11 11 $\TeX$ $\LaTeX$

Using Libical

Eric Busboom (eric@softwarestudio.org) January 2000

1 Introduction

Libical is an Open Source implementation of the iCalendar protocols and protocol data units. The iCalendar specification describes how calendar clients can communicate with calendar servers for users can store their calendar data and arrange meetings with other users.

Libical implements the following specifications and protocols

iCal Core 2445 iTIP 2446 iMIP 2447 iRIP draft CAP draft

(The current version, 0.15, does not implement iRip or CAP.)

This documentation assumes that you are familiar with the iCalendar standards RFC2445 and RFC2446. these specifications are online on the CALSCH webpage at:

```
http://www.imc.org/ietf-calendar/
```

1.1 The libical project

This code is under active development. If you would like to contribute to the project, you can contact me, Eric Busboom, at eric@softwarestudio.org. The project has a webpage at

http://softwarestudio.org/libical/index.html

and a mailing list that you can join by sending the following mail:

```
To: minimalist@softwarestudio.org
Subject: subscribe libical
```

1.2 License

The code and datafiles in this distribution are licensed under the Mozilla Public License. See http://www.mozilla.org/NPL/MPL-1.0.html for a copy of the license. Alternately, you may use libical under the terms of the GNU Library General Public License. See http://www.fsf.org/copyleft/lesser.html for a copy of the LGPL.

This dual license ensures that the library can be incorporated into both proprietary code and GPL'd programs, and will benefit from improvements made by programmers in both realms. I will only accept changes into my version of the library if they are similarly dual-licensed.

1.3 Purpose & Goals

1.4 Document version

$Id: UsingLibical.lyx,v 1.4 2000/02/18 23:06:04 eric Exp eric $

2 Building the Library

Libical uses autoconf to generate makefiles, although it uses none of the autoconf flags to influence the compilation. It should built with no adjustments on Linux, FreeBSD and Solaris.

3 Structure

The iCal calendar model is based on four types of objects: components, properties, values and parameters.

Properties are the fundamental unit of information in iCal, and they work a bit like a hash entry, with a constant key and a variable value. Properties may also have modifiers, called parameters. In the iCal content line

```
ORGANIZER;ROLE=CHAIR:MAILTO:mrbig@host.com
```

The property name is "ORGANIZER," the value of the property is "mrbig@host.com" and the "ROLE" parameter specifies that Mr Big is the chair of the meetings associated with this property.

Components are groups of properties that represent the core objects of a calendar system, such as events or timezones.

The central goal of libical is to parse iTIP data into an internal representation of Components, Properties, Parameters an Values, and to allow the user to manipulate the data in various ways

3.1 Components

3.2 Properties

3.3 Values

3.4 Parameters

3.5 Enumerations

3.6 Types

3.7 The Parser

3.8 Restrictions

3.9 Memory Management

4 Differences From RFCs

Libical has been designed to follow the standards as closely as possible, so that the key objects in the standards are also keey objects in the library. However, there are a few areas where the specifications are (arguably) irregular, and following them exactly would result in an unfriendly interface. These deviations make libical easier to use by maintaining a self-similar interface.

4.1 Pseudo Components

Libical defines components for groups of properties that look and act like components, but are not defined as components in the specification. XDAYLIGHT and XSTANDARD are notable examples. These pseudo components group properties within the VTIMEZONE components. XDAYLIGHT starts with "BEGIN:DAYLIGHT" and ends with "END:DAYLIGHT, just like other components, but is not defined as a component in RFC2445. (See RFC2445, page 61) In Libical, it is a component.

There are also pseudo components that are conceptually derived classess of VALARM. RFC2446 defines what properties may be included in each component, and for VALARM, the set of properties it may have depends on the value of the ACTION property.

For instance, if a VALARM component has an ACTION property with the value of "AUDIO," the component must also have an "ATTACH" property. However, if the ACTION value is "DISPLAY," the component must have a DESCRIPTION property.

To handle these various, complex restrictions, libical has pseudo components for each type of alarm: XAUDIOALARM, XDISPLAYALARM, XEMAILALARM and XPROCEDUREALARM.

4.2 Combined Values

Many values can take more than one type. TRIGGER, for instance, can have a value type of with DURATION or of DATE-TIME. These multiple types make it difficult to create routines to return the value associated with a property.

It is natural to have interfaces that would return the value of a property, but it is cumbersone for a single routine to return multiple types. So, in libical, properties that can have multiple types are given a single type that is the union of their RFC2445 types. For instance, in libical, the value of the TRIGGER property resolves to struct icaltriggertype. This type is a union of a DURATION and a DATE-TIME.

4.3 Multi-Valued Properties

Some properties, such as CATEGORIES have only one value type, but each CATEGORIES property can have multiple value instances. This also results in a cumbersome interface – CATEGORIES accessors would have to return a list while all other accessors returned a single value. In libical, all properties have a single value, and multi-valued properties are broken down into multiple single valued properties during parsing. That is, an input line like,

```
CATEGORIES: work, home
```

becomes in libical's internal representation

```
CATEGORIES: work
CATEGORIES: home
```

Oddly, RFC2445 allows some multi-valued properties (like FREEBUSY) to exist as both a multi-values property and as multiple single value properties, while others (like CATEGORIES) can only exist as single multi-valued properties. This makes the internal representation for CATEGORIES illegal. However when you convert a component to a string, the library will collect all of the CATEGORIES properties into one.

5 Implementation Limitations

6 Using libical

6.1 Creating Components

There are three ways to create components in Libical: creating individual objects and assembling them, building entire objects in massive vaargs calls, and parsing a text file containing iCalendar data.

6.1.1 Constructor Interfaces

Using constructor interfaces, you create each of the objects seperately and them assemble them in to components:

```
event = icalcomponent_new(ICAL_VEVENT_COMPONENT);
icalcomponent_add_property(event, icalproperty_new_dtstamp(atime) );
icalcomponent_add_property(event,icalproperty_new_uid(strdup("guid-1.host1.com")) );
property=icalproperty_new_organizer(strdup("mrbig@host.com"));
icalproperty_add_parameter(property,icalparameter_new_role(ICAL_ROLE_CHAIR) );
icalcomponent_add_property(event,property);
```

6.1.2 vaargs Constructors

6.1.3 Parsing Text Files

6.2 Accessing Components

6.2.1 Finding Components

6.2.2 Removing Components

Removing an element from a list while iterating through the list can cause problems, since you will probably be removing the element that the internal iterator points to. This will result in the iteration loop terminating immediately after removing the element. To avoid the problem, you will need to step the iterator ahead of the element you are going to remove, like this:

```
for(c = icalcomponent_get_first_component(parent_comp,ICAL_ANY_COMPONENT);
       c != 0;
       c = next
{
    next = icalcomponent_get_next_component(parent_comp,ICAL_ANY_COMPONENT);
    icalcomponent_remove_component(parent_comp,c);
}
```

6.2.3 Finding Properties

6.2.4 Removing Properties

6.2.5 Getting Values

6.2.6 Setting Values

6.2.7 Getting Parameters

6.2.8 Setting Parameters

6.2.9 Removing Parameters

6.2.10 Checking Component Validity

6.3 Storing Objects

The libical distribution inclues a seperate library, libicalss, that allows you to store iCal component data to disk in a variety of ways. This library is documented seperately.

6.4 Memory Management

Libical relies heavily on dynamic allocation for both the core objects and for the strings used to hold values. Some of this memory the library caller owns and must free, and some of the memory is managed by the library. Here is a summary of the memory rules.

1)

If the function name has "new" in it, the caller gets control of the memory. (such as icalcomponent_new(), or icalproperty_new_clone())

2)

If you got the memory from a routine with new in it, you must call the corresponding *_free routine to free the memory. (Use icalcomponent_free() to free objects created with icalcomponent_new())

3)

If the function name has "add" in it, the caller is transfering control of the memory to the routine. (icalproperty_add_parameter())

4)

If the function name has "remove" in it, the caller passes in a pointer to an object and after the call returns, the caller owns the object. So, before you call icalcomponent_remove_property(comp,foo), you do not own "foo" and after the call returns, you do.

5)

If the routine returns a string, libical owns the memory and will put it on a ring buffer to reclaim later. You'd better strdup() it if you want to keep it, and you don't have to delete it.

6.5 Error Handling

icalerror_errno. Return values. #defines. icalerror_stop_here

6.5.1 Return values

6.5.2 icalerrno

6.5.3 Component errors

6.6 Naming Standard

Structures that you access with the "struct" keyword, such as "struct icaltimetype" are things that you are allowed to see inside and poke at.

Structures that you access though a typedef, such as "icalcomponent" are things where all of the data is hidden.

Component names that start with "V" are part of RFC 2445 or another iCal standard. Component names that start with "X" are also part of the spec, but they are not actually components in the spec. However, they look and act like components, so they are components in libical. Names that start with "XLIC" or "X-LIC" are not part of any iCal spec. They are used internally by libical.

Enums that identify a component, property, value or parameter end with "_COMPONENT," "_PROPERTY," "_VALUE," or "_PAAMETER"s

Enums that identify a parameter value have the name of the parameter as the second word. For instance: ICAL_ROLE_REQPARTICIPANT or ICAL_PARTSTAT_ACCEPTED.

The enums for the parts of a recurarance rule and request statuses are irregular.

7 Useful Recipies

Iteration

Copying components. Remember that you must clone or remove an object before putting in on anothr list.

Finding compliance errors

8 Performance

Checking restrictions is computationally expensive

9 Hacks and Bugs