Calendaring Specification
Interoperability of Web-based Calendar and Palm pilot

MS Project

Major Professor: Dr. Kal Toth

Madhuri Gourishetty

SID 930-30-6546

Dept. of Computer Science

OSU

CONTENTS

1.0 Abstract

Interoperability is the ability to enable different systems to work together and exchange data. Interoperability between different systems is achieved by using common standards and specifications. This paper talks about our research work on interoperability in Calendaring and Scheduling. There are many calendaring and scheduling software products on the market but only few of them are interoperable. The goal of my research was to make an existing web-based calendaring system ("The Wise Net" or TWN for short) interoperable with other calendaring systems, in particular, the Palm Pilot. A framework is developed for web based TWN calendar after understanding the interoperability issues that are dealt in other systems like Outlook, iPlanet calendar etc. After a complete understanding of the systems it was found that Outlook and other such popular calendaring systems are based on iCal and iTIP protocol standards. My aim was to discover how to make The Wise Net Calendar support these two protocols and develop a common specification for calendaring and scheduling that would support interoperability of TWN and Palm Pilot calendaring systems.

An initial working prototype is built to make the TWN calendar interoperable with palm pilot. This application synchronizes TWN calendar with the palm pilot calendar. A prototype is also built for synchronizing address book. TWN users can have a mirror image of their web based calendars on their palm pilots. Conduit Development Kit (CDK403) is used to develop the application. Conduits are software plug-ins for the HotSync Manager application. They exchange and synchronize data between a desktop computer and a Palm OS platform handheld computer.

2.0 Introduction

The Wise Net (TWN) product supports web-enabled information services for the real estate industry. It provides a real estate database, search tools, user accounts, email, contacts, calendaring and other work group services. The TWN Calendar is a customizable Internet-based calendaring solution that enables group scheduling and personal calendaring for real estate agents and their customers. Through a web browser, subscribers can insert and update appointments, publish their calendars for viewing by others and establish reminders and other notices.

The TWN calendar is designed to perform Group Scheduling. In particular, this calendar, allows a user to create an event to which other attendees are invited. The attendees can accept or decline invitations. Attendees are typically people on the same calendar server. So it means that it is not interoperable with other calendaring systems. If the calendar can interact with other calendars on different servers using standard messaging and protocols, it is considered interoperable. Standards like iCal, iTIP and iMIP are already defined for calendars. If all the calendars are designed according to these standards, then they can be interoperable. Outlook and iPlanet are interoperable. So the user of Outlook can send invitations to the user of iPlanet.

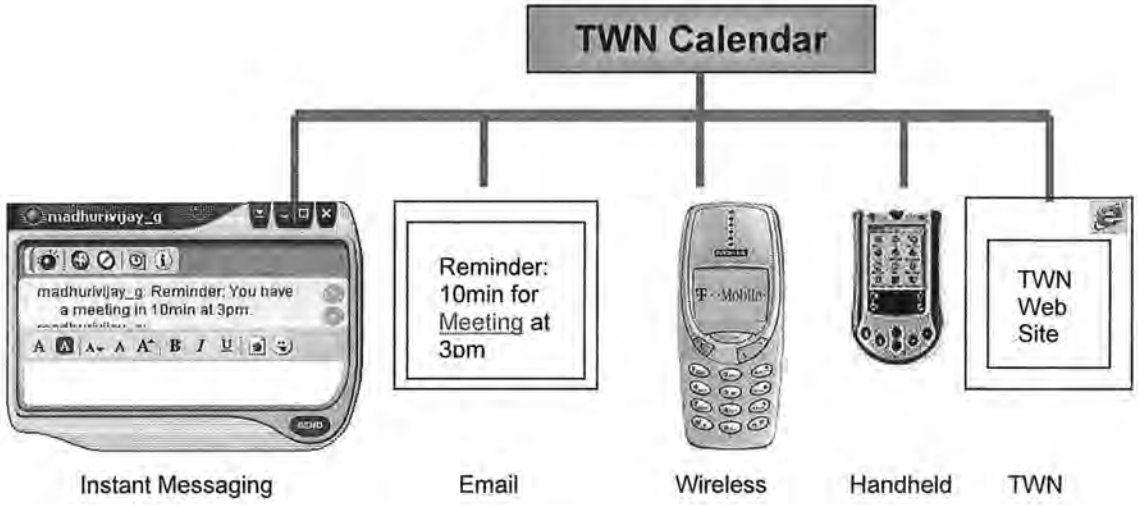
TWN was not implemented according to the iCal standards, so a specification has been written to map the existing calendaring system to make it interoperable.

Interoperability, according to dictionary meaning is "The ability of software and hardware on multiple machines from multiple vendors to communicate". So a calendar talking to the calendar of other devices like Palm Pilot, Pocket PC etc. is also interoperability. By including this feature, all the users of TWN can communicate with their Palm Pilots and synchronize their web based calendar with their Palm Pilots. The users need not be online to check their appointments.

TWN provides the following benefits:

- Simplified system management, online backup/restore, entire database backup/restore
- Powerful search agent
- Login facility for all the users
- Individuals can store their personal contacts in an address book and store tasks and appointments in a calendar
- Automatic e-mail notification of appointments and appointment reminders sent to selected recipients.
- Daily, weekly, monthly, yearly, and comparison views
- Web-based Group Scheduling
- Ability to delegate calendar ownership to others who may act on behalf of the primary owner.
- To-do management through task lists
- Instant messaging system. According to the preferences given by the users, a message is sent to others through different channels like email, cell, palm etc. So the user need not be online to get the messages.

In order for online calendar information to be useful, it must be readily available. Though users may have access to a web browser much of the time, there will be times when the information must be available on other devices: a cellular phone, a pager, a PDA. The diagram below shows the different devices with which the TWN calendar interacts. If there is a new event then the message is sent to any one of the channels. All the messages are by default seen on the TWN calendar.



3.0 Standards for Calendaring and Scheduling Interoperability

Standards are important for interoperability. If all the systems follow these standards then communication between them will be easy. We will see some important calendaring standards here.

3.1 Internet Calendaring and Scheduling Core Object Specification (iCalendar)

This memo has been defined to provide the definition of a common format for openly exchanging calendaring and scheduling information across the internet. This memo defines the format for specifying iCalendar object methods. An iCalendar object method is a set of usage constraints for the iCalendar object. For example, these methods might define scheduling messages that request an event be scheduled, reply to an event request, send a cancellation notice for an event, modify or replace the definition of an event, provide a counter proposal for an original event request, delegate an event request to another individual, request free or busy time, reply to a free or busy time request, or provide similar scheduling messages for a to-do or journal entry calendar component. Basically this standard gives the variables involved in calendaring and scheduling process. The iCalendar format is suitable as an exchange format between applications or systems. This will enable the calendar object to be exchanged using several transports, including but not limited to SMTP, HTTP, a file system, desktop interactive protocols such as the use of a memory-based clipboard or drag/drop interactions, wired-network transport, or some form of unwired transport such as infrared might also be used.

For details of this standard, see Appendix A

3.2 iCalendar Transport-Independent Interoperability Protocol(iTIP)

Scheduling Events, BusyTime, To-dos and Journal Entries: This standard is different from the above one. iCalendar specifies the variables, whereas this standard gives the methods involved in scheduling. This document specifies how calendaring systems use iCalendar objects to interoperate with other calendar systems. It does so in a general way so as to allow multiple methods of communication between systems. The standard outlines a model for calendar exchange that defines both static and dynamic event, to-do, journal and free/busy objects. Static objects are used to transmit information from one entity to another without the expectation of continuity or referential integrity with the original item. Dynamic objects are a superset of static objects and will gracefully degrade to their static counterparts for clients that only support static objects.
For details of this standard, see reference Appendix A.

Based upon these standards, a requirements specification is made for TWN to make it interoperable with other calendaring systems like Outlook, iPlanet etc. The requirements document can be seen in Appendix A.

The next part of this report describes the implementation of interoperability of TWN and palm pilot. For communicating between palm and browser, Conduit Development Kit (CDK) is installed on the client PC. Next section explains the CDK and how it is used in our project.

4.0 Conduit Development

There are six software components involved in using a handheld:
• **Desktop applications**, which are developed by a developer, run on a desktop computer and operate on data that is sent to or retrieved from a handheld.

• **Palm OS applications** are developed to run on any Palm Powered handheld. These are also referred to as **handheld applications**.

• The **HotSync Manager** application runs on a desktop computer and communicates with a handheld. When the user presses the HotSync button, HotSync Manager awakens and calls each of the conduits that are properly installed and configured on the user's desktop computer.

• The **HotSync client** launches on the handheld when the user presses the HotSync button on the cradle. This handheld application wakes up HotSync Manager and responds to Sync Manager requests to access databases on the handheld.

• **Conduits** are the programs that HotSync Manager runs to interact with specific data on the handheld. You use the Conduit Development Kit to create conduits. Generally, when a handheld application is written, a conduit is written to synchronize its data with data on the desktop.

Conduits also transfer, import/export data, or cause Palm OS applications to be installed. Conduits reside on the desktop computer and are run by the HotSync Manager program when the user places a handheld into the cradle and presses the HotSync button.

• The **Sync Manager API** provides a programmatic interface that conduits use for communicating with a handheld. This communications API allows the conduit to remain independent of the connection type between a handheld and a desktop computer.

A conduit is nothing but a simple program which communicates with the desktop computer. A conduit can be written in any language – C, C++, Java, and COM. For our project, Java is chosen as the language of communication.

Conduits synchronize data for a specific application on the handheld with the desktop computer. Conduits perform the following tasks - open and close databases on the handheld and add, delete, and modify records on the handheld and desktop computer, converting formats as required. There are different databases on the handheld like address database and calendar database, to-do list and memo database.

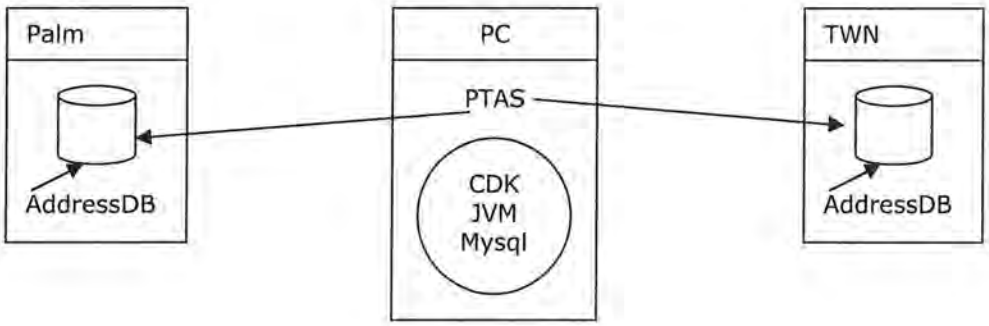
The next section gives the design of the whole system.

5.0 Palm TWN Address Synchronization (PTAS)

5.1 Design Flow

Objective is to synchronize the palm address database with TWN contacts table.

5.1.1 Low Level Architecture



Three components are involved in this architecture. Each component is explained in detail here.

- Palm: Palm pilot stores all the contacts in Address database (AddressDB).
- TWN: The TWN server has address table (AddressDB) to store the contacts for every user like the palm.
- PTAS – Palm TWN Address Synchronization: This is a java based program used to synchronize the two databases – palm and TWN. After the synchronization is performed

these two databases are the same. They are the mirror images of each other. PTAS is compiled to get an executable and this connects to the two databases – details are given in later sections.

5.1.2 High Level Architecture

The high level architecture is shown in figure 2. This shows the basic architecture of the synchronization process. There are different blocks in the diagram. Explanation is given for each block.

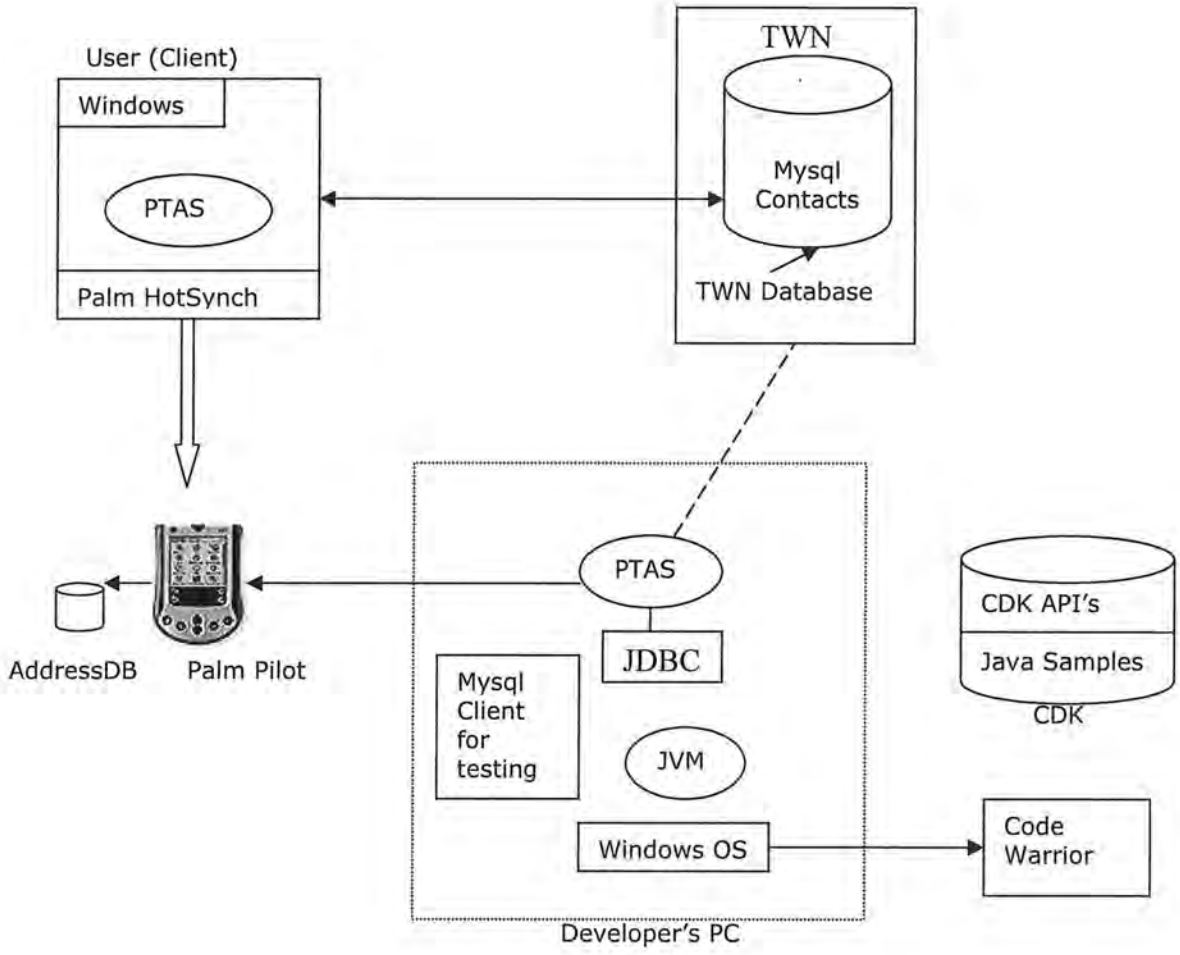
CDK (Conduit Development Kit): CDK4.0 is installed from the palm source website. This kit is very useful for developing conduits. The CDK provides a standard library, which has all the java files and class files. This library is very useful for writing conduits. This gives a basic idea how to access the palm databases. Along with the library, few sample programs are provided which synchronize the palm databases with the desktop databases, which come with palm SDK. A custom java file is created using the sample programs. This idea is called as PTAS (Palm TWN Address Synchronization).

TWN: Mysql server is installed on owl machine (TWN server). A table called as 'ADDRESS' is created in a database, which is on owl server. This table is synchronized with the contacts table in the palm.

Palm: Palm has a database called as AddressDB to store the contacts of a user. This database is accessed (read and write) using the API's provided by CDK.

Developer: The conduit developer uses Windows OS. Mysql client is installed on this computer (this is just for testing). Java is used as a tool for all the applications. JDK1.3 is installed. The applications are compiled and executed in Code Warrior which is a Windows based IDE. Mysql JDBC driver is installed to connect the application to the database. The application is nothing but a conduit. CDK is also installed on the developer PC. PTAS (Palm TWN Address Synchronization) is a java conduit, which synchronizes the palm AddressDB and TWN contacts table. PTAS is developed using the Java API's provided by the CDK. Implementation details are explained in the later sections.

User (Client): The user is assumed to have a windows OS which supports JVM. The user desktop should also have Palm software, which helps in connecting to a palm pilot. PTAS conduit is installed on the client side (user side). This will basically be a class or an executable.


TWN
User (Client)
Windows
PTAS
Palm HotSynch
Mysql
Contacts
TWN Database
PTAS
JDBC
Mysql
Client
for
testing
JVM
Windows OS
Developer's PC
CDK API's
Java Samples
CDK
Code
Warrior
AddressDB
Palm Pilot

5.2 Database Schema

Address table

This table exists on both the palm and TWN.

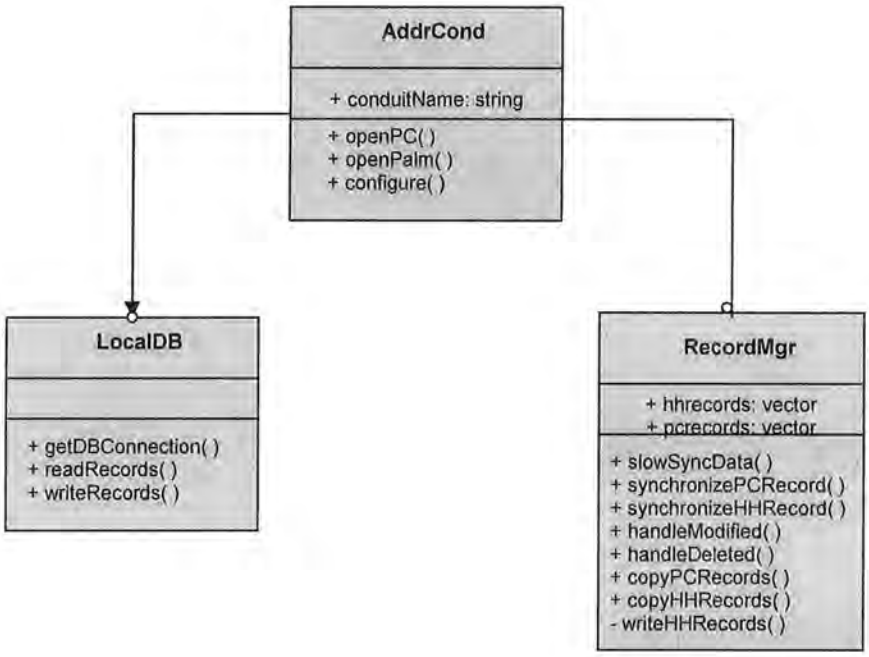
Name	Type	Null	Default
ID	int(3) unsigned	Yes	0
CATEGORYINDEX	int(5) unsigned	Yes	0
NAME	varchar(25)	Yes	null
FIRST_NAME	varchar(20)	Yes	null
TITLE	varchar(10)	Yes	null
COMPANY	varchar(10)	Yes	null
PHONE1	varchar(10)	Yes	null
PHONE2	varchar(12)	Yes	null
PHONE3	varchar(12)	Yes	null
PHONE4	varchar(12)	Yes	null
PHONE5	varchar(12)	Yes	null
PHONELABEL1	int(12) unsigned	Yes	0
PHONELABEL2	int(12) unsigned	Yes	1
PHONELABEL3	int(12) unsigned	Yes	2
PHONELABEL4	int(12) unsigned	Yes	3
PHONELABEL5	int(12) unsigned	Yes	4
ADDRESS	varchar(25)	Yes	null
CITY	varchar(12)	Yes	null
STATE	varchar(10)	Yes	null
COUNTRY	varchar(10)	Yes	null
ZIPCODE	varchar(10)	Yes	null
CUSTOM1	varchar(10)	Yes	null
CUSTOM2	varchar(10)	Yes	null
CUSTOM3	varchar(10)	Yes	null
CUSTOM4	varchar(10)	Yes	null
NOTE	varchar(255)	Yes	null
ISARCHIVED	varchar(5)	Yes	false
ISDELETED	varchar(5)	Yes	false
ISNEW	varchar(5)	Yes	true
ISMODIFIED	varchar(5)	Yes	false
ISPRIVATE	varchar(5)	Yes	false

Description of the database

- Every record has a unique record ID
- There are different categories provided in palm like public/private/confidential. Each category has a unique index.
- Each address record stores all the contact information of a person like name, company, address, phone etc.
- isDeleted, isNew and isModified are called as status flags. These flags are there in both the databases. These are used for synchronization.
- For example, if the palm contact has isNew as true then this record is copied into the TWN DB. This is explained more clearly in the next section.

- By comparing the status flags, synchronization takes place.
- Fast synchronization is considered for our research. There is also slow synchronization.
- Archive records are not considered.
- If palm has a record where isNew is true and also isDeleted true then this record is not added to the TWN during synchronization
- So we have to check all the deleted records first
- If palm has a record where isNew is true and isModified is true then only the modified record is copied.
- If the palm has a record where isModified is true and isDeleted is true then this record is not copied. These are explained in the table.

5.3 Class diagram



Description of the classes:

AddrCond is the main class, which interacts with the other classes

- Connection to TWN database is made using the JDBC driver using the following command.

Class.forName("org.gjt.mm.mysql.Driver");
connection=DriverManager.getConnection("jdbc:mysql: 128.193.40.80", "user", "password");

- Connection to Palm address database is made using the API's provided by CDK403.

db=SyncManager.openDB("AddressDB",0,SyncManager.OPEN_READ|
SyncManager.OPEN_WRITE);

- All the handheld records are read into a Java Vector - hhRecords
- All the TWN records are also read into a Java Vector – pcRecords (using LocalDB class)

- The program checks whether to perform fast synchronization or slow synchronization.

Slow Synchrnization: SlowSync performs slow mirror synchronization of records. SlowSync is used when the statusFlags can not be relied on for accuracy because they could have been reset during a previous synchronization with a different PC. Therefore, the slowSync method must read and process all the records from the device regardless of their status values. If the HH is HotSynced with another PC, then the statusFlags are wiped out. SlowSync uses the Backup file, which is a copy of the file after the last HotSync, to determine which records have been added, changed, or deleted on the HH since the last HotSync. To perform a SlowSync, every record must be read in from the HH, as opposed to the FastSync which reads only the modified records.

Fast Synchronization: Performs fast synchronization of records by reading and writing only the modified records to and from the Handheld. It relies on the statusFlags for each record to be accurate.

- In this case, a user will always connect to the same database (TWN). So fast synchronization makes more sense for this project.
- The two vectors (hhRecords and pcRecords) are synchronized in the RecordManager class
- The records are written back to their respective databases.

RecordManager Class: This class compares the two vectors (records) which are from different databases.

- Each record is compared one by one
- The records are compared using the statusflags– isNew(N), isModified(M) and isDeleted(D). Statusflags are the fields in the database. Initially all the flags are set to false.
- First it checks whether the record Id of handheld is there in TWN DB or not. If the record Id is same it means that we have to check the M and D bits. Otherwise the handheld records are checked for D bits. If the D bit is true then the record is not copied into the DB.
- The details of the algorithm and implementation are described in the later sections
- These two vectors are updated after comparing
- hhRecords vector is copied into the palm database
- pcRecords vector is copied into the TWN database

LocalDB class: This class is used to read the records from the TWN database. This has two main methods – one method to read the records and the other to write the records.

- Read method: This method just reads all the records and copies into a Java vector – pcRecords. This method is invoked in the AddrCond class.
- Write method: Updated pcRecords (newRecords) is obtained after synchronization (after comparing with hhRecords)
- This newRecords is different from the pcRecords vector. Now the TWN should reflect the changes. The newRecords vector is compared with the pcRecords vector and only the modified records are copied into another vector (newDBRecords)

- This new vector is then checked for isNew, isModified and isDeleted statusflags and then added or copied or deleted accordingly.

5.4 Implementation Details

The records in palm and TWN are compared using the status flags – isNew, isModified and isDeleted fields. These bits are checked on each side for synchronization. Say for example, if the record in the palm has isNew bit as 1 then it means that it is a new record and obviously there will not be a same record on the other side for the same record ID. So the palm bits will be 1, 0, 0 and the TWN bits will be Ø, Ø, Ø. Initially all the bits are Ø, Ø, Ø. So all the cases are considered here in the table and all the cases are analysed.

The following notation is used in the table.

N – isNew D – isDelete M – isModified

A – Add C – Copy/overwrite D – Delete

P – Palm T – TWN Action – 3types of actions – A, C, D according to the bits

True -1 False – 0 Null – Ø

No:	Palm				TWN				
1	N	D	M		N	D	M	Action	Where to where
2	0	0	0		0	0	0	Do nothing	-
3	1	0	0		0	0	0	Not possible	-
4	0	1	0		0	0	0	D	P
5	0	0	1		0	0	0	C	P-T
6	0	0	0		1	0	0	Not possible	-
7	0	0	0		0	1	0	D	T
8	0	0	0		0	0	1	C	T-P
9	1	1	0		0	0	0	Not possible	-
10	0	1	1		0	0	0	D	P
11	0	0	1		1	0	0	Not possible	
12	0	0	0		1	1	0	Not possible	-
13	0	0	0		0	1	1	D	T
14	1	0	1		0	0	0	Not possible	-
15	1	0	0		1	0	0	Not possible	
16	1	0	0		0	1	0	Not possible	
17	1	0	0		0	0	1	Not possible	
18	0	1	0		1	0	0	Not possible	
19	0	1	0		0	1	0	D	P,T
20	0	1	0		0	0	1	Check time(D)	P,T
21	0	0	1		1	0	0	Not possible	
22	0	0	1		0	1	0	Check time	P,T

23	0	0	1		0	0	1	C	P-T
24	0	0	0		1	1	0	Not possible	-
25	0	0	0		1	0	1	Not possible	-
26	0	0	0		0	1	1	D	P,T
27	1	0	1		1	0	0	Not possible	
28	1	0	0		1	1	0	Not possible	
29	1	0	0		0	1	1	Not possible	
30	1	0	0		1	0	1	Not possible	
31	1	0	1		0	0	1	Not possible	
32	0	1	1		1	0	0	Not possible	
33	0	1	0		1	1	0	Not possible	
34	0	1	0		0	1	1	D	P,T
35	0	1	1		0	1	0	D	P,T
36	1	1	0		1	0	0	Not possible	
37	1	1	0		0	1	0	Not possible	
38	1	1	0		0	0	1	Not possible	
39	0	0	1		1	1	0	Not possible	
40	0	0	1		0	1	1	Check time	
41	0	0	1		1	0	1	Not possible	
42	0	0	0		1	1	1	Not possible	-
43	0	1	0		1	0	1	Not possible	
44	1	1	1		1	0	0	Not possible	
45	1	0	1		1	1	0	Not possible	
46	1	0	0		1	1	1	Not possible	
47	1	0	1		0	1	1	Not possible	
48	1	1	0		1	1	0	Not possible	
49	1	1	0		0	1	1	Not possible	
50	1	1	0		1	0	1	Not possible	
51	1	1	1		0	0	1	Not possible	
52	1	1	1		0	1	0	Not possible	
53	0	1	1		1	1	0	Not possible	
54	0	1	1		0	1	1	D	
55	0	1	0		1	1	1	Not possible	
56	0	0	1		1	1	1	Not possible	
57	0	1	1		1	0	1	Not possible	
58	0	1	1		1	1	1	Not possible	
59	1	1	1		1	1	0	Not possible	
60	1	0	1		1	1	1	Not possible	
61	1	1	0		1	1	1	Not possible	
62	1	1	1		1	0	1	Not possible	

63	1	1	1		1	1	1	Not possible	
64	Ø	Ø	Ø		1	0	0	A	T-P
65	1	0	0		Ø	Ø	Ø	A	P-T
66	1	1	0		Ø	Ø	Ø	D	P
67	Ø	Ø	Ø		1	1	0	D	T
68	1	0	1		Ø	Ø	Ø	C	P-T
69	0	1	1		Ø	Ø	Ø	D	P
70	0	1	0		Ø	Ø	Ø	D	P
71	Ø	Ø	Ø		1	0	1	A	T-P
72	Ø	Ø	Ø		1	1	1	D	T
73	0	1	1		Ø	Ø	Ø	Not possible	-
74	0	1	0		Ø	Ø	Ø	Not possible	-
75	0	0	0		Ø	Ø	Ø	Not possible	
76	0	0	1		Ø	Ø	Ø	Not possible	
77	Ø	Ø	Ø		0	1	1	Not possible	-
78	Ø	Ø	Ø		0	1	0	Not possible	-
79	Ø	Ø	Ø		0	0	0	Not possible	
80	Ø	Ø	Ø		0	0	1	Not possible	

The above table shows all the 80 possible combinations of 0, 1 and Ø. But all these 80 cases are not valid ones. As explained earlier, if the bits on one side are 1, 0, 0 then the bits on other side should Ø, Ø, Ø. No other combinations are possible except for this. So, many cases become 'Not possible' cases as shown above. The table below shows only the possible cases and they come out to be 24.

Palm					TWN				
No	N	D	M		N	D	M	Action	Where to where
1	1	0	0		Ø	Ø	Ø	A	P-T
2	1	0	1		Ø	Ø	Ø	A	P-T
3	Ø	Ø	Ø		1	0	0	A	T-P
4	Ø	Ø	Ø		1	0	1	A	T-P
5	0	0	1		0	0	0	C	P-T
6	0	0	0		0	0	1	C	T-P
7	0	1	0		0	0	0	D	P-T
8	0	0	0		0	1	0	D	P,T
9	0	1	1		0	0	0	D	P,T
10	0	0	0		0	1	1	D	P,T

11	0	0	0		0	1	1	D	P,T
12	0	1	0		0	1	1	D	P,T
13	0	1	1		0	1	0	D	P,T
14	1	1	0		Ø	Ø	Ø	D	P
15	1	1	1		Ø	Ø	Ø	D	P
16	Ø	Ø	Ø		1	1	0	D	T
17	Ø	Ø	Ø		1	1	1	D	T
18	0	1	1		0	1	1	D	P,T
19	0	0	1		0	1	0	Check time	P,T
20	0	0	1		0	0	1	Check time	P-T
21	0	0	1		0	1	1	Check time	
22	0	1	1		0	0	1	Check time	
23	0	1	0		0	0	1	Check time	P,T
24	0	0	0		0	0	0	Do nothing	-

5.4.1 Synchronization process

There are 3 status flags in the address database of palm and TWN – isNew, isDeleted and isModified. These bits are very important in the synchronization process, because a decision is made only after checking the bits.

Since there are 3 bits on each side, all together they make 6bits. And each bit can be either true or false or null.

All the possible cases should be analyzed to get an optimistic algorithm. If only 1's and 0's are considered on both sides, then the total possible cases will be 2^6, which is equal to 64. All the cases are shown in the table1. There are also cases where the records are null (Ø).

I have come up with a notation to represent all the above cases in a general format. If null cases are taken into consideration in addition to the 64 cases, the total number of cases will be 64+16 = 80 cases. But all these 80 cases are not considered for synchronization.

Example:

If the handheld record bits are 1 0 0 and the TWN bits are 1 1 0. This is not possible, because the records are compared with respect to the record ID. And the record ID on palm is different from the record ID on the TWN. So a record which has 1 0 0 as its bits means that that is new record. So there should not be a record with the same ID on the other side. The bits on the other should be all null, because there is no record. So there are number of cases like this which are removed from the table1 because they don't happen.

The second table shows all the possible cases. So only these cases are considered for synchronization. There are 24 possible cases. Since the table looks a bit long it is reduced to a general notation.

Let x be 0 or 1.

All not possible cases are shown below

1	x	x	x	x	x	24 cases
x	x	x	1	x	x	24 cases
0	x	x	Ø	Ø	Ø	4 cases
Ø	Ø	Ø	0	x	x	4 cases
						56 cases

Total there are 80 cases where 56 are not possible cases. We will see the possible cases below. This is a generalization of table2. These are sorted according to the functionality. There are 5 possible things that can occur during synchronization.

- Add (inserting a new record)
- Delete (deleting a record)
- Copy (overwriting the already existing record)

- Do nothing (if there is no change in the bits on either side then do nothing. Example 0's on both sides like 0 0 0 and 0 0 0).
- Check time (if the record is modified on both sides then a decision should be made according to the time or priority should be given to either palm or TWN)

There are altogether 24 possible cases. The order is as follows

4 - Add
12 - Delete
2 - Copy
6 - Check time

Again x is 1 or 0. The above order can be written as

Add

1	0	x	Ø	Ø	Ø	2 cases
Ø	Ø	Ø	0	x	x	2 cases

Delete

0	1	x	0	x	0	4 cases

0	x	0	0	1	x	4 cases
1	1	x	Ø	Ø	Ø	2 cases
Ø	Ø	Ø	1	1	x	2 cases
Copy						
0	0	1	0	0	0	1 case
0	0	0	0	0	1	1 case
Checktime						
0	0	1	0	0	1	1 cases
0	0	1	0	1	x	2 cases
0	1	x	0	0	1	2 cases
Do nothing						
0	0	0	0	0	0	1 case
						24 cases

This algorithm is used for synchronization. All the 24 cases are taken into consideration.

5.4.2 Data Structure

There are records in the palm database and there are records in the TWN database. These two databases are compared for synchronizing. While programming, connecting to the TWN database is time consuming. And if the connection has to be made more times then more time will be wasted. Instead of connecting to the TWN database each time, all the records from TWN are copied into a vector data type. Similarly all the records from the palm are copied to a vector which is already explained earlier. Now these two vectors are compared and updated after synchronization. These vectors after synchronization are copied back to their respective databases.

So now the problem is 'what is the efficient way to copy the updated vector to TWN database'.

There are 2ways to do this. We will name the synchronized (updated) vector as 'newRecords' and the database records vector as 'pcRecords' (initial vector which is not updated).

1. All the pcRecords can be deleted and all the newRecords can be added to TWN. But in this case, all the saved TWN records are also deleted. Deleting in this case means not marking the record as deleted but completely removing the record from the database. So this is not the best way to do.

2. There is one more method to do this. This needs a small step wise procedure. The pcRecords and newRecords are compared and the action is taken. Say newRecords has a new record which is

not there in pcRecords. Then this new record is added to the TWN database. Similarly if a record is deleted in the newRecords then the TWN record should be deleted. For the modified records, if the record is marked as modified in newRecords then delete the record in the TWN and copy this new modified record. So this will be done in 2 steps - deleting the old record and copying the new record.

Second method is used in the implementation because it is more efficient one. Steps are given below.

- pcRecords and newRecords are compared
- only changed / modified records are copied into a new vector – newDBRecords
- Each newDBRecords is checked for isDeleted bit and if it is true then the records from the TWN database are deleted
- Each newDBRecords is checked for isModified bit and if it is true then the record from the TWN is deleted and this new modified record from the palm is inserted into the TWN
- Each newDBRecords is checked for isNew bit and if it is true then this is inserted into the TWN address table.

5.4.3 Assumptions

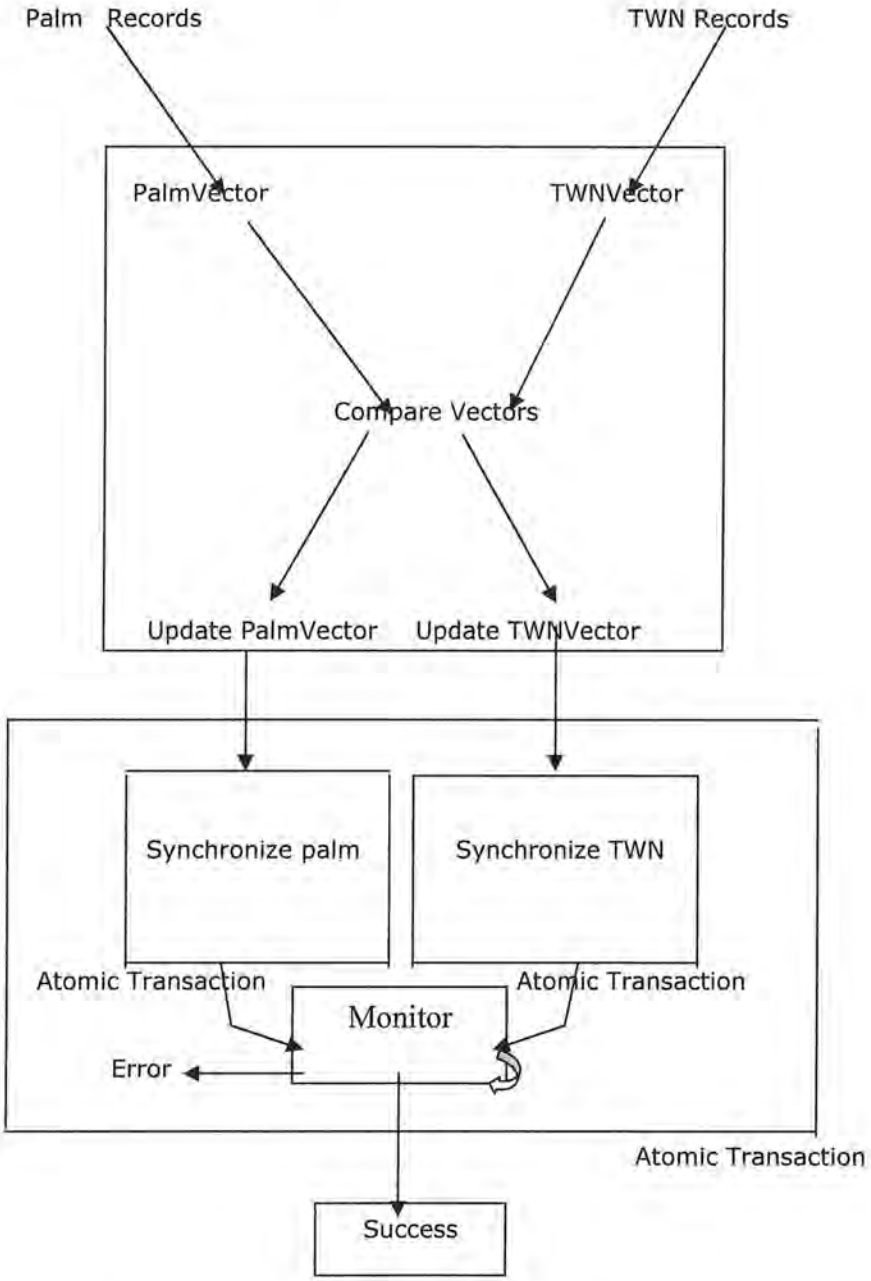
The main assumption made in the entire process is that the records are not lost and records are not accidentally deleted by administrator or hacker or faulty code.

Normally if a record is deleted, then it is marked as isDeleted = true in TWN. Only after the synchronization, the records are expunged from the database. But if someone deletes (expunges) the record in TWN accidentally then it will be a problem. After synchronization also the record will still exist in the palm database. And it is not deleted at all. This may cause memory flow after certain amount of time. In order to avoid this problem, slow synchronization is used.

For this project, we are assuming that these kinds of problems never occur.

5.4.4 Failure cases

Palm and TWN are updated in parallel.



All the palm address records are copied into a vector data structure. Similarly TWN records are copied into a different vector. These two vectors are compared and the status flags are checked. According to the status flags, isNew, isModified and isDelete, specified action is taken according to the table given above. So if the palm bits are 0,1,0 and if the TWN bits are 0,0,0, then TWN record (vector) is updated to 0,1,0. Similarly all the records are compared.

From the above discussion it is very clear that the synchronization process is not a single step process. In this section we will discuss more on the failure of the system

What happens if a system fails during synchronization?
The system can fail at three levels.

1. While copying the vectors from the respective DB's
2. While comparing vectors or while updating vectors
3. While synchronizing palm
4. While synchronizing TWN
5. Both fail

The most possible thing that can happen when the system fails is one DB is updated while other is not. We will see how the failure affects the two DB's.

A detailed explanation of the above failures and their consequences are described below.

1. While copying the vectors from the respective DB's
Then no problem, the two databases remain same after the failure also. But synchronization doesn't occur. So the user has to perform this process again.

2. While comparing vectors or while updating vectors
This type of failure also doesn't have much effect on the synchronization process. Both the DB's will be the same and the user should once again perform the synchronization.

3. While synchronizing palm
If a failure takes place while synchronizing the palm, then the palm system roll backs so synchronization is not completed. But if the TWN is synchronized then there will be a problem. Say for example, if there is a new record on TWN (1,0,0) and it has to be copied to palm(Ø,Ø,Ø), but during synchronization say if the palm synch failed. Since the palm synch failed the handheld is not updated. But on the TWN side the updated TWN record for 1,0,0 will become 0,0,0. So when the user again synchronizes, this new record is not copied to palm. If the TWN has a record with status flags 0,1,0 and if the process fails, then TWN updated record will be 0,0,0. But on the other side, the record on the palm will be still 0,0,0 and it is not deleted at all.

4. While synchronizing TWN
This is similar to the above case. Memory will be wasted if records are not deleted and they will be there for ever. New records will not be copied whenever there is an error. The data will not be consistent.

5. Both fail
If both fail then also the synchronization process is fully altered.

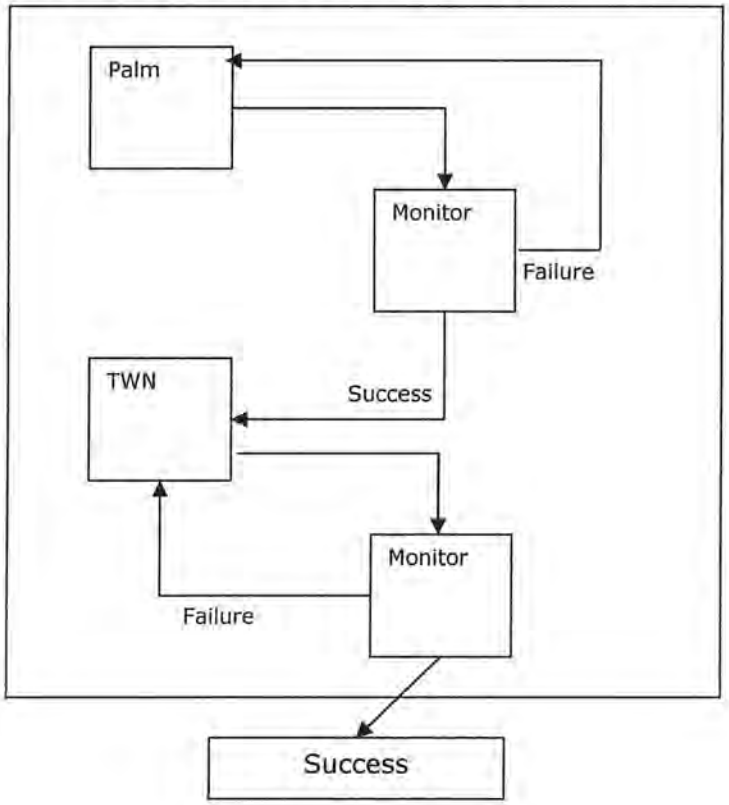
Steps to be taken:

1. Each transaction can be assumed as an atomic transaction. If there is any failure then the system should completely roll back on both sides. Because if one side rolls back and the other gets updated then it is again the same problem. So both the systems should roll back.
2. The user should be informed about the failure
3. Use a monitor to check if the transactions are successfully completed or not.

Palm synchronization is written such that it rolls back whenever it gets interrupted. But we have to make the TWN system roll back if there is any problem. The system should be locked from the user so that it is not disturbed while performing synchronization. And it becomes easy to roll back the system.

Palm and TWN are updated one after the other. Palm is synchronized first. TWN synchronized if and only if palm synchronization is successful. The diagram which is given below shows the sequential synchronization of palm and TWN. A monitor is provided to check the status of the synchronization. If palm synchronization fails then all the transactions are rolled back and the user is informed about the same.

If TWN fails, then synchronization should be performed only to the TWN records recursively as shown below. It comes out of the loop only if TWN is successfully completed.

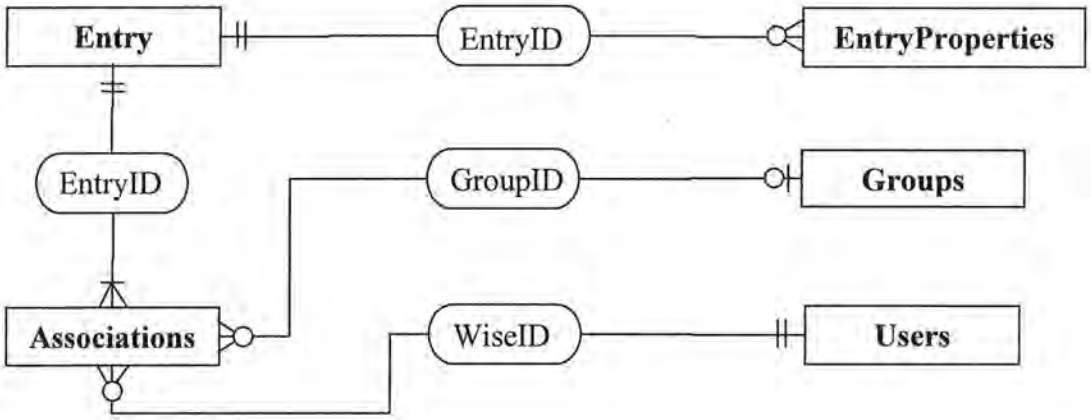


6.0 Palm TWN Calendar Synchronization (PTCS)

Calendar synchronization is similar to the Address synchronization. So the details of the synchronization algorithm are not discussed here. Only the table structure and variables are explained in this section.

Synchronization software is used to keep two or three different calendars in agreement. For example, if you use a Palm PDA, the synchronization software will compare appointments in your Palm with appointments in the central database, and keep them in agreement. It uses the most recent data from each. Thus if you add an appointment in your PDA, next time you sync that appointment will be added to the central database. Similarly, if someone puts an appointment on your calendar in the central database, next time you sync it will be loaded into your Palm's datebook.

6.1 Calendar database structure



This is the existing database structure at thewisenet. One more table is added to this database called as PalmAttributes, which is used for handheld operations.

There are all together four tables.

- Entry
- Associations
- EntryProperties
- PalmAttributes

Description of each table is given below.

1. *Table: Entry*

This table represents scheduled calendar entries across all TWN domain applications.

Field	Type	Description
*EntryID	Int	Every calendar entry has a unique id to make it distinct.
Description	Varchar	A brief text description of the entry.
TargetDate	Datetime	The scheduled beginning of an appointment or the due date of a task.
Duration	Int	This is the amount of time in minutes between the start and the end of the appointment. If the entry is a task, this is the expected duration of the task.
Category	Varchar	A semicolon delimited list of user defined category strings to classify the entry's context. This is defined by the organizer of the entry and is copied to the Category field of each association bound to the entry.
Status	Varchar	The current state of the entry could be: "Tentative", "Confirmed", "Cancelled", "Pending", and "Completed" and possibly more states will be identified as we develop. Any special state related flags that we need that are not mutually exclusive from other values of the Status field can be placed in the EntryProperties table.
Notes	Varchar	A more detailed text description of the entry.
CreationDate	Datetime	The date and time of when an entry was first created.
EntryGroup	Int	An id representing a group to which the entry belongs. Possibly an index to a GroupEntry table or, for now, a simple integer where 0 implies ungrouped.
DomainID	Int	Represents which Wise Net application this calendar entry belongs to. This doesn't really have anything to do with calendaring other than to separate calendar entries created in different TWN contexts.
Priority	Int	An integer between 0 and 9 that specifies the priority level of the entry. A value of 0 means the priority is undefined. 1 is highest priority and 9 is lowest.
Type	Varchar	The type of entry could be an 'Appointment', 'Task', or perhaps eventually some other entry type.
Location	Varchar	A location where the appointment is to take place.

Area	Varchar	An area where the appointment is to take place. RE specific.
Organizer	Varchar	The WiseID of the entry's creator.

Name	Type	Null
EntryID	int(11)	No
Description	varchar(64)	Yes
Duration	bigint(4)	Yes
Category	varchar(64)	Yes
Type	varchar(15)	Yes
Priority	int(11)	Yes
Notes	varchar(255)	Yes
EntryGroup	int(11)	Yes
DomainID	int(11)	Yes
TargetDate	datetime	Yes
CreationDate	datetime	Yes
Status	varchar(64)	Yes
Location	varchar(64)	Yes
Area	varchar(15)	Yes
Organizer	varchar(64)	Yes

2. *Table: EntryProperties*

This table represents a dynamic set of properties that can be unique to an individual calendar entry. It can be used, for example, to association 'Location' with an entry.

Field	Type	Description
*EntryID	Int	The id corresponding to the entry in the Entry table for which this property-value pair is assigned.
*Property	Varchar	The name of the property associated with the calendar entry.
Value	Varchar	The value of the property associated with the calendar entry.

Name	Type	Null
EntryID	int(11)	No
Property	varchar(64)	No
Value	varchar(255)	Yes

3. *Table: Associations*

This table represents user and group specific information that is relative to a particular calendar entry. When a user is associated with an entry, that user can see the entry in their calendar and

possibly modify it depending on their Role. When a group is associated with an Entry, the associations parameters apply to all the members of that group.

Field	Type	Description
*EntryID	Int	The id of the Entry that a user or group is to be associated with.
*WiseID	Int	This field contains a WiseID that represents a user to be associated with the entry.
*GroupID	Int	Specifies a group of users that are to be associated with the entry. Not to be confused with EntryGroup.
Role	Varchar	The role of the user or group relative to the entry. Possible role values are outlined below this table.
Status	Varchar	Represents the user or group's participation level relative to the entry. Possible status values are outlined below this table.
Category	Varchar	This field can be used to allow individual attendee's to sort their calendar entries according to their own categorization method. When an attendee is first associated with an entry, this category is directly copied from the original category field of the entry. Attendee's can then change their personal categorization of the entry if they want to. Multiple categories can be stored here, separated by semicolons.
View	Int	This boolean variable determines if the attendee is visible privately or publicly.
Delegator	Varchar	Used for task associations to reference the delegator of the association.
Watch	Int	A flag specifying that the user would like to set a watch on the entry so that it is displayed in their daily calendar view.

Name	Type	Null
EntryID	int(11)	No
WiseID	varchar(64)	No
GroupID	int(11)	No
Role	varchar(15)	Yes
Status	varchar(15)	Yes
Category	varchar(64)	Yes
View	varchar(7)	Yes
Delegator	varchar(64)	Yes
Watch	int(11)	Yes

4. *Table: PalmAttributes*

This table represents a dynamic set of properties that can be unique to an individual calendar entry. It can be used, for example, to association 'Location' with an entry.

Field	Type	Description
*EntryID	Int	The id corresponding to the entry in the Entry table for which this property-value pair is assigned.
PalmID	Int	The id corresponds to the palm record. This id is present in the palm Calendar database also.
isNew	Varchar	A Boolean variable which tells whether the record is new or not
isModified	Varchar	A Boolean variable which tells whether the record is modified or not after its creation
isDeleted	Varchar	A Boolean variable which tells whether the record is deleted or not
isPrivate	Varchar	A Boolean variable which tells whether the record is private or not
Organizer	Varchar	The WiseID and also the PalmID of the entry's creator.

Name	Type	Null
EntryID	int(11)	No
PalmID	int(3)	Yes
isNew	varchar(5)	Yes
isModified	varchar(5)	Yes
isDeleted	varchar(5)	Yes
isPrivate	varchar(5)	No
Organizer	varchar(65)	Yes

All the above four tables are dependant on each other. So modifying one record in one table is not so easy, because we may need to modify in other table also. If a new record is added in 'Entry' table then new record should be added to 'Associations', 'EntryProperties' and 'PalmAttributes'.

Palm pilot also has the calendar database and it is called as DateBook. This DB stores all the information related to the tasks of a user. This is similar to the Entry table on TWN server.

Palm DateBook Database Schema.

Name	Type	Null
ID	int(10) unsigned	Yes
CATEGORYINDEX	int(5) unsigned	Yes
STARTDATE	datetime	Yes
ENDDATE	datetime	Yes
ISUNTIMED	varchar(5)	Yes
ISALARMED	varchar(5)	Yes
ALARMADVANCETIME	int(5) unsigned	Yes
ALARMADVANCEUNIT	int(5) unsigned	Yes
ISREPEATING	varchar(5)	Yes
REPEATTYPE	int(5) unsigned	Yes
REPEATENDDATE	date	Yes
REPEATFREQUENCY	int(5) unsigned	Yes
REPEATON	varchar(10)	Yes
REPEATSTARTWEEK	int(5)	Yes
DESCRIPTION	varchar(22)	Yes
NOTE	varchar(255)	Yes
ISMODIFIED	varchar(5)	Yes
ISARCHIVED	varchar(5)	Yes
ISPRIVATE	varchar(5)	Yes
EXCEPTIONS	varchar(10)	Yes
ISDELETED	varchar(5)	Yes
ISNEW	varchar(5)	Yes
CURRENTTIME	timestamp(14)	Yes

There are number of variables in the above table. Many of the variables describe a recurrence event. If the event is recurrent then information like repeat day, time etc. are stored. For now we are not considering recurring events. So these variables are not seen in TWN database.

Variables considered for synchronization:
Id, CategoryIndex, Startdate, Enddate, Description, Note, isNew, isModified, isDeleted, and isPrivate. Enddate is actually not used. Instead duration is specified in the TWN database.

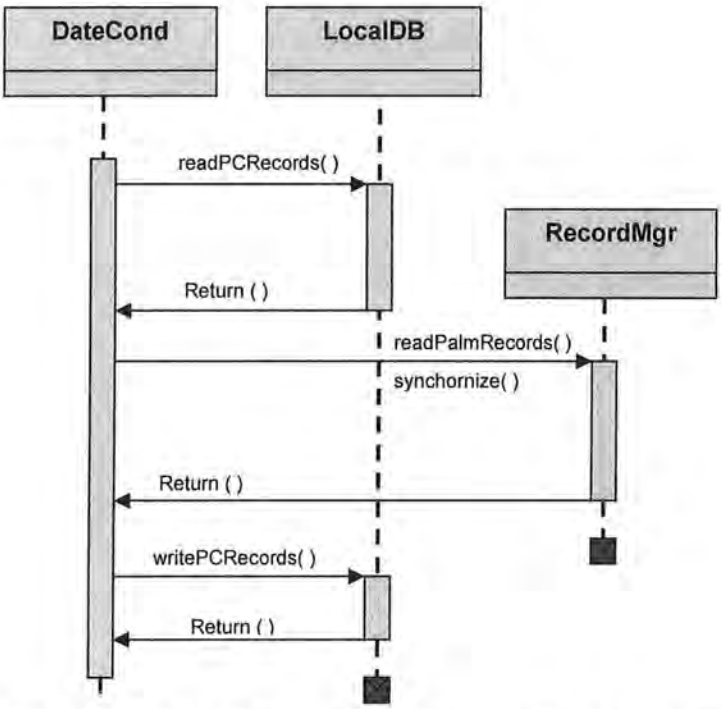
The comparison and synchronization process is already described for PTAS. The only difference is PTAS had only one table in TWN. Though there are four tables for calendar only 'Entry' and 'PalmAttributes' tables are used for most of the purposes.

Entry table is the main one, because it stores all the information related to a task. And PalmAttributes stores all the information related to a record in Entry. PalmAttributes table describes the status of the record, whether it is newly added, or deleted or modified. These status bits are important as we saw for PTAS.

6.2 Implementation Details

The datecond java conduit is written similar to the addresscond which is described above. Sequence diagram and class diagram for date conduit are given below.

6.2.1 Sequence Diagram



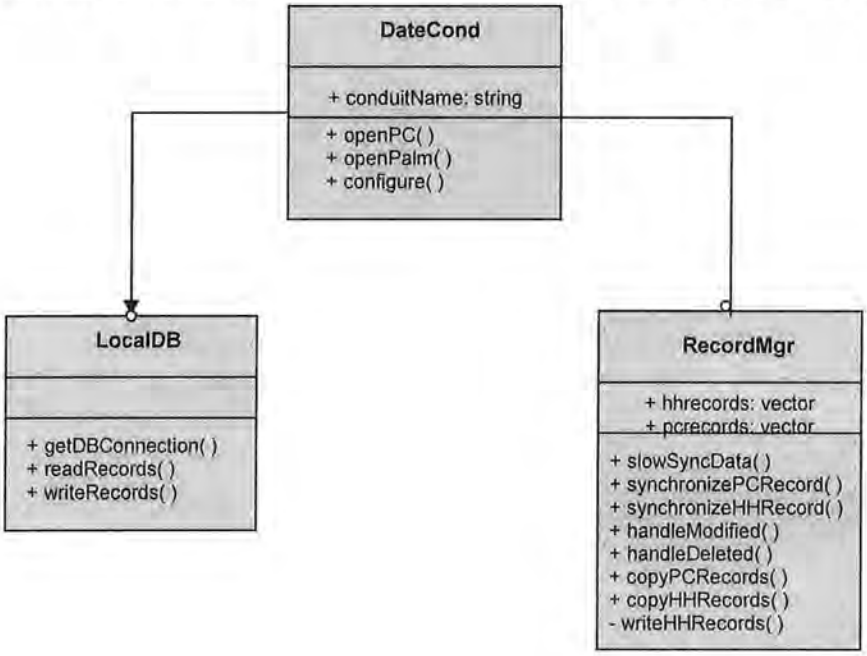
DateCond class is the active class and creates an instance of both LocalDb class and RecordMgr class.

Sequence of actions:

- Opens a connection with the local database
- Reads the records from the local database and writes into a vector
- Opens a connection with the palm database
- Reads the records from palm and writes into a vector
- Compares the two vectors using the status bits
- Synchronizes the two vectors
- Writes the records to palm and local database

6.2.2 Class Diagram

The class diagram gives a clear picture regarding the interaction between the objects and the methods involved. HH represents the handheld and pc represents local database.



6.3 Testing

Extensive testing is performed to make sure that there are no logical errors in the program. Testing also included boundary testing, stress testing, functional testing and performance testing.

6.4 Results

Synchronization is performed perfectly for address book and calendar. Code details can be available in Appendix B and java docs are attached in Appendix C.

6.5 Future Work

- To make the TWN calendar compatible with iCal specifications
- To make the web based calendar interoperable with Outlook and other standard calendaring systems
- Right now the clients need the java code to synchronize their calendars with palm. But in future, they should be supplied with a java executable which automatically does everything.
- The process can be made more user friendly by providing a button on the webpage to synchronize the web based calendar and palm datebook.
- Recurring events are not considered at this point. Implementation of recurring events and its synchronization will be a very good future work.
- Performance can be improved by making use of different data structures. The algorithm described above takes $O(n^2)$ execution time. This is can be improved by making use of Hash Tables.

APPENDIX A - Calendar Specification – iCal supported

All the variables in iCal are represented with capital letters.

A.1 Variables in iCal

BINARY - pre-defined data type

BOOLEAN - pre-defined data type

CAL-ADDRESS (CALENDAR USER ADDRESS) this value type is used to identify properties that contain a calendar user address. This is an URI.

DATE – this is used to identify values that contain a calendar date. (yyyymmdd). ISO 8601 Date string can be used.

DATE-TIME - this is used to identify values that contain a calendar date and also accurate time. The text format is a concatenation of the "date", followed by the CAPITAL LETTER T time designator, followed by the "time" format. ISO 8601 DateTime string can be used.

Date can be specified in three different ways

-Local time: this is simple local time; there is no conversion to the standard time

-UTC time: this is the standard time. Local time is converted to the standard time and the events are sent. This is identified by the time value appended by Z.

-Date with local time and time zone reference: Date along with the name of the time zone is mentioned.

DURATION - The format can represent durations in terms of weeks, days, hours, minutes, and seconds. ISO 8601 Duration string can be used.

FLOAT – pre-defined data type

INTEGER – pre-defined data type

PERIOD: This defines period of time. It can be start time/end time or start time/duration. ISO 8601 Period string can be used.

RECUR (RECURRENCE RULE): This defines lot of variables. This rule is a combination of different parts in any order. Same recurrence relation can be specified in different forms. We will see how a recurrence relation like – Meeting on every Tuesday for 2months. We will see what are the variables defined first.

FREQ –YEARLY/MONTHLY/WEEKLY/HOURLY/MINUTELY/SECONDLY

We can create a variable freq,

Eg: if freq = 1 YEARLY

= 2 MONTHLY

= 3 WEEKLY

= 4 HOURLY

= 5 MINUTELY

= 6 SECONDLY

INTERVAL – contains a positive integer representing how often the recurrence rule repeats

UNTIL – defines a date time value (until this time, which is inclusive)

COUNT – Number of occurrences of the event

BYDAY – MO/TU/WE/TH/FR/SA/SU. This can be preceded by a +ve or -ve accordingly. It represents the nth occurrence of the specific day within the MONTHLY and YEARLY RRULE.
BYMONTHDAY – value lies between -31 to -1 and 1 to 31. -10 represents the tenth to the last day of the month.
BYYEARDAY – value lies between -366 to -1 and 1 to 366.
BYWEEKNO – value lies between -53 to -1 and 1 to 53.
BYMONTH – value lies between -12 to -1 and 1 to 12.
WKST – This specifies the day on which the workweek starts. Valid values are MO/TU/WE/TH/FR/SA/SU
BYSETPOS – value lies between -366 to -1 and 1 to 366. If the RRULE contains BYMONTHLY and BYSETPOS = -1 then it represents the last day of the month

STRING: to specify a set of characters
TIME: This defines the time (hhmmss). This can also be represented in three forms like DATE-TIME. ISO 8601Time string can be used.
URI (Uniform Resource Identifier): This data type is defined by any IETF RFC.
UTC-OFFSET: This data type is used to identify properties that contain an offset from UTC to local time.

These are the variables which are used to describe some of the information related to other variables like attendee, organizer etc. PARTSTAT variable can be used to describe the status of the attendee, so it goes with ATTENDEE variable.
CN (Common Name) - String: To specify the common name to be associated with the calendar user.
CUTYPE (calendar User Type) - Integer: To specify the type of calendar user. CUTYPE can be Individual/Group/Resource/Room etc.
DELEGATED-FROM - email address: This variable is used when the delegator sends requests on behalf of the organizer.
DELEGATE-TO – email address: This variable us used when the organizer delegates his work to someone else.
FBTYPE - Integer: Valid values are FREE/BUSY/BUSY-UNAVAILABLE/BUSY-TENTATIVE
LANGUAGE - String: This specifies language for the text values
PARTSTAT-Integer: NEEDS-ACTION / ACCEPTED /DECLINED /TENTATIVE /DELEGATED / COMPLETED / IN-PROCESS
RANGE - Integer: To specify the effective range of recurrence instances from the instance specified by the recurrence identifier. Valid values are THISANDPRIOR / THISANDFUTURE
RELATED - integer: To specify the relationship of the alarm trigger with respect to the start or end of the calendar component. Valid values are START/END.
ROLE - Integer: To specify the participation role for the calendar user specified by the property.
CHAIR / REQ-PARTICIPANT/ OPT-PARTICIPANT/ NON-PARTICIPANT
RSVP - Boolean: To specify whether there is an expectation of a favor of a reply from the calendar user

SENT-BY – email address: To specify the calendar user that is acting on behalf of the calendar user

TZID (Time Zone Identifier): To specify the identifier for the time zone definition for a time component. Eg: US-Eastern.

Parameters:

METHOD - Integer: This property defines the iCalendar object method associated with the calendar object. If this property is not present in the iCalendar object, then a scheduling transaction MUST NOT be assumed. In such cases, the iCalendar object is merely being used to transport a snapshot of some calendar information; without the intention of conveying a scheduling semantic.

If *method* = 1 PUBLISH
= 2 REQUEST
= 3 REPLY
= 4 ADD
= 5 CANCEL
= 6 COUNTER
= 7 DECLINE-COUNTER

PRODID - String: This specifies the identifier for the product that created the iCalendar object.

VERSION - String: This specifies the identifier corresponding to the highest version number or the minimum and maximum range of the iCalendar specification that is required in order to interpret the iCalendar object.

ATTACH - semicolon-separated list of strings. The strings are URLs. This is for sending attachments.

CATEGORY - String: Defines the categories for a calendar component.

Eg: APPOINTMENT / EDUCATION / MEETING

CLASSIFICATION - String: Defines the access classification for a calendar component.

Eg: PUBLIC / PRIVATE / CONFIDENTIAL

GEO (Geographic Position) – Float: specifies information related to the global position for the activity specified by a calendar component.

PERCENT-COMPLETE - Integer: used by an assignee or delegatee of a to-do to convey the percent completion of a to-do to the Organizer.

RESOURCES - String: Defines the equipment or resources anticipated for an activity specified by a calendar entity.

Eg: LAPTOP, PROJECTOR, VCR

SUMMARY - String: Specifies a brief description for the activity.

COMMENT - String: Specifies a comment along with the event. This cane be a sentence or so.

DESCRIPTION - String: This gives a more description about the event than the SUMMARY variable.

LOCATION - String: Specifies the location of the event that is going to take place.

PRIORITY - Integer: Priority is given from 0 – 9. 0 = lowest priority and 9 = highest priority
STATUS - String: defines the overall status or confirmation for the calendar component.
Valid values are TENTATIVE / CONFIRMED / CANCELLED / NEEDS-ACTION / COMPLETED / IN-PROCESS / DRAFT / FINAL.
CLASS (Classification) - String: Defines the access classification for a calendar component.

DTSTAMP - Date-Time: Defines the date and time that a to-do was actually completed.
DTSTART - Date-Time: Specifies the date and time that a calendar component starts.
DTEND - Date-Time: Specifies the date and time that a calendar component ends.
COMPLETED - Date-Time: Defines the date and time that a to-do was actually completed.
DUE - Date-Time: Defines the date and time that a to-do is expected to be completed.
DURATION - Duration: Specifies a positive duration of time.
FREEBUSY - Period: Defines one or more free or busy time intervals. FBTYPE is also specified here. FREEBUSY gives the time and FBTYPE specifies whether the user is busy or not.
TRANSP (Time Transparency) - String: defines whether an event is transparent or not to busy time searches. Time Transparency is the characteristic of an event that determines whether it appears to consume time on a calendar. Events that consume actual time for the individual or resource associated with the calendar should be recorded as OPAQUE, allowing them to be detected by free-busy time searches. Other events which do not take up the individual's (or resource's) time should be recorded as TRANSPARENT, making them invisible to free-busy time searches.

ATTENDEE - CAL-ADDRESS: Defines the attendee. This can be an email address. Parameters include language, calendar user type, group or list membership, participation role, participation status, RSVP expectation, delegatee, delegator, sent by, common name etc.
CONTACT - String: This is used to represent contact information or alternately a reference to contact information associated with the calendar component.
ORGANIZER - Cal-Address: Defines the organizer. This can be an email address. Parameters include language, common name, sent by parameters can be specified on this.
RECURRENCE-ID - Date-Time: This property is used in conjunction with the "UID" and "SEQUENCE" variables to identify a specific instance of a recurring "VEVENT", "VTODO" or "VJOURNAL" calendar component. The ID value is the effective value of the "DTSTART" variable of the recurrence instance.
RELATED-TO - String: This is used to represent a relationship or reference between one calendar component and another.
URL – URI: defines a Uniform Resource Locator (URL) associated with the iCalendar object.
UID (Unique Identifier) - String: Defines the persistent, globally unique identifier for the calendar component. Each event has a unique identifier. The primary key for referencing a particular iCalendar component is the "UID" property value.

EXDATE (Exception Date) - Date-Time: defines the list of date/time exceptions for a recurring calendar component.

EXRULE (Exception Rule) - RECUR: This is a rule, so other parameters can be used to define a rule. This defines a rule or repeating pattern for an exception to a recurrence set. All rules related to RECUR could be used here.
RRULE - RECUR: Defines a rule or repeating pattern for recurring events, to-dos, or time zone definitions.

PRODID: This gives information related to the product.
VERSION: This gives the version of the calendar
DTSTAMP - Date-Time: Indicates the date/time that the instance of the iCalendar object was created.
LAST-MODIFIED - Date-Time: Specifies the date and time that the information associated with the calendar component was last revised in the calendar store.
SEQUENCE - Integer: This specifies the number of times the event changed after creating it. The latest changed event has a higher sequence number than the initially created instance.

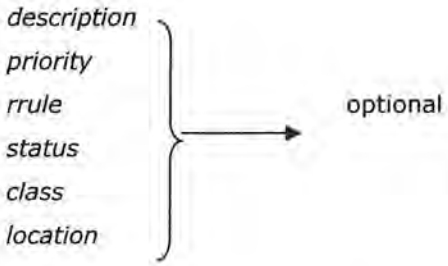
A.2 VEVENT: (This is for group scheduling)

Vevent component is basically used for scheduling purposes. It is very useful in small organizations. The Calendar User (CU) who initiates an exchange takes on the role of ORGANIZER according to the iCal spec. For example, the CU who proposes a group meeting is the ORGANIZER. He is responsible for posting the events and scheduling the meetings. No one other than ORGANIZER can post messages to the group. Only he can schedule meetings. The CUs asked to participate in the group meeting by the "Organizer" take on the role of "Attendee". Delegating is another important aspect of Vevent, which is mentioned in iTip.
Following are the methods related to the VEVENT component.

1. PUBLISH: Post notification of an event. Used primarily as a method of advertising the existence of an event. There is no interactivity between the publisher and any other calendar user. The events can be published in three ways – 1. Embedding the event as an object in a web page 2. Emailing an event to a distribution list 3. Posting an event to a newsgroup.
Variables involved in publishing an event.
prodid and *version* are common for all events.

Publishing an event

method = 1
prodid
version
organizer
UID
dtstamp
dtstart
dtend
summary
sequence



Updating a Published event

This is an extension of publishing event. Changes are made to the before event. Changes can be made to location, start time or end time etc. Sequence number is incremented but UID is the same and *method = 4.*

Canceling the Published event

method = 5
organizer
comment
description
dtstamp
uid
sequence

2. REQUEST: Whenever a request is made by the organizer to the attendees, then the attendee list also contains organizer as one of the attendee with the PARTSTAT value as ACCEPTED. This means that the organizer is making a request to himself with the status, ACCEPTED. This is true for all the users who send the request.

The "REQUEST" method in a "VEVENT" component provides the following scheduling functions:

2.1 Invite "Attendees" to an event: Only Organizer can send a request to the attendees. No other user except for the delegator can send a Request.

The Attendee parameter can also include other parameters, which give information about the Attendee. The organizer can mention these while sending the Request. Here is an example.

method = 2
organizer
attendee (email) [role, partstat, CN, RSVP, category]
dtstamp
dtstart
dtend
summary
uid
sequence

status
location

2.2 Reschedule an existing event: The "REQUEST" method may be used to reschedule an event. A rescheduled event involves a change to the existing event in terms of its time or recurrence intervals and possibly the location or description. If the recipient of a "REQUEST" method finds that the "UID" property value already exists on the calendar, but that the "SEQUENCE" (or "DTSTAMP") property value in the "REQUEST" method is greater than the value for the existing event, then the "REQUEST" method describes a rescheduling of the event.
method = 4
sequence – incremented
UID – unchanged
Other parameters may change compared to 2.1.

2.3 Update the details of an existing event, without rescheduling it: The "REQUEST" method may be used to update or reconfirm an event. An update to an existing event does not involve changes to the time or recurrence intervals, and might not involve a change to the location or description for the event. If the recipient CU of a "REQUEST" method finds that the "UID" property value already exists on the calendar and that the "SEQUENCE" value in the "REQUEST" is the same as the value for the existing event, then the "REQUEST" method describes an update of the event details, but no rescheduling of the event.
The update "REQUEST" method is the appropriate response to a "REFRESH" method sent from an "Attendee" to the "Organizer" of an event.
The "Organizer" of an event may also send unsolicited "REQUEST" methods. The unsolicited "REQUEST" methods may be used to update the details of the event without rescheduling it, to update the "partstat" parameter of "Attendees", or to reconfirm the event.

2.4 Forward a "VEVENT" to another uninvited CU: An "Attendee" invited to an event may invite another uninvited CU to the event. This kind of CU is referred as 'party crasher'. The invited "Attendee" accomplishes this by forwarding the original "REQUEST" method to the uninvited CU. But the decision whether the new Attendee is to be added or not depends on the Organizer. So updates of the event may not reach the 'party crasher'.

2.5 For an existing "VEVENT" calendar component, delegate the role of "Attendee" to another CU: If A is the organizer and scheduling a meeting with B, C and D. C wants to delegate the event to E, then C sends a REPLY to A and sends a REQUEST to E. C is called the delegator and E is called the delegatee.
C sends REPLY to A
method = 3
organizer

attendee (email id of C) [partstat = delegated; delegated-to(email id of E)]
uid
sequence
dtstamp
request-status

C sends REQUEST to E
method = 2
organizer
attendee (email id of C) [partstat = delegated; delegated-to (email id of E)]
attendee (email id of E) [RSVP = true; delegated-from (email id of C)]
other variables

Next step: E can accept or decline the REQUEST of C for delegating the event. E can send a REPLY to C by setting his *partstat* to ACCEPTED or DECLINED accordingly.

3.REPLY: This can be of any type. Attendee can send a REPLY to the organizer. When replying the Attendee sends the reply to himself. The attributes related to this method are given below.
method = 3
organizer (email id)
attendee (email id of the sender) [partstat]
uid
sequence
dtstamp
comment

4.ADD: This is mainly used when recurrence events occur. Sometimes the location/duration/time/ description have to be changed for recursive events. So the Organizer edits the event with proper information. This is discussed later.

5.CANCEL: The "CANCEL" method in a "VEVENT" calendar component is used to send a cancellation notice of an existing event request to the "Attendees". The message is sent by the Organizer" of the event. For a recurring event, either the whole event or instances of an event may be cancelled. To cancel the complete range of recurring event, the "UID" property value for the event MUST be specified and a "RECURRENCE-ID" MUST NOT be specified in the "CANCEL" method. In order to cancel an individual instance of the event, the "RECURRENCE-ID" property value for the event MUST be specified in the "CANCEL" method. *method = 4.*

6. REFRESH: The "REFRESH" method in a "VEVENT" calendar component is used by "Attendees" of an existing event to request an updated description from the event "Organizer". The "REFRESH" method must specify the "UID" property of the event to update. The "Organizer" responds with

the latest description and version of the event. This is also used when the user wants to confirm the location/time of the meeting.

7. COUNTER: The "COUNTER" method for a "VEVENT" calendar component is used by an "Attendee" of an existing event to submit to the "Organizer" a counter proposal to the event description. The "Attendee" sends this message to the "Organizer" of the event. This can be requesting a change of time/location etc. The attendee can include a comment saying what to change.

8. DECLINE-COUNTER: The "Organizer" can decline the counter proposal using this method. Comment can include saying that the change cannot be made.

9. Removing Attendees: Organizer can send a CANCEL message to the attendee to be removed. Then the organizer can send the updated event to others on the list except the one removed.

10. Replacing the organizer: Changed organizer can send the same event with incremented sequence number.

Recurrence Events: This also comes under VEVENT.
REQUEST for a recurrence event
method = 2
uid
sequence
rrule [freq = monthly; bymonthday = 1; until (time)]
organizer
attendee list
.
.
Recurence-id may be the dtstart of the event. dtstart changes for every instance of the event because dtstart is different for every event.

Update a recurrence event: To make some changes to the recurrence event, add a recurrence-id, which is the dtstart of the already existing event (to differentiate between different recurrence events). Mention the parameters to be changed.
recurrence-id
sequence – incremented by 1

CANCEL an instance of the recurrence event
method = 5
recurrence-id (dtstart of the instance of the event to be cancelled)
sequence - increment

Change all the future instances of the recurrence event. Give new dtstart and dtend or any other information related to the change.
method = 2
recurrence-id (range = thisandfuture)
sequence - increment
dtstart
dtend

Add a new instance to the recurring event
method = 4
sequence - incremented
give new dtstart and dtend and information related

Counter proposal for a recurrence event
method = 6
recurrence-id – of the instance
comment
new time or other parameters that has to be changed

Attendee can also give a REPLY if there is anything wrong in the RRULE sent by the organizer.

A.3 VFREEBUSY COMPONENT

Free busy times should be sorted in ascending order based on the start time and end time, with the earliest periods first. VFREEBUSY has three methods.

1. PUBLISH: Publish unsolicited busy time data.
The "PUBLISH" method in a "VFREEBUSY" calendar component is used to publish busy time data. The method may be sent from one CU to any other. The purpose of the method is to provide a message for sending unsolicited busy time data. That is, the busy time data is not being sent as a "REPLY" to the receipt of a "REQUEST" method. FREEBUSY and FBTYPE are mentioned here. These parameters are already discussed in the beginning.
method = 1
dtstart
dtend
freebusy (specify time) [fbtype]
.
.
dtstamp
uid

2. REQUEST: Request busy time data.

The "REQUEST" method in a "VFREEBUSY" calendar component is used to ask a CU for their busy time information. The request may be for busy time information bounded by a specific date and time interval. The request can be send to more than one CU at a time.

If the originator of the "REQUEST" method is not authorized to make a busy time request on the recipient's calendar system, then an exception message SHOULD be returned in a "REPLY" method, but no busy time data need to be returned.

The start and end times should be mentioned in the freebusy Request and also the attendee list.

3. REPLY: Reply to a busy time request.

The "REPLY" method in a "VFREEBUSY" calendar component is used to respond to a busy time request. The method is sent by the recipient of a busy time request to the originator of the request and also to himself.

A.4 VTODO Component

VTODO can be a group task. If some work is allotted to a group then they can discuss among themselves how much work is completed. Whoever sends the Request will be the organizer. A todo can be completion of a project specification, designing a component etc.

It represents an action item or assignment. It can also be used to represent an item of work assigned to an individual, such as turn in travel expense today. Example: Income Tax Preparation due tomorrow at 3pm.

If the method parameter is not there then it is not a scheduling todo.

REQUEST a todo

method = 2

organizer

attendee [rsvp = true]

dtstart

due

uid

sequence

summary

status - NEEDS-ACTION

priority

Other methods are similar to the VEVENT. Organizer can Request for the updated status of the attendees like how much percent of the work is completed. The attendees reply with the parameter specifying *percent-complete* and the *partstat* as *in-process* or *completed.*

Recurrence todos can also be there. All the methods for VEVENT can be applied here.

A.5 VJOURNAL COMPONENT

This component represents one or more descriptive text notes associated with a particular calendar date. It does not take up time on a calendar. So it does not play a role in free or busy time searches.

1. PUBLISH: Post a journal entry. Used primarily as a method of advertising the existence of a journal entry. This is similar to a task. No Attendees are involved here.

2. ADD: The "ADD" method in a "VJOURNAL" calendar component is used to add one or more instances to an existing "VJOURNAL" entry. If the "UID" property value in the "ADD" is not found on the recipient's calendar, then the recipient MAY treat the "ADD" as a "PUBLISH".

3. CANCEL: The "CANCEL" method in a "VJOURNAL" calendar component is used to send a cancellation notice of an existing journal entry. For a recurring journal entry, either the whole journal entry or instances of a journal entry may be cancelled. To cancel the complete range of a recurring journal entry, the "UID" property value for the journal entry MUST be specified and a "RECURRENCE-ID" property MUST NOT be specified in the "CANCEL" method. In order to cancel an individual instance of the journal entry, the "RECURRENCE-ID" property value for the journal entry MUST be specified in the "CANCEL" method.

A.6 VALARM COMPONENT: This is a remainder or alarm for an event or a todo. Attributed involved here are described below.

ACTION - String: Valid values are AUDIO / DISPLAY / EMAIL. This specifies what action has to take place when an alarm is triggered.

AUDIO should be an attachment (ATTACH), which points to the sound resource, which is rendered when the alarm is triggered.

DISPLAY can be a description of the event that has to take place. It may be very brief or elaborated it depends on the event.

EMAIL can be sent to the Attendee regarding the meeting or any other important todo. Email may contain an attachment or a brief description or summary.

REPEAT (Repeat Count): Defines the number of times the alarm should be repeated, after the initial trigger.

TRIGGER - Duration/ Date-Time: Specifies when an alarm will trigger. This may be just 5 minutes before the event or the time specified.

A.7 Use Cases

Here are the use cases for the above requirements. Use cases are UML based.

A.7.1 Request for a meeting

Actors

Organizer, Attendees

Pre Conditions

User is logged in

Post Condition

The appointment becomes associated with each recipient. Email is sent to all the Attendees selected.

Flow of events

Basic flow

1. The calendar user wants to schedule a meeting with other calendar users.
2. The user selects one or more CU's from the directory listing.
3. The CU enters the duration/time, location, description for the meeting
4. CU also mentions the participation status of the other CU's, whether it is compulsory to attend the meeting or not.
5. Then he clicks the invite button.

Secondary Scenario

Alternative paths

If the organizer doesn't find the required CU then he will make an extended search using the Wisenet search engine.

Subordinate Use Case

Search Engine

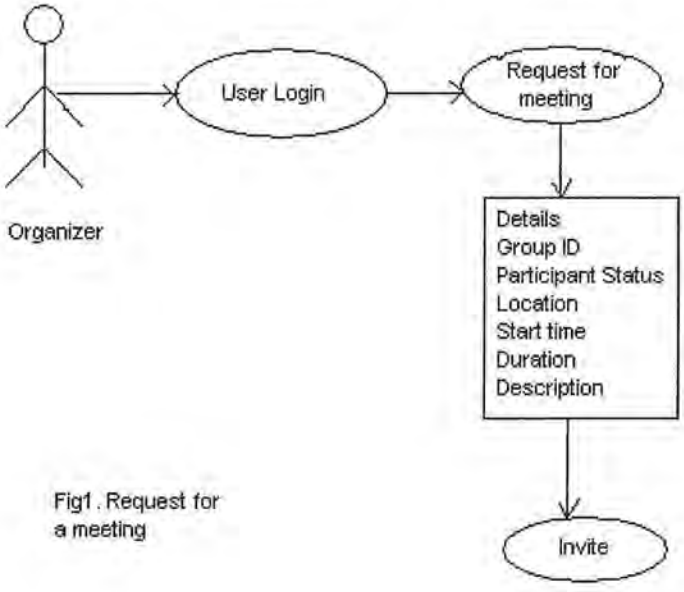


Fig1. Request for a meeting

The automated scheduling is not mentioned in the iCal specification. But we can build our own scheduling engine like Outlook and iPlanet. The Attendee through the email views the invitation and then he sends a reply "Accept/Decline". All the group members will have same id (not the calendar user name)

A.7.2 Respond to an Invitation

Actors

Organizer, Attendees

Pre Conditions

User is logged in

Post Condition

Accepted/Tentatively Accept/declined

Flow of events

Basic flow

1. The calendar user wants to accept/decline/tentatively accept a meeting with the organizer.
2. The status changes when he accepts or declines

Secondary Scenario

Alternative paths

An alternative path is possible after step2.

Subordinate Use Case

The user can send an email back to the organizer explaining the reason for declining

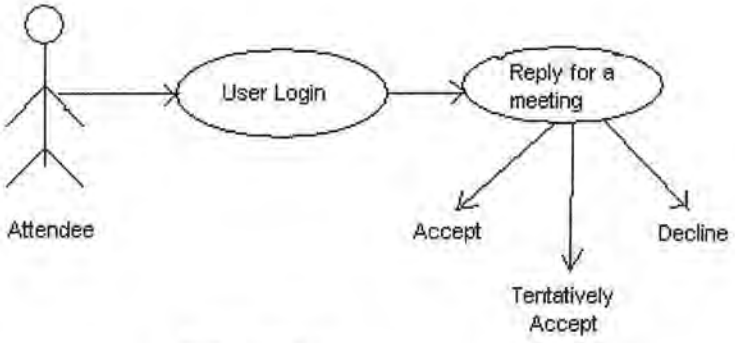


Fig2. Reply for a meeting

A.7.3 Publish a TODO

Actors

Calendar user

Pre Conditions

User is logged in

Post Condition

A Todo is posted on the user's calendar.

Flow of events

Basic flow

1. The calendar user wants to create a todo.
2. The CU enters the SUMMARY of the todo, due date of the todo or the duration
3. CU can also enter the PRIORITY (number), CLASS (Confidential, Public, Private) and CATEGORY (family, finance etc.) for the todo.
4. The STATUS of the todo is always "NEEDS-ACTION".

Secondary Scenario

Alternative paths

Subordinate Use Case

Cancel the todo.

Add one or more instances to an existing todo.

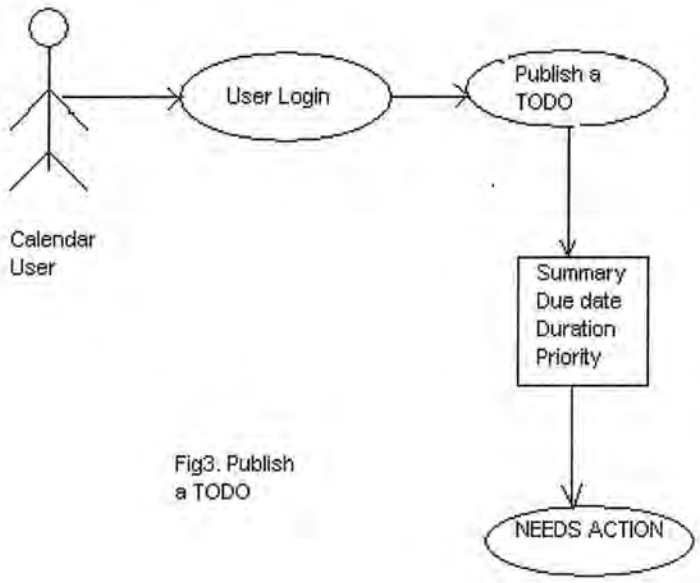


Fig3. Publish a TODO

References

1. http://www.imc.org/pdi/vcaloverview.html - VCalendar overview

2. http://docs.sun.com/db/doc/816-5520-10 - iPlanet Calendar Server 5.1 Programmer's Manual

3. http://systems.microlink.ee/failid/SCS_5.1.pdf - iPlanet Calendar Server paper

4. http://itadmin.sapp.auckland.ac.nz/APPFAeX/iPlanet/wp_calendar5.pdf

5. http://wwws.sun.com/software/products/calendar_srvr/wp_calendar5.html -
Web Based Scheduling, white paper by iPlanet

6. http://www.ietf.org/rfc/rfc2446.txt - iTIP protocol

7. http://www.ietf.org/rfc/rfc2447.txt - iCal protocol

8. http://wwws.sun.com/software/ - Sun One Software