSTAMP:20110506T192710Z
CLASS:PUBLIC
DESCRIPTION:This is a sample description
SUMMARY:Appointment example
LOCATION:Area 51
UID:6ABE2BD358613EAD85257888006A5C6F-Lotus_Notes_Generated
X-LOTUS-NOTESVERSION:2
X-LOTUS-APPTTYPE:0
END:VEVENT
END:VCALENDAR
```
A.2.2 Multiple Events in iCalendar Format

The following is an example of multiple events in iCalendar format:

```
BEGIN:VCALENDAR
X-LOTUS-CHARSET:UTF-8
VERSION:2.0
BEGIN:VEVENT
DTSTART:20110101T140000Z
DTEND:20110101T150000Z
DTSTAMP:20110506T195351Z
SEQUENCE:0
CLASS:PUBLIC
SUMMARY:Recurring appointment
LOCATION:Undisclosed location
UID:FE0219D2B25EF91685257888064531E_3-Lotus_ReadRange_Generated
X-LOTUS-APPTTYPE:0
END:VEVENT
BEGIN:VEVENT
DTSTART:20110103T140000Z
DTEND:20110103T150000Z
DTSTAMP:20110506T195351Z
SEQUENCE:0
CLASS:PUBLIC
SUMMARY:Appointment example
LOCATION:Area 51
UID:6ABE2BD358613EAD85257888006A5C6F_2-Lotus_ReadRange_Generated
X-LOTUS-APPTTYPE:0
END:VEVENT
BEGIN:VEVENT
DTSTART:20110104T140000Z
DTEND:20110104T150000Z
DTSTAMP:20110506T195351Z
SEQUENCE:0
CLASS:PUBLIC
SUMMARY:Meeting example
LOCATION:Your office
UID:906CEF2CB4DCCB2C85257888006A9059_1-Lotus_ReadRange_Generated
X-LOTUS-APPTTYPE:3
END:VEVENT
END:VCALENDAR
```

The above example includes three individual VEVENT components. Each event contains only summary properties. Non-summary properties like DESCRIPTION are omitted.

A.2.3 Event with Attendees in iCalendar Format

The following example shows a single event with ORGANIZER and ATTENDEE properties:

```
BEGIN:VCALENDAR
X-LOTUS-CHARSET:UTF-8
VERSION:2.0
BEGIN:VTIMEZONE
TZID:Eastern
BEGIN:STANDARD
DTSTART:19501105T020000
TZOFFSETFROM:-0400
TZOFFSETTO:-0500
RRULE:FREQ=YEARLY;BYMINUTE=0;BYHOUR=2;BYDAY=1SU;BYMONTH=11
END:STANDARD
BEGIN:DAYLIGHT
DTSTART:19500312T020000
TZOFFSETFROM:-0500
TZOFFSETTO:-0400
RRULE:FREQ=YEARLY;BYMINUTE=0;BYHOUR=2;BYDAY=2SU;BYMONTH=3
END:DAYLIGHT
```

The above example shows a single event with ORGANIZER and ATTENDEE properties.
A.2.4 Recurring Event in iCalendar Format

The following is an example of a recurring event in iCalendar format. The DTSTART and DTEND properties are for the first instance of the recurring event. The RRULE property specifies the recurrence rule. In this case the event recurs weekly on Saturday for four weeks:

BEGIN:VCALENDAR
X-LOTUS-CHARSET:UTF-8
VERSION:2.0
PRODID://Lotus Development Corporation//NONSGML Notes 8.5.3//EN
BEGIN:VTIMEZONE
TZID:Eastern
BEGIN:STANDARD
DTSTART:19501105T020000
TZOFFSETFROM:-0400
TZOFFSETTO:-0500
RRULE:FREQ=YEARLY;BYMINUTE=0;BYHOUR=2;BYDAY=1SU;BYMONTH=11
END:STANDARD
BEGIN:DAYLIGHT
DTSTART:19500312T020000
TZOFFSETFROM:-0500
TZOFFSETTO:-0400
RRULE:FREQ=YEARLY;BYMINUTE=0;BYHOUR=2;BYDAY=2SU;BYMONTH=3
END:DAYLIGHT
END:VTIMEZONE
BEGIN:VEVENT
DTSTART;TZID="Eastern":20110101T090000
DTEND;TZID="Eastern":20110101T100000
TRANSP:OPAQUE
DTSTAMP:20110509T134458Z
SEQUENCE:0
ATTENDEE;ROLE=CHAIR;PARTSTAT=ACCEPTED;CN="Duke Lawson/Peaks" ;RSVP=FALSE:mailto:DukeLawson@xyz.com
ATTENDEE;ROLE=REQ-PARTICIPANT;PARTSTAT=NEEDS-ACTION ;CN="Dean Melnyk/Peaks";RSVP=TRUE:mailto:DeanMelnyk@xyz.com
CLASS:PUBLIC
DESCRIPTION:Review project plan
SUMMARY:Meeting example
LOCATION:Your office
ORGANIZER;CN="Duke Lawson/Peaks":mailto:DukeLawson@xyz.com
UID:906CEF2CB4DCCB2C85257888006A9059-Lotus_Notes_Generated
X-LOTUS-BROADCAST:FALSE
X-LOTUS-NOTESVERSION:2
X-LOTUS-APPTTYPE:3
END:VEVENT
END:VCALENDAR

A.2.5 Recurring Event with an Exception in iCalendar Format

Sometimes a recurring event has exceptions – event instances that have been modified or removed. In this case the VCALENDAR component will contain more than one VEVENT:
In the above example, the first \textit{VEVENT} describes the overall event including its recurrence rule. The second \textit{VEVENT} describes an exception that occurs January 14. The DTSTART, DTEND and DESCRIPTION properties are different for the exception.

\textbf{A.2.6 An Instance of a Recurring Event in iCalendar Format}

As described in sections A.2.4 and A.2.5, the service often represents a recurring event as a whole (all event instances). It can also represent a single instance of a recurring event. The example below shows an instance of the same event from section A.2.5. The instance occurs on January 21. It has the same UID property, but it also has a \texttt{RECURRENCE-ID} property. The \texttt{RECURRENCE-ID} property uniquely identifies the instance:
TZOFFSETFROM:-0400
TZOFFSETTO:-0500
RRULE:FREQ=YEARLY;BYMINUTE=0;BYHOUR=2;BYDAY=1SU;BYMONTH=11
END:STANDARD
BEGIN:DAYLIGHT
DTSTART:19500312T020000
TZOFFSETFROM:-0500
TZOFFSETTO:-0400
RRULE:FREQ=YEARLY;BYMINUTE=0;BYHOUR=2;BYDAY=2SU;BYMONTH=3
END:DAYLIGHT
END:VTIMEZONE
BEGIN:VEVENT
DTSTART:20110121T140000Z
DTEND:20110121T150000Z
TRANSP:OPAQUE
RECURRENCE-ID:20110121T140000Z
DTSTAMP:20110509T143343Z
CLASS:PUBLIC
SUMMARY:Another recurring appointment
LOCATION:Situation room
UID:6365E29A99F8CE658525788B004D2718-Lotus_Notes_Generated
X-LOTUS-NOTESVERSION:2
X-LOTUS-APPTTYPE:0
END:VEVENT
END:VCALENDAR
Appendix B. References

The following references are available only on the IBM intranet:

1. Project Vulcan: Atom Patterns and Guidelines,
   http://w3.ibm.com/connections/wikis/home?
   lang=en#/wiki/W74f1b6abee50_4f23_a230_88cddd42a9a0/page/Atom%20Patterns%20and
   %20Guidelines

2. Project Vulcan: JSON Patterns Master List,
   http://w3.ibm.com/connections/wikis/home?
   lang=en#/wiki/W74f1b6abee50_4f23_a230_88cddd42a9a0/page/JSON%20Patterns%20Master
   %20List

3. Project Vulcan: URL Patterns and Query String Parameters Master List,
   http://w3.ibm.com/connections/wikis/home?
   lang=en#/wiki/W74f1b6abee50_4f23_a230_88cddd42a9a0/page/URL%20Patterns%20and
   %20Query%20String%20Parameters%20Master%20List