Kalendi API Manual

Contents

Introduction

Kalendi is a web-based enterprise-wide calendaring system with many distinguishing features including:

- Building of multiple collaborative group calendars to share
- Keyword searching to quickly locate current or past events
- Uploading of multiple files (photos, documents, etc.) per event
- Sending event reminders via email & SMS
- A fine-grained permission system
- Moderated calendars
- CalDav access

The system has an easy-to-use web interface with access to all of the above features. For a complete description of the web interface, see http://www.kalendi.com. In addition, all of Kalendi's functionality is exposed via an Application Programming Interface (API), enabling programmatic access to Kalendi's functionality. This means that it is possible to write specialized applications that make use of Kalendi's features. This manual contains documentation of the kalendi API. It includes:

- A description of how the API handles security.
- A description of the calendar calls
- A description of the event calls
- A description of the adminstrative calls

Companies

The administrative unit in Kalendi is called a company. When you start using Kalendi, a company is created for you. You are asked to supply a company name and phone number. In addition, the following is created:

- A company calendar
- An account for an administrator including a username and password enabling the adminstrator to log on (this is probably you)

The administrator can use all of the functionality available to any Kalendi user and, in addition, can perform all of the administrative functions available in Kalendi. When using the API to perform administrative functions, calls must be made with an administrator's credentials. Some of Kalendi's administrative functions include:

- Editing the properties of the company
- Creating new users
- Removing users
- Getting information about users
- Changing the privileges and other properties of users
- Creating company categories
- Creating company organizations
- Creating company custom properties

A complete list of all administrative API calls can be found later in this document.

Perhaps the most important administrative function initially is the create user function. The administrator can create new users, give them different privileges, assign them to organizations, etc. When a user is first created, she is subscribed to the company calendar. In addition, the user can create her own calendars and (depending on the user's privileges and organization memberships) may be able to subscribe to other calendars in the company. Hence, the users in your company can share event information by enabling each other to subscribe to the various calendars in your company. But which users can see which events can be carefully controlled.

Calendars

Calendars can be of types company, organization (also called group), personal or public. The purpose of these different types is to provide fine-grained yet convenient control over which users can perform which operations on calendars. There are the following types of permissions on calendars:

- subscribe - allowing a user to see the events on a calendar (unless the visibility of the event is hidden or busy)
- append - allowing a user to add new events to a calendar
- modify - allowing the user to modify the events on a calendar
- delete - allowing the user to delete the events on a calendar
- meta - allowing the user to change the properties of the calendar as well as do to all of the above functions

Permissions can be either individual or role permissions. As an example of an individual permission, the creator of a personal calendar can provide any or all permissions individually to any other user in the same company. In addition, role permissions are granted based on the roles that users play in the company. For example, the subscribe permission can be given on a calendar to all employees of the company or the append permission can be given to all (and only) managers of the company.

Public calendars can be subscribed to by any individual in any company on the kalendi server but, otherwise, have the same permission behavior as personal calendars.

Initially a company has two roles: administrator and employee, but a company administrator can add new roles to a company.

Organization calendars are associated with organizations created by a company administrator. When an organization is created is has two roles: leader and member. In addition, an administrator can add new roles to an organization. When a calendar has type organization, permissions can be granted as described above and, in addition, can be granted based on the organization's roles. For example, members of the organization "marketing" can have subscribe permission to an organization calendar associated with that organization.

Calendars can have properties, called metadata. Metadata are key, value pairs that can be associated with calendars and calendars can be retrieved by searching for strings in their metadata values. This allows calendars to be manipulated in ways similar to events, for example, they can be grouped together by having the same or similar metadata.

Calendars can also be moderated. In this case, a user in the company is identified as the moderator and when events are created on the calendar, rather than appearing immediately, the moderator controls whether or not the events are accepted. If, after review, the moderator rejects the event, an email can be sent to the event creator describing the reason for rejection. Moderated calendars allow an organization even more control of what events appear on a calendar.

Events

Calendars are, of course, collections of events. Kalendi events can be thought of as a collection of properties including start time, end time, caption, description, location, etc. Events can be one-time or recurring and they can have a start time or simply be all day events. Recurring events can recur daily, weekly, monthly, or yearly and can have complicated recurrence rules such as occurring on the third from the last day of every month.

Kalendi provides search functions to allow events to be found within a date (or time) range and that have certain properties, for example, events in particular categories. Events in Kalendi can also be assigned to categories. An administrator can add any number of categories to their company and events (when created or modified) can be placed in zero of more of these categories. Categories can be arranged in a two-level hierarchy of organizational categories and (regular) categories but events can only be assigned regular categories. The call `getCategories` returns all of the categories arranged in the hierarchy, so that an application can have access to the hierarchy to, for example, display the categories hierarchically in a graphical user interface to a search function.

As mentioned, events have a set of fixed properties including: a caption, a description, a start date, etc. In addition, Kalendi enables a company to define custom properties. When events are displayed (or returned to a query via the API), the values of custom properties are included along with the values of the fixed properties. Custom properties can have the following types:

- integer
- currency
- string
- list
- boolean
- URL
- phone
- mapit

Kalendi checks the values of custom properties based on their type when they are added to events. For example, the values of custom properties of type integer are checked to ensure that they are integers. In addition, some of the types are used by the Kalendi web interface to control their display or selection. For example, the web interface displays a custom property of type list as a drop down when the value is selected. The web interface displays a custom property that is of type mapit so that if the value is clicked, a google map of the location of the event is displayed.

Security Model

To date, the Kalendi API has been used to create desktop applications and components added to web pages such as the widgets available at http://www.kalendi.com/products/widgets/index.html. These widgets are code placed on web pages to access and display events from Kalendi. As such there is a security risk to accessing Kalendi using login/password information. For this reason, the API includes features to enable API access without exposing a user's credentials. Hence, accessing Kalendi using a username and password should only be done from applications that make secure connections to the Kalendi server.

For all other access, either a token or a referrer should be used. To use a token, one first needs to make a getToken API call to get Kalendi to issue a token. An expiration time and date can be placed on the token. When this login method is used, the username used to issue the token should appear in calls using the token, but no password should appear and the token should appear in the cookies with cookie name 'kalendiAPI'.

A less secure technique is also available but should only be used with accounts that have only subscribe access privileges to your calendars. In this case, one first needs to make a setReferrer API call to set a host or referer. Once one of these is set, logins without a password are accepted if the host or referer of the request matches the one set with setReferrer.

Calendar Calls

These API calls enable the manipulation of calendars.

Creating a calendar

This call creates a new calendar. The owner of the new calendar will be the user whose userName and password were supplied unless this user is an administrator and the optional argument `ownerName` is provided. In this case, the new calendar will belong to the user whose user name is `ownerName`.

Form: `createCalendar.api`

Required Arguments:

- `userName` - a valid user name
- `password` - a password
- `name` - the name of the new calendar
- `type` - the calendar type which can be either Group (or equivalently Organization) or Personal
- `zoneID` - the name of the time zone that calendar should display events in

Optional arguments:

- `ownerName` - the userName of another user in this company
- `organizationName` - the name of an organization (this argument is required if the type argument is included and has a value of Group or Organization)
- `isModerated` - "yes" or "no" (default is "no")
- `subscribe` - whether or not to subscript the owner, "yes" or "no" (default is "no")
- `publishWeb` - "yes" or "no" (default is "no")
- `publishICS` - "yes" or "no" (default is "no")
- `iCalFile` - an ics file to be uploaded and have it events placed on the new calendar
- `description` - text describing the calendar
- `type_scope_role` - This is the way to specify a role permission, for example, subscribe_company_employee=yes
- `type_individual` - This is the way to specify an individual permission, for example, meta_individual=jvb@happyjacksoftware.com where this is a user name.
- `noConflicts` - if specified and has value "yes", then kalendi will not allow events to overlap on the calendar.

Discussion:

If iCalFile is used, the API call must be a post and the value of this parameter must conform to the multi-part/form-data format. The owner of the calendar is automatically given all permissions to it. All other permission must be specific explicitly.

Examples:

- Create a calendar for another user:

```
http://localhost/kalendi/createCalendar.api?
    userName=noah10@happyjacksoftware.com&
    password=noahd&
    name=fooBar1&
    ownerName=kenk2@happyjacksoftware.com&
    subscribe=yes&
    zoneID=America/Denver
```

- Create a calendar with an individual permission:

```
http://localhost/kalendi/createCalendar.api?
    userName=noah10@happyjacksoftware.com&
    password=noahd&
```

```
    name=fooBar5&
    ownerName=kenk2@happyjacksoftware.com&
    subscribe=yes&
    zoneID=America/Denver&
    meta_individual=kenk5@happyjacksoftware.com
```

- Create a calendar with company employees having subscribe permission:

```
http://localhost/kalendi/createCalendar.api?
    userName=noah10@happyjacksoftware.com&
    password=noahd&
    name=fooBar6&
    ownerName=kenk2@happyjacksoftware.com&
    subscribe=yes&
    zoneID=America/Denver&
    subscribe_company_employee=yes
```

- Create new organization calendar with organization role permissions granted:

```
http://localhost/kalendi/createCalendar.api?
    userName=noah10@happyjacksoftware.com&
    password=noahd&
    name=fooBar8&
    subscribe=yes&
    zoneID=America/Denver&
    subscribe_org_member=yes&
    type=Organization&
    organizationName=squirmers
```

- Create new moderated organization calendar:

```
http://localhost/kalendi/createCalendar.api?
    userName=noah10@happyjacksoftware.com&
    password=noahd&
    name=fooBar9&
    subscribe=yes&
    zoneID=America/Denver&
    subscribe_org_member=yes&
    append_org_member=yes&
    isModerated=yes&
    type=Organization&
    organizationName=squirmers
```

Errors:

- calendar name exists
- no such group
- no such role
- incorrect permission specification
- proposed owner does not exist
- missing timezone

Editing a Calendar

This call enables the editing of an existing calendar.

Form: `editCalendar.api`

Required Arguments:

- `userName` - a valid user name
- `password` - a password

Optional arguments:

- `calendarID` - the ID of the calendar to edit
- `type` - the calendar type which can be either Group (or equivalently Organization) or Personal

- `organizationName` - the name of an organization (this argument is required if the type argument is included and has a value of Group or Organization)
- `isModerated` - "yes" or "no" (default is "no")
- `publishWeb` - "yes" or "no" (default is "no")
- `publishICS` - "yes" or "no" (default is "no")
- `zoneID` - the name of the time zone that calendar should display events in
- `iCalFile` - an ics file to be uploaded and have it events placed on the new calendar
- `description` - text describing the calendar
- `type_scope_role` - This is the way to specify a role permission, for example, subscribe_company_employee=yes or remove_subscribe_company_employee=yes
- `type_individual` - This is the way to specify an individual permission, for example, meta_individual=jvb@happyjacksoftware.com or remove_meta_individual=jvb@happyjacksoftware.com where this is a user name.

Discussion:

Exactly one of the calendarSpec of calendarID parameter must be given. Permissions are removed by specifying the name of the permission preceded by "remove_", for example, remove_company_employee. If iCalFile parameter is given, the API call must be a post and the value of this parameter must conform to the multi-part/form-data format. The owner of the calendar is automatically given all permissions to it. All other permission must be specified explicitly.

Examples:

- Make a calendar moderated, group type, and change the organization name

```
localhost:8080/kalendi/editCalendar.api?
userName=jvb&
password=jvb&
calendarID=4725&
isModerated=yes&
type=Group&
organizationName=Some Organization
```

- Make a calendar web published and ICS published

```
localhost:8080/kalendi/editCalendar.api?
userName=jvb&
password=jvb&
calendarID=4725&
publishWeb=yes&
publishICS=yes
```

- Change a calendar's time zone and description

```
localhost:8080/kalendi/editCalendar.api?
userName=jvb&
password=jvb&
calendarID=4725&
zoneID=America/Denver&
publishICS=Our company calendar for vacation days
```

- Add the subscribe permission for all Employee's in the company

```
localhost:8080/kalendi/editCalendar.api?
userName=jvb&
password=jvb&
calendarID=4725&
subscribe_company_employee=yes
```

- Give meta privileges to the manager of the company

```
localhost:8080/kalendi/editCalendar.api?
userName=jvb&
password=jvb&
calendarID=4725&
meta_individual=manager@companydomain.com
```

Deleting a Calendar

This command allows you to delete a Calendar identified by calendarID.

Form: `deleteCalendar.api`

Required Arguments:
- `userName` - a valid user name
- `password` - the password
- `calendarID` - the ID of the calendar to delete

Discussion:

You must have "meta" privileges on the calendar in order to delete it.

Examples:
- Delete a Calendar

```
http://localhost:8080/kalendi/deleteCalendar.api?
    userName=jvb&
    password=jvb&
    calendarID=4231
```

Getting info for a Calendar

This command allows you to get information on a Calendar, all calendars a user has access to, or the calendars a user have access to that have metadata fields containing a query string.

Form: `getCalendar.api`

Required Arguments:
- `userName` - a valid user name
- `password` - the password

Optional Arguments:
- `calendarSpec` - the calendar specification
- `calendarID` - the calendar ID.
- `fields` - a comma-separated list of metadata field names to be searched
- `searchString` - the string to search for

Discussion:

You can call this method using a Calendar Spec or ID, although it is preferable to use the CalendarID as that is an unchanging value. If the calendarSpec and calendarID are both omitted, this will return all the calendars that the user has permissions for.

Examples:
- Getting info on a Calendar

```
http://localhost:8080/kalendi/getCalendar.api?
    userName=jvb&
```

```
   password=jvb&
 calendarID=22
```

- Example Output:

```
         <calendars>
          <calendar>
           <calendarID>5897</calendarID>
           <calendarName>Test All Params</calendarName>
           <calendarType>Personal</calendarType>
           <owner>jvb</owner>
           <description>Just a test</description>
           <zoneID>America/Denver</zoneID>
           <moderator>jvb</moderator>
           <publishICS>y</publishICS>
           <publishWeb>y</publishWeb>
           <indRoles>
            <person username="jvb">
             <value>append</value>
             <value>subscribe</value>
             <value>modify</value>
             <value>delete</value>
             <value>meta</value>
            </person>
            <person username="BOYER">
             <value>append</value>
            </person>
           </indRoles>
           <grRoles>
            <role name="append">
             <value>Manager</value>
            </role>
           </grRoles>
          </calendar>
         </calendars>
```

- Getting info on all Calendars

```
http://localhost:8080/kalendi/getCalendar.api?
         userName=jvb&
         password=jvb
```

- Getting info on all calendars with metadata

```
http://localhost:8080/kalendi/getCalendar.api?
      userName=jvb&
      password=jvb&
               fields=public_url&
               searchString=happyjacksoftware.com
```

This call finds all calendars to which the user has access that have a metadata field called `public_url` that contains the string `happyjacksoftware.com`.

Get info for a public Calendar

Public calendars are can be subscribed to from any company in Kalendi. This command allows you to get information about a public calendar, all public calendars, or to search for public calendars with certain string in metadata fields..

Form: `getPublicCalendar.api`

Required Arguments:

- `userName` - a valid user name
- `password` - the password

Optional Arguments:

- `calendarSpec` - the calendar specification
- `calendarID` - the calendar ID.
- `fields` - a comma-separated list of metadata field names to be searched
- `searchString` - the string to search for

Discussion:

You can call this method using a Calendar Spec or ID, although it is preferable to use the CalendarID as that is an unchanging value. If the calendarSpec and calendarID are both omitted, this will return all the public calendars.

Examples:

- Getting info on a Calendar

```
http://localhost:8080/kalendi/getPublicCalendar.api?
                userName=jvb&
                password=jvb&
                calendarID=4329
```

- Example Output:

```
<calendars>
  <calendar>
   <calendarID>4329</calendarID>
   <calendarName>U.S. Holiday</calendarName>
   <calendarType>Public</calendarType>
   <owner>mona@happyjacksoftware.com</owner>
   <description/>
   <zoneID>US/Mountain</zoneID>
   <publishICS>n</publishICS>
   <publishWeb>y</publishWeb>
  </calendar>
</calendars>
```

- Getting info on all Calendars

```
http://localhost:8080/kalendi/getCalendar.api?
                    userName=jvb&
                    password=jvb
```

- Searching for public calendars by metadata fields

```
http://localhost:8080/kalendi/getCalendar.api?
                userName=jvb&
                password=jvb&
                fields=public_url&
                searchString=happyjacksoftware.com
```

  This call finds all public calendars that have a metadata field called `public_url` that contains the string `happyjacksoftware.com`.

Subscribing to a Calendar

The purpose of this call is to allow a user to subscribe to a calendar to which he/she has subscribe permission.

Form: `subscribe.api`

Required Arguments:

- userName - a valid userName
- password - the kalendi user's password
- calendarID - the ID of a calendar

Examples:

- ```
  http://localhost/kalendi/subscribe.api?
                          userName=jvb@happyjacksoftware.com&
                          password=noahd&
                          calendarID=<calendarID>
  ```

Errors:

- Login failed
- The calendar does not exist or you do not have privilege to subscribe

Unsubscribing from a Calendar

The purpose of this call is to allow a user to unsubscribe from a calendar. If the user has not subscribed to the calendar, this call does nothing.

Form: `unsubscribe.api`

Required Arguments:

- userName - a valid userName
- password - the kalendi user's password
- calendarID - the ID of a calendar

Examples:

- ```
  http://localhost/kalendi/unsubscribe.api?
                          userName=jvb@happyjacksoftware.com&
                          password=noahd&
                          calendarID=<calendarID>
  ```

Errors:

- Login failed

Getting a Calendar's ICS

This command allows you to get the contents of the specified calendar in ICS format.

Form: `getICS.api`

Required Arguments:

- `userName` - a valid user name
- `password` - the password

Optional Arguments:

- `calendarSpec` - the name of the Calendar (full API name)
- `calendarID` - a calendar ID

Discussion:

You must provide a valid calendarSpec or calendar ID, and you must have subscribe permissions to run this command. The format of the return value is as follows:

Return:

```
<ICS>
   <![CDATA[ contents of ICS file ]]>
</ICS>
```

Examples:

- Get ICS from Calendar

```
http://localhost:8080/kalendi/getICS.api?
           calendarSpec=UW Computer Science/Jeffrey Van Baalen/jvb's cal&
           userName=jvb&
           password=jvb
```

Changing a Moderator

This command allows you to change a calendar's moderator. As well as the usual "userName" and "password" arguments, you pass in "calendarSpec" and "changeUser", where calendarSpec is the long calendar name for the calendar you wish to change, and changeUser is the login name of the user you wish to set as moderator.

Form: `changeModerator.api`

Required Arguments:

- `userName` - a valid user name
- `password` - the password
- `changeUser` - the login username of the user you wish to be the new moderator

Optional Arguments:

- `calendarSpec` - the full calendar name of the calendar on which you wish to change the moderator
- `calendarID` - the calendar ID of the calendar on which you wish to change the moderator.

Discussion:

You must specify a `calendarSpec` or a `calendarID`. You must have sufficient privileges (meta) on the specified calendar to successfully execute this call. If you do not have meta privileges an error will be thrown. The calendar must also be moderated in order for this call to work. Finally if you specify an invalid username or an invalid calendar name the call will fail.

Examples:

- Change the moderator

```
http://localhost:8080/kalendi/changeModerator.api?
 calendarSpec=UW Computer Science/Jeffrey Van%20Baalen/jvb's cal&
 changeUser=BOYER@happyjacksoftware.com&
 userName=jvb&
 password=jvb
```

Adding Calendar Metadata

AddCalendarMetaData enables the user to add metadata fields to calendars to which he or she has meta privileges. The fields can have any names desired and the values are arbitrary strings.

Form: `addCalendarMetaData.api`

Required Arguments:

- `userName` - a valid user name
- `password` - the password
- `key` - the key (or field) name of the metadata
- `value` - the value for the key which can be an arbitrary string of up to 200 characters

Optional Arguments:

- `calendarSpec` - a calendar specification
- `calendarID` - a calendar ID

Discussion

The specified key-value pair is added to the calendar's metadata. If the calendar already has metadata with the given key, it is replaced. You can call this method using a Calendar Spec or ID, although it is preferable to use the CalendarID as that is an unchanging value.

Example:

```
http://localhost/kalendi/addCalendarMetaData.api?
                username=jvb&password=jvb&
                calendarID=5465&
                key=blog_url&
                value=http://www.happyjacksoftware.com/blogspace
```

Deleting Calendar Metadata

deleteCalendarMetaData enables the user to delete metadata fields from calendars to which he or she has meta privileges.

Form: `deleteCalendarMetaData.api`

Required Arguments:

- `userName` - a valid user name
- `password` - the password

Optional Arguments:

- `calendarSpec` - a calendar specification
- `calendarID` - a calendar ID
- `key` - the key (or field) name of the metadata

Discussion

The specified key is deleted to the calendar's metadata. If a key is not specified, all metadata for the calendar is deleted. You can call this method using a Calendar Spec or ID, although it is preferable to use the CalendarID as that is an unchanging value.

Example:

```
http://localhost/kalendi/deleteCalendarMetaData.api?
                username=jvb&password=jvb&
                calendarID=5465&
                key=blog_url
```

Getting Calendar Metadata

GetCalendarMetaData enables the user to retreive a calendar's metadata either for a specific field or all metadata.

Form: `getCalendarMetaData.api`

Required Arguments:

- `userName` - a valid user name
- `password` - the password

Optional Arguments:

- `calendarSpec` - a calendar specification
- `calendarID` - a calendar ID
- `key` - the key (or field) name of the metadata

Discussion

If a key is specified, return the data for that key, otherwise return all pf the key-value pairs for the calendar. You can call this method using a Calendar Spec or ID, although it is preferable to use the CalendarID as that is an unchanging value.

Example:

```
http://localhost:8080/kalendi/getCalendarMetaData.api?
                calendarID=5971&
                userName=jvb&
                password=jvb
```

Returns:

```
<metadata>
  <calendar id='5971'>
    <data key='blog_url'>http://foo.bar.com</data>
    <data key='another_key'>Here is some value</data>
  </calendar>
</metadata>
```

Event Calls

The API calls enable the manipulation of events.

Get Events

GetEvents gets events from the calendars to which a user has subscribe permission.

Form: `getEvents.api`

Optional Arguments:

- `userName` - a valid user name
- `password` - the password
- `calendars` - A comma-separated list of calendar specifications to search. If this parameter is not present all calendars to which the user can subscribe are searched.
- `calendarSpec` - This is an alternative to the parameter immediately above. It can be repeated as many times as desired in the argument string. This enables calendars whose names contain commas to be specified. If you include this parameter one or more times, the value of any calendars parameter will be ignored.
- `calendarID` - the calendarID of the calendar to be searched.
- `fields:` - A list of fields to search on. The default fields are "caption" and "description". You can also specify "location" to search events location fields. If there are custom properties you want to search you simply include the custom property's name. For example, if there is an "Instructor" custom property you can specify `fields=Instructor` to have the search string cover that field as well.
- `string:` - A search string. All events containing any of the words in the search string either in their caption or their description (or `fields` if specified) will be returned.
- `andString:` - A search string. All events containing all of the words in the search string either in their caption or their description (or `fields` if specified) will be returned.
- `exactString:` - A search string. All events containing exactly the given string either in their caption or their description (or `fields` if specified) will be returned.
- `startDT` - start date for the search in the form DD-MM-YYYY. Events after this date that meet the other criteria are returned. If this parameter is omitted, today is used.
- `endDT` - end date for the event in the form DD-MM-YYYY. Events before this date that meet the other criteria are included. If this parameter is omitted, the first thirty events meeting the rest of the criteria are returned.

Discussion

If none of the arguments: calendars, calendarSpec, or calendarID is given, all of the calendars to which the user have subscribe permission are searched. If none of the parameters: `string`, `andString`, `exactString` is given, all events that meet the other criteria will be returned. GetEvents returns an XML document containing either an error or containing the events found. The structure of the XML return is as follows:

```
<VEvents>
    <VEvent>
      <eventID>an integer</eventID>
      <caption>The caption</caption>
      [<description>Text of description</description>]
      <visibility>public|hidden|busy|open</visibility>
      <calendarName>The name of the calendar on which the event appears</
calendarName>
      <date>YYYY-MM-DD</date>
      [<allDay/>| <stTime> a time </stTime> <endTime> a time </endTime>]
      [<location> Text </location>]
      <recurring>yes|no</recurring>
      [<refurl>A url that points to the day of the event</refurl>]
      <customProp1>prop value1</customProp1>
```

```
      ...
    </VEvent>
    …
</VEvents>
```

Add an event

Add a new event to a calendar in your company. Events can be all day in which case, no start time or end time is given.

Form: `addEvent.api`

Required Arguments:
- `userName` - a valid user name
- `password` - the password
- `calendar` - the specification for the calendar on which to place the event
- `caption` - the text of the event caption
- `startDT` - start date for the event in the form DD-MM-YYYY
- `endDT` - end date for the event in the form DD-MM-YYYY

Optional Arguments:
- `allDay` - indicates whether or not the event is an all day event ("yes" or "no"), defaults to "no"
- `startHour` - an integer between 0-23, required if allDay=no
- `endHour` - an integer between 0-23, required if allDay=no
- `startMinute` - an integer between 0-59, required if allDay=no
- `endMinute` - an integer between 0-59, required if allDay=no
- `startPM` - "yes" or "no"
- `endPM`
- `durHour` - an integer specifying the duration of the event in hours
- `durMinutes` - an integer specifying the duration of the event in minutes
- `startTZ` - the name of a valid java timezone
- `endTZ` - the name of a valid java timezone
- `description` - text of the description of the event
- `customProp` - a comma-separated list of custom property name, value pairs with the pairs separated by ":"
- `categoryNames` - a comma-separated list of category names
- `reminder` - an integer specifying the number of minutes before an event occurs that a reminder should be sent
- `notifications` - a comma-separated list of email or cell phone address to which the reminder for this event should be sent. Note that these must be verified addresses (see "Adding a Reminder Address"). If no notificvations are given. The reminder is sent to the users default email address (in the user's profile)
- `visibility` - the visibility of the event with possible values: “public,” “hidden,” “open,” “busy”. If omitted, defaults to “public”. If the event is a portal event, further possibilities are: "portal", "highlight"
- `imageFile` - the value is the contents of an image file in the standard Form File type, sent via multipart/form-data request.
- `freq` - If this is a recurring event, this gives the frequency of the event. Possible values: “none” (same as omitting freq altogether), “daily”, “weekly”, “monthly”, “yearly”
- `interval` - an integer that gives the interval of occurrence, for example, if freq=daily and interval=1, the event occurs every day
- `maxEvents` - an integer specifying the number of occurrences of the event
- `recurEndDT` - the date of the last occurrence of the event in MM-DD-YYYY format
- `includeWeekends` - this parameter only has an effect when freq=daily. If it is included and its value is “1”, then days on the weekend are included. Otherwise, only week days are included
- `daysOfWeek` - is a comma-separated list of integers where 1=Sunday, 2=Monday, 4=Tuesday, 8=Wednesday, 16=Thursday, 32=Friday, 64=Saturday. Hence, one can create an event that recurs weekly on Monday, Wednesday, and Friday by including * `daysOfWeek`=2,8,32
- `monthDay` - an integer between 1-31 specifying the day of the month on which the event should occur
- `whichWay` - the possible values are “forward” and “backward” allowing you to say things like `monthDay`=3&`whichWay`=backward which specifies that the event occurs on the third day from the end of the

month or `monthDay`=2&`daysOfWeek`=2&`whichWay`=backward which specifies that the event occurs on the second to last Monday of each month. This parameter defaults to “forward”.

- `yearDT` - the day of the year on which the event occurs in MM-DD-YYYY format.

Discussion:

If `allDay`=no, `startHour` and `startMinute` are required. If `startPM` is specified, then start time is in AM/PM format and `startHour` must be an integer between 1-12, otherwise, the start time is in military format and `startHour` must be a value between 0-23. If `startTZ` is specified, the start time is interpreted as being in the given timezone, otherwise, it is interpreted as being in the calendar's default timezone. the name of JAVA name of the timezone of the start time. Either a duration or an end time can be specified. If both are specified, duration takes precedence. It is an error if neither is specified. Duration can be given in minutes, hours or a combination (in which case the duration is the total number of hours and minutes) of both with the two parameters: `durHour` and `durMinutes`. If a duration is not given, then the event end time must be given using `endHour`, `endMinute`, `endPM`, and `endTZ` in same format as their “start” counterparts.

The `customProp` argument can either have a value that is a comma-sepearted list of name, value pairs or the argument can be given multiple times, one for each name, value pair.

If the `freq` argument is given and its value is not “none”, the event is recurring. All recurring events must include the `interval` argument. All recurring events must also include either the `maxEvents` argument or the `recurEndDT` argument.

If an event recurs daily then it can also include `includeWeekends` parameter. If an event recurs weekly, then it can also include the `daysOfWeek` parameter. If you do not provide this parameter, then the event will recur weekly on the day of the week of the start day. If an event recurs monthly, then it can include the `monthDay`, `daysOfWeek`, and `whichWay` parameters. In this case, the `daysOfWeek` is a single integer (as opposed to a list even though the argument is the same name as above) with 1=Sunday, 2=Monday, 4=Tuesday, 8=Wednesday, 16=Thursday, 32=Friday, 64=Saturday. This parameter is used in conjunction with the previous parameter to say that, e.g., an event occurs on the second Tuesday of each month. This argument cannot be used without monthDay. If an event recurs yearly, then it can include the `yearDT` parameter. If this parameter is omitted, `startDT` is used.

If a portal event has a highlight image associated with it, that image can be submitted using the `imageFile` parameter. If the visibility of an event is anything other than "portal" or "highlight", then this argument is ignored.

Examples:

- ```
  http://localhost/kalendi/addEvent.api?
      userName=nn@happyjacksoftware.com&
      password=nn&
      calendar=LCC/Jeffrey Van Baalen/Harold Green&
      caption=Harold's first event&
      description=First event description&
      allDay=no&
      startDT=05-16-2009&
      startHour=14&
      startMinute=00&
      endDT=05-16-2009&
      endHour=15&
      endMinute=00&
      customProp=type:Doctor
   Appt.,uniqueID:DC8DD3DF547C0D6264D5CEDC6DE8CFD5846B53BE
  ```
- ```
  http://localhost/kalendi/addEvent.api?
      allDay=no&
      caption=Goshen County Brawl&
      endDT=08-02-2009&
      startDT=07-27-2009&
      calendar=Houston/Tourism Members Members/All Events&
      endHour=9&
      password=w3bevent&
      userName=wyotour_webevent@happyjacksoftware.com&
      startTZ=America/Denver&
      submittersEmail=goshencountychamber@yahoo.com&
      location=Goshen County Fairgrounds&
  ```

```
    visibility=portal&
    endMinute=00&
    endTZ=America/Denver&
    freq=none&startHour=7&
    endPM=yes&
    startMinute=00&
    categoryNames=Rodeo,Animal Shows ,Animal Competitions,Events/
Celebrations,County and State Fairs&
    description=Annual Goshen County Fair. Events include: ...&
    startPM=no&
    customProp=city:Torrington&
    customProp=Contact:Stephanie Lofink&
    customProp=Organization:Goshen County Fair Association&
    customProp=Contact email:goshencountychamber@yahoo.com&
    customProp=MapIt:7078 Fairgrounds Rd., Highway 26, Torrington, WY
 82240&
    customProp=Web Site:http://www.goshencountychamber.com&
    customProp=Contact Phone Number:307-532-2525&
    customProp=Pricing:Gate admission free, events vary
```

- A daily recurring event every other day 5 times (no weekends):

```
http://localhost/kalendi/addEvent.api?
    userName=jvb&
    password=jvb&
    calendar=UWCS/Jeffrey Van Baalen/jvb's cal&
    caption=Harold's first event&
    description=First event description&
    allDay=no&
    startDT=06-25-2009&
    startHour=14&
    startMinute=00&
    endDT=06-25-2009&
    endHour=15&
    endMinute=00&
    freq=daily&
    interval=2&
    maxEvents=5
```

- A daily recurring event every other day until July 15 (no weekends):

```
http://localhost/kalendi/addEvent.api?
    userName=jvb&
    password=jvb&
    calendar=UWCS/Jeffrey Van Baalen/jvb's cal&
    caption=Harold's second event&
    description=Second event description&
    allDay=no&
    startDT=06-26-2009&
    startHour=14&
    startMinute=00&
    endDT=06-26-2009&
    endHour=15&
    endMinute=00&
    freq=daily&
    interval=2&
    recurEndDT=07-15-2009
```

- A weekly event:

```
http://localhost/kalendi/addEvent.api?
    userName=jvb&
    password=jvb&
    calendar=UWCS/Jeffrey Van Baalen/jvb's cal&
    caption=Harold's third event&
    description=Third event description&
```

```
    allDay=no&
    startDT=06-26-2009&
    startHour=13&
    startMinute=00&
    endDT=06-26-2009&
    endHour=15&
    endMinute=00&
    freq=weekly&
    recurEndDT=07-15-2009
```

- A weekly event on Sunday and Tuesday:

```
http://localhost/kalendi/addEvent.api?
    userName=jvb&
    password=jvb&
    calendar=UWCS/Jeffrey Van Baalen/jvb's cal&
    caption=Harold's fourth event&
    description=Fourth event description&
    allDay=no&
    startDT=06-26-2009&
    startHour=13&
    startMinute=00&
    endDT=06-26-2009&
    endHour=15&
    endMinute=00&
    freq=weekly&
    recurEndDT=07-15-2009&
    daysOfWeek=1,4
```

- A monthly event:

```
http://localhost/kalendi/addEvent.api?
    userName=jvb&
    password=jvb&
    calendar=UWCS/Jeffrey Van Baalen/jvb's cal&
    caption=Harold's fifth event&
    description=Fifth event description&
    allDay=no&
    startDT=06-26-2009&
    startHour=13&
    startMinute=00&
    endDT=06-26-2009&
    endHour=15&
    endMinute=00&
    freq=monthly&
    maxEvents=5
```

- A monthly event on the third day of each month:

```
http://localhost/kalendi/addEvent.api?
    userName=jvb&
    password=jvb&
    calendar=UWCS/Jeffrey Van Baalen/jvb's cal&
    caption=Harold's sixth event&
    description=Sixth event description&
    allDay=no&
    startDT=06-26-2009&
    startHour=13&
    startMinute=00&
    endDT=06-26-2009&
    endHour=15&
    endMinute=00&
    freq=monthly&
    maxEvents=5&
    monthDay=3
```

- A monthly event on the third to last day of each month:

```
http://localhost/kalendi/addEvent.api?
    userName=jvb&
    password=jvb&
    calendar=UWCS/Jeffrey Van Baalen/jvb's cal&
    caption=Harold's sixth event&
    description=Sixth event description&
    allDay=no&
    startDT=06-26-2009&
    startHour=13&
    startMinute=00&
    endDT=06-26-2009&
    endHour=15&
    endMinute=00&
    freq=monthly&
    maxEvents=5&
    monthDay=3&
    whichWay=backward
```

- Add a monthly event on the 3rd Tuesday of every month:

```
    http://www.kalendi.com/kalendi/addEvent.api?
    userName=flexkalendi@gmail.com&
    password=happyjackdev&
    caption=3rd Tuesday of the Month&
    calendarID=6184&
    startDT=06-07-2012&
    endDT=06-07-2012&
    description=This event occurs on the 3rd Tuesday of every month for
 three months&
    location=Test Location&
    freq=monthly&
    interval=1&
    maxEvents=3&
    monthDay=3&
    whichWay=forward&
    daysOfWeek=4&
    allDay=yes
```

- A yearly event:

```
http://localhost/kalendi/addEvent.api?
    userName=jvb&
    password=jvb&
    calendar=UWCS/Jeffrey Van Baalen/jvb's cal&
    caption=Harold's seventh event&
    description=Seventh event description&
    allDay=no&
    startDT=06-26-2009&
    startHour=13&
    startMinute=00&
    endDT=06-26-2009&
    endHour=15&
    endMinute=00&
    freq=yearly&
    maxEvents=5
```

Deleting an event

This command allows you to delete existing events, identified by eventID.

Form: `deleteEvent.api`

Required Arguments:

- `userName` - a valid user name
- `password` - the password
- `eventID` - the ID of the event to be deleted

Examples:

- ```
  http://localhost/kalendi/deleteEvent.api?
      userName=jvb&
      password=jvb&
      eventID=4450183
  ```

Deleting an event instance

This command allows you to delete existing event instances, identified by eventID and startDT, i.e., the call will delete the instance of eventID that starts on day startDT.

Form: `deleteInstance.api`

Required Arguments:

- `userName` - a valid user name
- `password` - the password
- `eventID` - the ID of the event instance to be deleted
- `startDT` - the startDT of the event instance to be deleted

Examples:

- ```
  http://localhost/kalendi/deleteInstance.api?
      userName=jvb&
      password=jvb&
      eventID=4450183&
    startDT=05-10-2010
  ```

Adding an Attachment

This command allows you to add an attachment to an event.

Form: `addAttachment.api`

Required Arguments:

- `userName` - a valid user name
- `password` - the password
- `eventID` - the ID of the event to add the attachment to
- `attachFile` - the file you wish to add as an attachment

Discussion:

You must provide a valid `eventID`. If the event is not on any of your companies calendar's then the call will fail. This api call must be called as a POST method. The `attachFile` is uploaded using a standard multi-part Form File

Examples:

- Add an Attachment (in Ruby)

```
          require 'rubygems'
          require 'httpclient'
```

```
          HTTPClient.post 'http://localhost:8080/kalendi/
addAttachment.api', { :attachFile => File.new("pic.jpg"), :eventID =>
 "2667636", :userName => "jvb", :password => "jvb"}
```

Get attachments for an Event

This command allows you to get attachments for an event.

Form: `getAttachments.api`

Required Arguments:

- `userName` - a valid user name
- `password` - the password
- `eventID` - the id of the event to get attachments for

Discussion:

This will return an XML document containing encoded file data for each attachment in an event.

Examples:

- Getting event attachments

```
http://localhost:8080/kalendi/getAttachments.api?
 userName=jvb&
 password=jvb&
 eventID=43615
```

- Example Output:

```
<attachments>
 <eventID id="43615">
      <fileName>Picture 2.png</fileName>
       <contents>
          <![CDATA[
          ....Encoded File Data here...
          ]]>
      </contents>
       </eventID>
</attachments>
```

Get attachment URIs for an Event

This command allows you to get attachment URIs for an event. If an event has attachments stored on the kalendi server instead of returning the contents of the file, a URI reference to the file on the kalendi server is returned.

Form: `getAttachmentURIs.api`

Required Arguments:

- `userName` - a valid user name
- `password` - the password
- `eventID` - the id of the event to get attachments for

Discussion:

This will return an XML document containing encoded file data for each attachment in an event.

Examples:

- Getting event attachments

```
http://localhost:8080/kalendi/getAttachmentURIs.api?
userName=jvb&
password=jvb&
eventID=43615
```

- Example Output:

```
<attachments>
<eventID id="43615">
<url>http://kalendi.com/calibrate_resources/attachments/
event43615/instance499834627/Picture 2.png</url>
</eventID>
</attachments>
```

Accept or Reject a pending moderated event

This command allows you to accept or reject a pending event for a moderated calendar.

Form: `pendingEvent.api`

Required Arguments:

- `userName` - a valid user name (must be the calendar's moderator)
- `password` - the password
- `eventID` - the ID of the event to accept/reject
- `acceptEvent` - a "yes" or "no" value specifying if you want to accept the event or not

Optional Arguments:

- `explanation` - an explanation for why you rejected the event. (does nothing if `acceptEvent` is "yes")

Discussion:

Once you accept an event it will be added to the calendar. An email will be sent to the event submitter telling them of your action, and if you set `acceptEvent` to "no" and specify an `explanation` it will send the explanation in the email.

Examples:

- Accepting an Event

```
http://localhost:8080/kalendi/pendingEvent.api?
 userName=jvb&
 password=jvb&
 eventID=426656&
 acceptEvent=yes
```

- Rejecting an Event

```
 http://localhost:8080/kalendi/pendingEvent.api?
  userName=jvb&
  password=jvb&
  eventID=426656&
  acceptEvent=no&
```

```
explanation=I do not like your description
```

Administrative Calls

Create a new company

This does roughly the same thing as signing up on the kalendi sign up page, i.e., it creates a new company on the kalendi server.

Note that, just like on the sign up web page, there is no security on this call. However, this functionality is conditionally enabled on each kalendi server through the `createCompany` parameter in the web.xml file.

Form: `createCompany.api`

Required Arguments:

- companyName - the name of the new company
- personFirstName - the kalendi administrator's first name
- personLastName - the kalendi administrator's last name
- email - the kalendi administrator's email address (used also as the administrator's user name)
- phoneNumber - the kalendi administrator's phone number
- password - the kalendi administrator's password

Optional Arguments:

- companyRoles - the company roles (administrator and employee are automatic)
- abbrev - a company abbreviation
- logoURL - the url to open if the company (or default) logo is clicked on the calendar page in the standard web interface
- logoFile - an image file to use to brand the standard web interface. The image will be scaled to 103px X 100px.
- categories - a list of the company's categories

Examples:

- ```
  http://localhost/kalendi/createCompany.api?
      companyName=The Company Name&
      personFirstName=Noah&
      personLastName=Smith&
      email=noahs@happyjacksoftware.com&
      password=noahd&
      phoneNumber=3077666177&
      abbrev=TCN
  ```
- ```
  http://localhost/kalendi/createCompany.api?
      companyName=The Company Name2&
      personFirstName=Noah&
      personLastName=Smith&
      email=noahs2@happyjacksoftware.com&
      password=noahd&
      phoneNumber=3077666177&
      logoURL=http://foo.bar.com
  ```
- ```
  http://localhost/kalendi/createCompany.api?
      companyName=Noahs Ark3&
      personFirstName=Noah&
      personLastName=Daniels&
      email=noah3@happyjacksoftware.com&
      password=noahd&
      phoneNumber=3077666177&
      companyRoles=Carrier,walker
  ```
- ```
  http://localhost/kalendi/createCompany.api?
  ```

```
    companyName=Noahs Ark8&
    personFirstName=Noah&
    personLastName=Daniels&
    email=noah8@happyjacksoftware.com&
    password=noahd&
    phoneNumber=3077666177&
    orgs=elites=runner:player,fools=clown:joker
```

- ```
  http://localhost/kalendi/createCompany.api?
      companyName=Noahs Ark10&
      personFirstName=Noah&
      personLastName=Daniels&
      email=noah10@happyjacksoftware.com&
      password=noahd&
      categories=urgent,boring&
      phoneNumber=3077666177
  ```

Discussion:

If the loginLogo parameter is being supplied, the call must be made as a post and the value of this parameter must conform to the multi-part/form-data format. The supplied image can be any of: jpg, gif, or png and is scaled to

Errors:

- Operation not allowed
- Company name exists
- Username exists
- Error uploading logo file - in this case the company is still created, but will not have a logo file.

Editing a Company

This command allows you to edit the abbreviation, logoURL, and logoFile of a Company.

Form: `editCompany.api`

Required Arguments:

- `userName` - a valid user name
- `password` - the password

Optional Arguments:

- `abbrev` - a new company abbreviation
- `logoURL` - a URL where the company logo is located
- `logoFile` - a form file containing the company logo

Discussion:

You do not provide a companyID for this operation, it just uses whichever company the logged in user is a member of. If you provide a `logoFile` it must be the standard Form File upload type.

Examples:

- Edit a Company Abbreviation

  ```
  http://localhost:8080/kalendi/editCompany.api?
   userName=jvb&
   password=jvb&
   abbrev=New Abbreviation
  ```

- Edit a Company LogoURL

  ```
  http://localhost:8080/kalendi/editCompany.api?
   userName=jvb&
   password=jvb&
   logoURL=www.happyjacksoftware.com
  ```
  ```

Get information for the Company

This command allows you to get information for the Company.

Form: `getCompany.api`

Required Arguments:

- `userName` - a valid user name
- `password` - the password

Discussion:

This will return XML describing all the attributes of the company including roles and organizations.

Examples:

- Getting info on the Company

```
http://localhost:8080/kalendi/getCompany.api?
    userName=jvb&
    password=jvb
```

- Example Output:

```
          <companyData>
            <companyName>UW Computer Science</companyName>
            <companyAbbrev>UWCS</companyAbbrev>
            <loginLogo>/kalendi_resources/images/UWCS/uw_cslogo2.gif</
loginLogo>
            <logoURL>http://www.cs.uwyo.edu</logoURL>
            <roles>
                <role>Administrator</role>
                <role>Employee</role>
            </roles>
            <organizations>
               <organization name="BookCat">
                   <role>Member</role>
                   <role>Leader</role>
               </organization>
               <organization name="Create Cal Org">
                   <role>Manager</role>
                   <role>Employee</role>
               </organization>
               <organization name="Test did it Change">
                   <role>Test Add Role</role>
               </organization>
            </organizations>
          </companyData>
```

Add (or replace) a Company's Facebook ID

The purpose of this call is to enable facebook applications (and plugins) to access a company's calendars in Kalendi given the company's facebook ID. The call allows an administrator to add (or replace) a company's facebook ID. It returns (and remembers) a token that, along with the Kalendi userName and secretKey, can be used to access calendars in the company to which the administrator has access. Which calendars and what permissions should be available through this token are also set in this call. After this call is made, the `getFacebookToken` call can be made to retreive the Token anytime the facebook needs it, i.e., the application does not need to remember the token.

Form: `addFacebookID.api`

Required Arguments:

- userName - a valid administrator userName
- password - the kalendi administrator's password
- facebookID - the company's facebook ID
- secretKey - the facebook application's secret key
- perms - a comma-separated list of permissions for one of the user's calendars in the form calendarID:permission_name

Examples:

- ```
  http://localhost/kalendi/addFacebookID.api?
                           userName=jvb@happyjacksoftware.com&
                           password=noahd&
                           facebookID=<facebookID>&
                           secretKey=<secretKey>
                           perms=96:subscribe,105:append
  ```

Discussion:

The calendar IDs are the IDs of user calendar in Kalendi. The user of this call must first call `getCalendar.api` to get those values. The call will not allow more liberal permissions than the user has to be granted or a permissions cannot be granted error is returned. The possible values for permission types are: subscribe, append, delete, modify, and meta.

Errors:

- Operation not allowed
- facebookID cannot be empty
- secretKey cannot be empty
- Explicit permissions must be given
- Permissions cannot be granted

Get the Access Token for a Company's Facebook ID

The purpose of this call is to allow a facebook application to retrieve a company's Kalendi access token.

Form: `getFacebookToken.api`

Required Arguments:

- userName - a valid administrator userName
- password - the kalendi administrator's password
- facebookID - the company's facebook ID
- secretKey - the facebook application's secret key

Examples:

- ```
  http://localhost/kalendi/getFacebookToken.api?
                           userName=jvb@happyjacksoftware.com&
                           password=noahd&
                           facebookID=<facebookID>&
                           secretKey=<secretKey>
  ```

Example return value:

- ```
  <?xml version='1.0' encoding='UTF-8'?>
  <token>660af3e0937711e0bed6001c42000009</token>
  ```

Errors:

- Login failed
- Not found
  ```

Delete a Company's Facebook token

The purpose of this call is to delete a company's facebook token.

Form: `deleteFacebookToken.api`

Required Arguments:

- userName - a valid administrator userName
- password - the kalendi administrator's password
- facebookID - the company's facebook ID
- secretKey - the facebook application's secret key

Examples:

- ```
  http://localhost/kalendi/deleteFacebookToken.api?
                           userName=jvb@happyjacksoftware.com&
                           password=noahd&
                           facebookID=<facebookID>&
                           secretKey=<secretKey>
  ```

Errors:

- Login failed

Getting Information about a User

This command allows you to get a users information.

Form: `getUser.api`

Required Arguments:

- `userName` - a valid user name
- `password` - the password

Optional Arguments:

- `changeUser` - the username of a user

Discussion:

If `changeUser` is provided, that users details will be returned, if no change user is provided, then information for ALL users will be provided.

Examples:

- Get User

  ```
  http://localhost:8080/kalendi/getUser.api?
   userName=jvb&
   password=jvb&
   changeUser=jdoe@gmail.com
  ```
- Example Output:

  ```
  <users>
   <user>
     <firstName>Jane</firstName>
     <lastName>Doe</lastName>
     <email>jdoe@gmail.com</email>
     <username>jdoe</username>
  ```
  ```

```
   <password>test</password>
   <verified>y</verified>
   <coRoles>
  <role>Employee</role>
  <role>Administrator</role>
   </coRoles>
 </user>
</users>
```

Creating a new user

This operations adds a new user to a company so that new staff can be added to kalendi (must be a kalendi administator). When the new account gets created, it will not be activated until the user checks their email and verifies the account by clicking on the link in the email.

Form: `createUser.api?`

Required Arguments:

- userName - the userName of an existing user with admin privileges
- password - the password of the admin user
- personFirstName - the first name of the person being created
- personLastName - the last name of the person being created
- newPassword - the password of the person being created
- email - the new person's email (will be their user name)

Optional Arguments:

- coRoles - A comma separated list of the company roles this person has (all users are automatically given the "employee" role).
- grRoles - A comma separated list of the group roles this person has (all users are automatically given the "member" role for each group included). Each list element is of the form: group-name:role-name1

Examples:

- ```
  http://localhost/kalendi/createUser.api?
      userName=noah10@happyjacksoftware.com&
      password=noahd&
      newPassword=kenk&
      personFirstName=Ken&
      personLastName=Kune&
      email=kenk1@happyjacksoftware.com
  ```
- User with company roles:

  ```
  http://localhost/kalendi/createUser.api?
      userName=noah10@happyjacksoftware.com&
      password=noahd&newPassword=kenk&
      personFirstName=Kenny&
      personLastName=Kune&
      email=kenk2@happyjacksoftware.com&
      coRoles=clown
  ```
- ```
  http://localhost/kalendi/createUser.api?
      userName=noah10@happyjacksoftware.com&
      password=noahd&newPassword=kenk&
      personFirstName=Kenny&
      personLastName=Kuney&
      email=kenk3@happyjacksoftware.com&
      coRoles=clown,administrator
  ```
- User with group roles:

```
http://localhost/kalendi/createUser.api?
    userName=noah10@happyjacksoftware.com&
    password=noahd&
    newPassword=kenk&
    personFirstName=Kenny&
    personLastName=Zipple&
    email=kenk4@happyjacksoftware.com&
    grRoles=squirmers:leader
```

- ```
  http://localhost/kalendi/createUser.api?
      userName=noah10@happyjacksoftware.com&
      password=noahd&
      newPassword=kenk&
      personFirstName=Kenny&
      personLastName=Cambell&
      email=kenk5@happyjacksoftware.com&
      grRoles=squirmers:leader,squirmers:follower
  ```

Changing a user

This operation allows some properties of a user to be changed (administrative privileges are required to change many of these properties).

Form: `changeUser.api`

Required Arguments:

- `userName` - the name of the user to change
- `password` - the password of the user to change

Optional Arguments:

- `changeUser` - include if `userName` is the name of an administrator, `changeUser` will reflect the changes rather than `userName`
- `newPassword` - the new password for the user
- `newUserName` - a new email address login. This will send a verification email to the `newUserName` address.
- `removeRoles` - a comma-separated list of company roles to remove from this user (`userName` must be the name of an administrator)
- `addRoles` - a comma-seperated list of company roles to add to this user (`userName` must be the name of an administrator)

Discussion:

If no changeUser is provided, the changes will be applied to the user given by userName. Only administrative accounts may pass in changeUser, addRoles, and removeRoles. If newUserName is given (it must be a valid email) a verification email will be sent to the address specified. The change in the users login will not be made until the address is verified.

Examples:

- Change a users username and password as an administrator

  ```
  http://localhost:8080/kalendi/changeUser.api?
      userName=admin&
      password=changeme&
      changeUser=jdoe@gmail.com&
      newUserName=johncdoe@gmail.com&
      newPassword=jdoe123
  ```

- Add and Remove some roles from your account

  ```
  http://localhost:8080/kalendi/changeUser.api?
  ```
  ```

```
    userName=admin&
    password=changeme&
    addRoles=Supervisor,Management&
    removeRoles=Employee
```

Deleting a user

Users can be deleted from a company. This should be done with care because all of the user's calendars are deleted as part of this operation (must be an administrator).

Form: `deleteUser.api`

Required Arguments:

- `userName` - the administrator's user name
- `password` - the administrator's password
- `changeUser` - the email (and hence userName) of the user to be deleted

Examples:

- Delete a user

```
http://localhost:8080/kalendi/deleteUser.api?
   userName=jvb&
   password=jvb&
   changeUser=testuser@companydomain.com
```

Adding a Category

This command allows you to add a Category. There are three types of categories you can add with this call: Top Level Categories, Org Categories, Sub Categories. You must specify a new category name otherwise the call will return an error and fail to add the category.

Form: `addCategory.api`

Required Arguments:

- `userName` - a valid user name
- `password` - the password
- `catName` - the name of the category
- `orgCatName` - the name of the org category

Discussion:

If just the catName is given a Top Level Category will be created. If just an orgCatName is given then an Org Category will be created. Finally, if both catName and orgCatName are given a new Category will be made which is a child of the specified Org Category.

Examples:

- Add a Top Level Category

```
http://localhost:8080/kalendi/addCategory.api?
    userName=jvb&
    password=jvb&
    catName=Entertainment
```

- Add an Org Category

```
http://localhost:8080/kalendi/addCategory.api?
    userName=jvb&
```

```
    password=jvb&
    orgCatName=Outdoor Activities
```

- Add a Sub Category

```
http://localhost:8080/kalendi/addCategory.api?
    userName=jvb&
    password=jvb&
    orgCatName=Outdoor Activities&
  catName=Tennis
```

Deleting a Category

This command allows you to delete a Category. You can delete a regular category by passing in "catName", this will delete the category, and delete all references to that category from any events. You can also delete an Org Category by passing in "orgCatName".

Form: `deleteCategory.api`

Required Arguments:

- `userName` - a valid user name
- `password` - the password
- `catName` - the name of the category
- `orgCatName` - the name of the org category

Discussion:

If just the catName is given then just that category will be deleted. If the orgCatName is passed in, then all sub-categories of that orgCat will be deleted as well as the orgCat itself. If both arguments are passed in, then the regular Category will be deleted, as well as the Org Category.

Examples:

- Delete a Regular Category

```
http://localhost:8080/kalendi/deleteCategory.api?
    userName=jvb&
    password=jvb&
    catName=Entertainment
```

- Delete an Org Category (and all sub-categories)

```
http://localhost:8080/kalendi/deleteCategory.api?
    userName=jvb&
    password=jvb&
    orgCatName=Outdoor Activities
```

Getting all Categories

This command allows you to get all the Categories for the company.

Form: `getCategories.api`

Required Arguments:

- `userName` - a valid user name
- `password` - the password

Discussion:

You will get back an XML List of categories. These can be structured with Organizational Categories containing children categories, as well as simply top level categories.

Examples:

- Get all Categories

```
http://localhost:8080/kalendi/getCategories.api?
 userName=jvb&
 password=jvb&
<Categories>
<Category>CALibrate Portal</Category>
<Category>Red Category</Category>
<Category>Birthday</Category>
<Category>Holiday</Category>
<Category>Business</Category>
<orgcat name="Academics">
  <Category>Lectures</Category>
  <Category>Symposia</Category>
  <Category>Seminars</Category>
  <Category>Slideshows</Category>
  <Category>Speech / Talk</Category>
  <Category>Meetings</Category>
  <Category>Conferences</Category>
  <Category>Debates</Category>
  <Category>Training</Category>
</orgcat>
<orgcat name="Arts & Entertainment">
  <Category>Readings</Category>
  <Category>Music</Category>
  <Category>Exhibits</Category>
  <Category>Film</Category>
  <Category>Cultural</Category>
  <Category>Theatre</Category>
  <Category>Dance</Category>
  <Category>Art</Category>
</orgcat>
<orgcat name="Miscellaneous">
  <Category>Sorority/Fraternity</Category>
  <Category>Alumni</Category>
  <Category>Community Events</Category>
  <Category>Administrative</Category>
  <Category>Social Awareness</Category>
  <Category>Political</Category>
  <Category>Auction</Category>
  <Category>Campus Tours</Category>
  <Category>Educational</Category>
</orgcat>
<orgcat name="Sports & Fitness">
  <Category>Cowgirl Soccer</Category>
  <Category>Cowgirl Tennis</Category>
  <Category>Club Sports</Category>
  <Category>Cowgirl Volleyball</Category>
  <Category>Cowgirl Basketball</Category>
  <Category>Cowboy Wrestling</Category>
  <Category>Outdoor Adventure Program</Category>
  <Category>Cowboy Football</Category>
  <Category>Cowboy Basketball</Category>
</orgcat>
</Categories>
```

Adding an Organization

This command allows you add a new Organization along with roles to be associated with that Organization

Form: `addOrganization.api`

Required Arguments:

- `userName` - a valid user name
- `password` - the password
- `orgName` - the name of the new Organization

Optional Arguments:

- `orgRoles` - a comma-separated list of roles for the organization

Discussion:

You must provide a valid orgName, also if the orgName is already in use you must choose another. If you specify a list of orgRoles, they will all get added (duplicates roles will be ignored).

Examples:

- Add a new Organization

```
http://localhost:8080/kalendi/addOrganization.api?
 userName=jvb&
 password=jvb&
 orgName=Marketing
```

- Add a new Organization with Roles

```
http://localhost:8080/kalendi/addOrganization.api?
 userName=jvb&
 password=jvb&
 orgName=Marketing&
 orgRoles=Manager,Salesman,Accountant
```

Removing an Organization

This command allows you to delete an Organization.

Form: `deleteOrganization.api`

Required Arguments:

- `userName` - a valid user name
- `password` - the password
- `orgName` - the name of the new Organization

Discussion:

You must provide a valid orgName, and you must have administrative privileges to run this command.

Examples:

- Delete an Organization

```
http://localhost:8080/kalendi/deleteOrganization.api?
 userName=jvb&
 password=jvb&
 orgName=Marketing&
```

Editing an Organization

This command allows you to edit an Organization along with roles for that Organization

Form: `editOrganization.api`

Required Arguments:

- `userName` - a valid user name
- `password` - the password
- `orgName` - the name of the Organization

Optional Arguments:

- `newOrgName` - a new name for the organization
- `addRoles` - a comma-separated list of roles to add to the organization
- `removeRoles` - a comma-separated list of roles to remove from the organization

Discussion:

You must provide a valid orgName, also if the newOrgName is already in use you must choose another.

Examples:

- Edit an Organization Name

```
http://localhost:8080/kalendi/editOrganization.api?
 userName=jvb&
 password=jvb&
 orgName=Marketing&
 newOrgName=Customers
```

- Remove roles from an Organization

```
http://localhost:8080/kalendi/editOrganization.api?
 userName=jvb&
 password=jvb&
 orgName=Marketing&
 removeRoles=Manager,Salesman,Accountant
```

- Add roles to an Organization

```
http://localhost:8080/kalendi/editOrganization.api?
 userName=jvb&
 password=jvb&
 orgName=Marketing&
 addRoles=Manager,Salesman,Accountant
```

Getting a Preference Group

This command allows you to get a group of user preferences. Most of these preferences are only relevant to Kalendi's web-based graphical user interface, so these are not of interest to most API users. However, one preference group that is of interest to API users is "reminders" which contains preferences specifying what text should appear in reminders sent by Kalendi. As described in the next section, these preferences can be changed through the API. Another preference group that is relevant "moderated". This group contains a preference for how often to send emails to a calendar moderator informing them of pending events for that calendar.

Form: `getPrefs.api`

Required Arguments:

- `userName` - a valid user name
- `password` - the password
- `prefGroup` - the name of the preference group to get

Discussion:

If the `prefGroup` doesn't exist, an empty set of preferences will be returned. Only the preferences that have been set by this user are returned (some have default values which are not returned).

Examples:

- Get Reminder Preferences

```
http://localhost:8080/kalendi/getPrefs.api?
 userName=jvb&
 password=jvb&
 prefGroup=reminders
```

- Example Output:

```
<Preferences>
  <remText>$stTime$ - $endTime$: $description$ $date$</remText>
</Preferences>
```

Setting a Preference

This command allows you set a user preference.

Form: `setPreference.api`

Required Arguments:

- `userName` - a valid user name
- `password` - the password
- `prefName` - the name of the preference to change
- `prefValue` - the value to change the preference to

Optional Arguments:

- `changeUser` - the name of the new user to change preference on

Discussion:

You must provide a valid prefName, and you must have administrative privileges to pass in changeUser. The possible preference names include:

- `remText` - a template for the text inserted into the body of reminder emails that kalendi sends.
- `remSubj` - a template for the text inserted into the subject linke of reminder emails that kalendi sends
- `emailPending` - a value controlling when emails are sent to the moderator of a moderated calendar to inform them of newly submitted pending events. Possible values for this preference are:
  - never
  - hourly
  - every 2 hours
  - every 4 hours
  - daily @ noon

Any text can be included in the values of the `remText` and `remSubj` templates and in addition, the following variables will have values substituted from the event for which the reminder is sent. These variables include:

- $caption$
- $stTime$ - the starting time and date of the event
- $endTime$ - the ending time and date of the event
- $description$

Examples:

- Change a preference

```
http://localhost:8080/kalendi/setPreference.api?
 userName=jvb&
 password=jvb&
```

```
prefName=remSubj&
prefValue=Reminder: $caption$
```

Getting all Custom Properties

This command allows you to get all the Custom Properties for the company.

Form: `getCustomProps.api`

Required Arguments:

- `userName` - a valid user name
- `password` - the password

Discussion:

You will get back an XML List of custom props. All attributes of the custom prop (ie. required, help, etc.) will be specified in the XML.

Examples:

- Get all Custom Properties

```
http://localhost:8080/kalendi/getCustomProps.api?
 userName=jvb&
 password=jvb
```

- Example XML:

```
 <customprops>
<customprop type="String" name="Enrollment" external="" required="n"
help=""></customprop>
<customprop type="Integer" name="fooey" external="y" required="n"
help=""></customprop>
<customprop type="String" name="dyna test" external="n" required="n"
help=""></customprop>
<customprop type="List" name="testList" required="n" external="y" help="">
 <value>Option 1</value>
 <value>Option 2</value>
</customprop>
 </customprops>
```

Adding a Custom Property

This command allows you to add a Custom Property. This operation requires administrative privileges.

Form: `addCustomProp.api`

Required Arguments:

- `userName` - a valid user name
- `password` - the password
- `required` - whether or not the custom property is required for every event
- `external` - whether or not this custom property is shown when you view events through the kalendi web interface
- `type` - the type of the custom property
- `help` - the help text that should be displayed to users
- `values` - the possible values (only used if `type` is "List" or "Boolean")

Discussion:

You must provide a unique alpha-numeric (only a-z 0-9 allowed) `label` for the custom property. If you do not provide a unique or alpha-numeric `label` the API will return an error.

If you set `required` to 'y', all events that get created will need to set that custom property or they will fail to be created.

Examples:

- Add a String CustomProp

```
http://localhost:8080/kalendi/addCustomProp.api?
    userName=jvb&
    password=jvb&
    label=Location&
  type=String
```

- Add a required CustomProp

```
http://localhost:8080/kalendi/addCustomProp.api?
   userName=jvb&
   password=jvb&
   label=Contact Name&
  required=y&
  type=String
```

- Add an non-external CustomProp

```
http://localhost:8080/kalendi/addCustomProp.api?
   userName=jvb&
   password=jvb&
   label=Hidden Prop&
  external=n&
  type=String
```

- Add a List CustomProp

```
http://localhost:8080/kalendi/addCustomProp.api?
   userName=jvb&
   password=jvb&
   label=City&
  type=List&
  values=San Diego,New York,Dallas
```

- Add a Boolean CustomProp

```
http://localhost:8080/kalendi/addCustomProp.api?
   userName=jvb&
   password=jvb&
   label=Free&
  type=Boolean&
  values=true,false
```

- Add a MapIt CustomProp

```
http://localhost:8080/kalendi/addCustomProp.api?
   userName=jvb&
   password=jvb&
   label=Address&
  type=MapIt
```

Editing a Custom Property

This command allows you to edit a custom property. You can change everything about a custom property including its label, possible values, and several other properties given in the arguments section.

Form: `editCustomProp.api`

Required Arguments:

- `userName` - a valid user name
- `password` - the password
- `label` - the label of the custom preoprty you want to change

Optional Arguments:

- `newLabel` - the new label of the custom property
- `type` - the new type of the custom property (must be a valid custom property type)
- `required` - whether or not the custom property is required for events
- `external` - whether or not the custom property is external
- `help` - the help text for the custom property
- `values` - a comma separated list of possible values for the custom property

Discussion:

You must provide a good label otherwise the call will fail because it cannot find the right custom property. You must also pass in a valid newLabel if you want to change the label. The external and required arguments must have values of either "y" or "n". You cannot change the type (except changing it to String) or the values of the custom property if there are events that use it.

Examples:

- Edit a custom property

```
http://localhost:8080/kalendi/editCustomProperty.api?
 userName=jvb&
 password=jvb&
 label=Test&
 newLabel=Organization Type&
 type=List&
 required=y&
 external=n&
 help=The type of organization sponsoring the event&
 values=Non Profit,For Profit,Government
```

Adding a Company Role

This command allows you to add a Company Role. You must specify a valid role name (alpha-numeric character only... a-z 0-9). To add a company role, the user must have admin privileges.

Form: `addCompanyRole.api`

Required Arguments:

- `userName` - a valid user name
- `password` - the password
- `roleName` - the name of the company role

Discussion:

If you specify an invalid `roleName` the API will return an error.

Examples:

- Add a Company Role

```
http://localhost:8080/kalendi/addCompanyRole.api?
    userName=jvb&
    password=jvb&
```

```
    roleName=Employee
```

Removing a Company Role

This command allows you to delete a Company Role.

Form: `deleteCompanyRole.api`

Required Arguments:

- `userName` - a valid user name
- `password` - the password
- `roleName` - the name of the Company Role to delete

Discussion:

You must provide a valid roleName, and you must have administrative privileges to run this command.

Examples:

- Delete a Company Role

```
http://localhost:8080/kalendi/deleteCompanyRole.api?
 userName=jvb&
 password=jvb&
 roleName=Employee
```

Adding a Reminder Address

This command allows you add a reminder address. You can add phone numbers and email addresses. You can also add a reminder address for ANOTHER user (if you have admin privileges).

Form: `addReminderAddress.api`

Required Arguments:

- `userName` - a valid user name
- `password` - the password
- `type` - the type of address ('email' or 'phone')
- `address` - the reminder address

Optional Arguments:

- `changeUser` - the user to add the address for (requires admin privileges)
- `carrier` - the phone service carrier (required if `type` is 'phone')

Discussion:

If the `type` is 'email', `address` must be a properly formatted email address.

If `type` is 'phone', the `address` must be a phone number formatted like '1234567890' ie. 10 digits, the first 3 digits being the area-code. If `type` is 'phone' you must also provide a `carrier`.

Valid carriers are "Verizon", "Quest", "AT&T Wireless", "Cingular", "Sprint PCS", "T-Mobile", and "Metro PCS".

If the `type` is 'email' and `carrier` is supplied, it will be ignored. Upon a successful call to addReminderAddress.api, an email or SMS message will be sent to the `address` given. This message will contain a link to the verify.api call with a unique code attached. The reminder address will not be valid until the link provided is processed.

Examples:

- Add an email address for another user

```
http://localhost:8080/kalendi/addReminderAddress.api?
 userName=jvb&
 password=jvb&
 type=email&
 address=jdoe@gmail.com&
 changeUser=otherUser
```

- Add a phone number for the API user

```
http://localhost:8080/kalendi/addReminderAddress.api?
 userName=jvb&
 password=jvb&
 type=phone&
 address=3074561234&
 carrier=Cingular
```

Deleting a Reminder Address

This command allows you delete a reminder address. You can delete a reminder address for ANOTHER user (if you have admin privileges).

Form: `deleteReminderAddress.api`

Required Arguments:

- `userName` - a valid user name
- `password` - the password
- `address` - the reminder address to delete

Optional Arguments:

- `changeUser` - the user to delete the address for (requires admin privileges)

Discussion:

Any reminder addresses that use the given address will be deleted for the user specified. If there are no addresses found an error is returned.

Examples:

- Delete an email address for another user

```
http://localhost:8080/kalendi/deleteReminderAddress.api?
 userName=jvb&
 password=jvb&
 address=jdoe@gmail.com&
 changeUser=otherUser
```

Verifying a Reminder Address

This command allows you verify a reminder address, given a unique code.

Form: `verify.api`

Required Arguments:

- `code` - the code that was sent in an email or SMS message

Discussion:

You must pass in a valid code. If the code isn't found, an error will be returned. If the code given is found then the associated reminder address will be marked as verified and will be usable for reminders.

Examples:

- Verify a reminder address

```
http://localhost:8080/kalendi/addReminderAddress.api?
 code=fb6eed80-1d7e-11df-b8d8-001c42000009
```

Using Token Based Authentication

This command allows you to use token based authentication for API calls. A user may create tokens that has limited privileges to their calendars and have limited lifetimes. These tokens can be given out to allow access through the Kalendi API to those calendars. Tokens are used to make API calls by including them as the value of the cookie kalendiAPI with the token returned by this call. When using this method of login, the username of the user who created the token is supplied and no password should be given.

Form: `getToken.api`

Required Arguments:

- `userName` - a valid user name
- `password` - the password

Optional Arguments:

- `expireTime` - the number of minutes before the token will expire (with default value 15)
- `perm` - a permission for one of the user's calendars in the form calendarID:permission_name
- `perms` - a comma-separated list of permissions as described above

Discussion:

To use token based authentication, simply retrieve the token from the result of this call. Then in any future API calls, pass the token as the value of the cookie kalendiAPI and you will have the access specified in the permissions. If no permissions are specified, the token user has the same access to event calls on calendars as the user. The arguments `perm` and `perms` are alternative methods of providing permissions. If `perm` is used, it can be repeated to provide multiple permissions. However, `perm` arguments and the `perms` argument cannot both be included.

Examples:

- Use Token Authentication

```
http://localhost:8080/kalendi/getToken.api?
    userName=jvb&
    password=jvb&
    perms=100:subscribe,100:append
```

- Example Output

```
<token>0f5225705d20b9dfa2d5601c520112a9</token>
```

Using Different Authentication Methods

This command allows you set an alternative authentication method for your API Login.

Form: `setAccessor.api`

Required Arguments:

- `userName` - a valid user name
- `password` - the password

Optional Arguments:

- `referrer` - the referrer from which you will login to the API
- `host` - the host from which you will login to the API

Discussion:

There are three types of alternate login methods: By Referrer, By Host, By Token (getToken.api call). If you pass in the `referrer` argument you will now be able to use that method, and likewise with the `host` argument.

When using the `referrer` and `host` methods of login, you may make any API call (that your account has access to) without passing a password. The request must be made from the host or referrer that you specify in the setAccessor call for this to work.

Examples:

- Use Referrer Authentication

```
 http://localhost:8080/kalendi/setAccessor.api?
userName=jvb&
password=jvb&
referrer=http://www.happyjacksoftware.com
```

- Use Host Authentication

```
 http://localhost:8080/kalendi/setAccessor.api?
userName=jvb&
password=jvb&
host=www.example1.com
```

Removing an Accessor

This command allows you to remove an accessor that was set by the setAccessor.api call.

Form: `removeAccessor.api`

Required Arguments:

- `userName` - a valid user name
- `password` - the password

Optional Arguments:

- `referrer` - the referrer which you would like to remove
- `host` - the host which you would like to remove

Discussion:

If you pass in a referrer that you perviously set, it will be removed and you will not be able to authenticate from that referrer anymore (unless you provide a password), and the same goes for `host`.

Examples:

- Remove Referrer Authentication

```
 http://localhost:8080/kalendi/removeAccessor.api?
userName=jvb&
password=jvb&
referrer=http://www.happyjacksoftware.com
```

- Remove Host Authentication

```
 http://localhost:8080/kalendi/removeAccessor.api?
userName=jvb&
password=jvb&
host=www.example1.com
```