

Data Synchronization for Wireless Applications

CHAPTER

12

IN THIS CHAPTER

- Overview 408
- SyncML 409
- Data Synchronization in J2ME MIDP 410
- Sample Implementation of SyncML: Mobile Scheduler 414

Overview

Mobile communication device users enjoy the convenience of accessing information anywhere at anytime. However, the information they want may not always be on the device they carry. Their cell phones need the phone numbers on their PDAs, and the really important e-mail is always on the corporate server. Users want ubiquitous access to information and applications from the device at hand, plus they want to access and update this information on-the-fly.

Mobile users may not want to be constantly connected to a network and data stored over the network, especially if they have to pay for the airtime by the minute. They want to retrieve data from the network and store it on the mobile device, where they can access and manipulate it locally. Periodically, they can reconnect with the network to send any local changes back to the networked data repository, and learn about updates made to the data over the network while the device was disconnected. Occasionally, they also need to resolve conflicts between the updates made to the two copies. This reconciliation operation—where updates are exchanged and conflicts are resolved—is known as *data synchronization*.

Data stored on many different types of devices must be able to be synchronized, as shown in Figure 12.1. A proliferation of different proprietary data synchronization protocols exists for mobile devices. But they are incompatible with each other. Users using one kind of device may not be able to synchronize data with other kinds of devices, which causes a lot of inconvenience for the users.

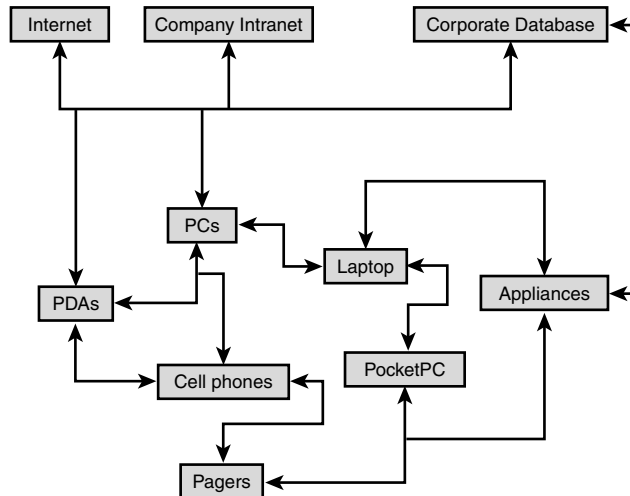


FIGURE 12.1

A universe of data to be synchronized.

To achieve universal synchronization, the following two requirements must be fulfilled:

- Synchronization servers must support synchronization with any mobile devices
- Mobile devices must be able to synchronize with any networked data

This chapter discusses a new open standard for universal data synchronization: SyncML. It then looks at data synchronization with J2ME MIDP. At the end of the chapter, you will see how to implement SyncML in J2ME MIDP applications.

SyncML

Over the last year, IBM, Lotus, Motorola, Nokia, Palm, Psion, and Starfish Software created a consortium to develop an open standard protocol for data synchronization: Synchronization Markup Language (SyncML). Since the release of the SyncML 1.0 specification at the end of last year, more than 600 companies have signed on to support SyncML.

SyncML is an XML-based standard. With SyncML, networked information can be synchronized with any mobile device, and mobile information can be synchronized with any networked applications. Using SyncML, any personal information, such as e-mail, calendars, to-do lists, and contact information will be consistent, accessible, and up to date, no matter where the information is stored. The data sources of synchronization include the sync server, corporate databases, and data stored on consumer service, PC, and mobile devices.

SyncML messages are represented in XML format. The MIME type, which is the industry standard for identifying different content types sent over the Internet, is registered for SyncML messages. The cleartext XML representation for SyncML messages is identified with MIME type `application/vnd.syncml+xml`. The WBXML binary representation for SyncML messages is identified with MIME type `application/vnd.syncml-wbxml`.

SyncML Protocols

SyncML consists of two parts: the SyncML representation protocol and the SyncML sync protocol. They both are based on the SyncML syntax. The sample program in this chapter primarily will demonstrate the features of the representation protocol.

The SyncML representation protocol focuses on organizing the data contents of the synchronization. It defines methods for uniquely naming and identifying records. It also defines the XML document type used to represent a SyncML message, such as common protocol commands and message containers. Synchronization command elements include `Add`, `Alert`, `Atomic`, `Copy`, `Delete`, `Exec`, `Get`, `Map`, `Replace`, `Search`, `Sequence`, and `Sync`.

The sync protocol focuses on managing the session operations of the synchronization. It defines the message flow between a SyncML client and server during a data synchronization

session. The types of synchronization include one-way sync from the client only, one-way sync from the server only, two-way sync, and server alerted sync. This protocol also defines how to challenge authentication, how to initiate a synchronization session, and how to resolve conflicts.

SyncML messages can be transported over a wireless network using HTTP; Wireless Session Protocol (WSP), which is part of the WAP protocol suite; or Object Exchange protocol (OBEX) for local connectivity.

The SyncML protocol holds the promise of universal synchronization—and its commerce implications in the future could be even more revolutionary for mobile computing. You can find more information about SyncML at <http://www.syncml.org>.

Data Synchronization in J2ME MIDP

J2ME opens a door for Java developers to develop applications using one language that can run on all types of devices. This was never possible before Java.

We not only want our applications to run on as many devices as possible, we also want our data to be interoperable with as many devices as possible. It is very costly to have your application support many different synchronization technologies and protocols. It also increases the complexity of the resulting product. The added complexity of the networked data repository can create a barrier to installation and adoption by service providers. SyncML makes possible unified data synchronization among a more diverse set of devices and networked data.

HTTP Network Transport Protocol

Because all MIDP devices are required to support the HTTP protocol, it is a natural candidate to be the network transport protocol used in data synchronization between MIDP devices and sync servers.

Once an HTTP connection is established, one or more SyncML messages can be sent to the server by the SyncML client in the body of HTTP requests or received from the server in the body of HTTP responses.

The POST method is used to transfer the SyncML message in an HTTP request. The following information needs to be specified in a HTTP header:

- Cache control—Used to control the caching mechanisms in the request/response chain between the HTTP client and the HTTP server.
- Accepted date type—Used to specify which MIME types are acceptable in the response message.

- Accepted character set—Used to specify which character sets are acceptable in the response message.
- Transfer-encoding—Used to indicate what type of transformation has been applied to the message body.
- Authorization information—Used by an HTTP client to authenticate itself to the HTTP server.
- User agent—Used to identify the type of user agent originating the request.

The following is a sample HTTP header used in SyncML data synchronization:

```
POST ./servlet/syncit HTTP/1.1
Host: www.datasync.org
Accept: application/vnd.syncml+xml
Accept-Charset: utf-8
Accept-Encoding: chunked
Authorization: Basic QWxhZGRpbjpwcmVudIHNlc2FtZQ==
Content-Type: application/vnd.syncml+xml; charset="utf-8"
User-Agent: MIDP sync product
Content-Length: 1023
Cache-Control: no-store
Transfer-Encoding: chunked
```

Synchronizing a Calendar

The data synchronization protocol synchronizes networked data with many different devices, including handheld computers, mobile phones, automotive computers, and desktop PCs. Many types of data need to be synchronized including e-mail, calendar, contacts, bank accounts, company files and documents, product information, customer information, product prices, and so on. For example, a user could read his calendar from either a handheld or a mobile phone, and still maintain a consistent, updated record of which messages had been read. SyncML can be used to synchronize calendar information between MIDP devices and an Internet sync server. The calendar format used in SyncML is based on vCalendar version 1.0.

NOTE

vCalendar is an exchange format for personal scheduling information. It is an open specification based on industry standards such as the x/Open and XAPIA Calendaring and Scheduling API (CSA), the ISO 8601 international date and time standard, and the related MIME e-mail standards. It is applicable to a wide variety of calendaring and scheduling products and is useful in exchanging information across a broad range of transport methods. vCalendar is receiving wide industry adoption. You can find more information on vCalendar at <http://www.imc.org/pdi/>.

The following is an appointment example in vCalendar format:

```
BEGIN:VCALENDAR
VERSION:1.0
BEGIN:VEVENT
DTSTART:20000509T063000Z
DTEND:20000509T073000Z
SUMMARY:SyncML Briefing
DESCRIPTION;ENCODING=QUOTED-PRINTABLE:John Smith is =
  the presenter.=0D=0ASyncML is the topic.
CLASS:PUBLIC
CATEGORIES:APPOINTMENT
AALARM:20000509T061500Z
END:VEVENT
END:VCALENDAR
```

This vCalendar specifies that an appointment begins at 6:30 a.m. on May 9, 2000 and ends at 7:30 a.m. The subject of the appointment is “SyncML briefing.” However, the calendar data format used by wireless applications or synchronization servers doesn’t have to be the vCalendar format. If a data field of vCalendar is not supported in the local representation, the field will be ignored. When a SyncML message is generated, the fields that are only supported in the local format will not be included in the vCalendar object. The following is a SyncML message sent from a sync server to a sync client:

```
<SyncML>
  <SyncHdr>
    <VerDTD>1.0</VerDTD>
    <VerProto>SyncML/1.0</VerProto>
    <SessionID>1</SessionID>
    <MsgID>2</MsgID>
    <Target><LocURI>my_phone</LocURI></Target>
    <Source>
      <LocURI>http://www.webyu.com/servlets/samsbook</LocURI>
    </Source>
  </SyncHdr>
  <SyncBody>
    <Sync>
      <CmdID>1</CmdID>
      <Target><LocURI>CalendarDB</LocURI></Target>
      <Source><LocURI>samsbook.nsf</LocURI></Source>
      <!--Add a new record to the CalendarDB, record ID 2021 -->
      <Add>
        <CmdID>4</CmdID>
        <Meta><mi:Type>text/x-vCalendar</mi:Type></Meta>
        <Item>
          <Source><LocURI>2021</LocURI></Source>
```

```

<Data><!--The vCalendar data would be placed here.-->
  BEGIN:VCALENDAR
  VERSION:1.0
  BEGIN:VEVENT
  DTSTART:20010510T063000Z
  DTEND:20010510T073000Z
  SUMMARY:SyncML Test Checkpoint DB002021
  DESCRIPTION;ENCODING=QUOTED-PRINTABLE:John =
    Smith is still the presenter.=0D=0ASyncML =
    is the topic.
  CLASS:PUBLIC
  CATEGORIES:APPOINTMENT
  AALARM:20010510T061500Z
  END:VEVENT
  END:VCALENDAR
</Data>
</Item>
</Add>
<!--Delete a record, record ID 2022 -->
<Delete>
  <CmdID>5</CmdID>
  <Meta><mi:Type>text/x-vCalendar</mi:Type></Meta>
  <Item>
    <Source><LocURI>2022</LocURI></Source>
  </Item>
</Delete>
</Sync>
</SyncBody>
</SyncML>

```

This SyncML message contains two components: the SyncML header (SyncHdr) and the SyncML body (SyncBody). The SyncML header specifies routing and versioning information about the SyncML message. The SyncML body is a container for one or more SyncML Commands. The SyncHdr identifies the revisioning, the source, and the target of the data contents. The source is an Internet data repository at <http://www.webyu.com/servlets/samsbook>. The target is a cell phone labeled “my phone.” The SyncBody specifies the contents of the synchronization including the sync commands and data items. In this example, the target of this SyncBody is CalendarDB and the source is samsbook.nsf. The SyncBody contains two sync operations: Add and Delete.

Within the Add operation, a new record with source record id 2021 will be added to the target calendar database CalendarDB on the cell phone. In this example, only one record is added. The actual appointment data is represented in vCalendar format enclosed in a Data tag. The Delete operation is simple in this case. One record with target record id 2022 needs to be deleted.

Sample Implementation of SyncML: Mobile Scheduler

This section looks at how to extend the MobileScheduler application you worked on in Chapter 8, “Persistent Storage,” to perform data synchronization with a test sync server. In this example, you will use SyncML as the data exchange protocol. The network protocol used in the example is the HTTP protocol. The synchronization involves two sync agents: one running on a MIDP device and one running on the sync server as depicted in Figure 12.2.

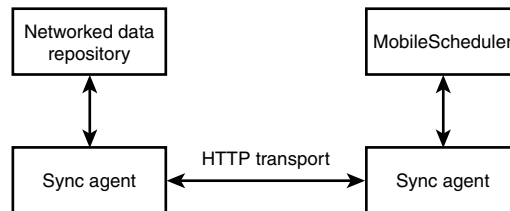


FIGURE 12.2

The Mobile Scheduler data synchronization system.

Sync Agent

Fully implementing SyncML is too lengthy and is not the intention of this chapter. The sync agents shown in this chapter implement only a subset of SyncML’s functionality. The sync agents are almost identical at the client and server. Both client and server interact with the sync agent to make synchronization modifications.

In this example, you will only implement synchronization commands: Add, Delete, Sync, and Update. No conflicts or synchronization results will be checked. Authentication is not checked either.

Listing 12.1 shows the subset of SyncML markup language definitions used in this example.

LISTING 12.1 syncml_subset.dtd

```

<!--Copyright NoticeCopyright Notice
Copyright (c) Ericsson, IBM, Lotus, Matsushita Communication IndustrialCo.,
LTD, Motorola, Nokia, Palm, Inc., Psion, Starfish Software (2000).
All Rights Reserved.
  
```

Implementation of all or part of any Specification may require licenses under third party intellectual property rights, including without limitation, patent rights (such a third party may or may not be a Supporter). The Sponsors of the

LISTING 12.1 Continued

Specification are not responsible and shall not be held responsible in any manner for identifying or failing to identify any or all such third party intellectual property rights.

THIS DOCUMENT AND THE INFORMATION CONTAINED HEREIN ARE PROVIDED ON AN "AS IS" BASIS WITHOUT WARRANTY OF ANY KIND AND ERICSSON, IBM, LOTUS, MATSUSHITA COMMUNICATION INDUSTRIAL CO. LTD, MOTOROLA, NOKIA, PALM INC., PSION, STARFISH SOFTWARE AND ALL OTHER SYNCML SPONSORS DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL ERICSSON, IBM, LOTUS, MATSUSHITA COMMUNICATION INDUSTRIAL CO., LTD, MOTOROLA, NOKIA, PALM INC., PSION, STARFISH SOFTWARE OR ANY OTHER SYNCML SPONSOR BE LIABLE TO ANY PARTY FOR ANY LOSS OF PROFITS, LOSS OF BUSINESS, LOSS OF USE OF DATA, INTERRUPTION OF BUSINESS, OR FOR DIRECT, INDIRECT, SPECIAL OR EXEMPLARY, INCIDENTAL, PUNITIVE OR CONSEQUENTIAL DAMAGES OF ANY KIND IN CONNECTION WITH THIS DOCUMENT OR THE INFORMATION CONTAINED HEREIN, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH LOSS OR DAMAGE.

The above notice and this paragraph must be included on all copies of this document that are made. -->

<!-- This DTD defines the SyncML DTD. The document type defines a common format for representing data sychronization protocol data units.

This DTD is to be identified by the URI string "syncml:syncml".

Single element types from this name space can be referenced as follows:

```
<element xmlns='syncml:syncml'>blah, blah</element>
-->
<!-- Root or Document Element and -->
<!ELEMENT SyncML (SyncHdr, SyncBody)>
<!ELEMENT SyncHdr (VerDTD, VerProto, SessionID, MsgID, Target, Source)>
<!ELEMENT SyncBody (Sync)+>
<!-- Value must be one of "Add" | "Delete" | "Update". -->
<!ELEMENT Cmd (#PCDATA)>
<!-- Sync message unique identifier for command -->
<!ELEMENT CmdID (#PCDATA)>
<!-- Reference to command identifier -->
<!ELEMENT CmdRef (#PCDATA)>
<!-- Location displayable name -->
<!ELEMENT LocName (#PCDATA)>
<!-- Location URI -->
<!ELEMENT LocURI (#PCDATA)>
<!-- SyncML Message ID -->
<!ELEMENT MsgID (#PCDATA)>
<!-- Reference to a SyncML Message ID -->
```

LISTING 12.1 Continued

```

<!ELEMENT MsgRef (#PCDATA)>
<!-- SyncML session identifier -->
<!ELEMENT SessionID (#PCDATA)>
<!-- Source location -->
<!ELEMENT Source (LocURI, LocName?)>
<!ELEMENT SourceRef (#PCDATA)>
<!-- Target location information -->
<!ELEMENT Target (LocURI, LocName?)>
<!ELEMENT TargetRef (#PCDATA)>
<!-- SyncML specificaiton major/minor version info. -->
<!-- For this version of the DTD, the value is "1.0" -->
<!ELEMENT VerDTD (#PCDATA)>
<!-- Data sync protocol major/minor version -->
<!-- For example, "xyz/1.0" -->
<!ELEMENT VerProto (#PCDATA)>
<!-- Synchronization data elements -->
<!-- Item element type -->
<!ELEMENT Item (Target?, Source?, Data?)>
<!-- Actual data content -->
<!ELEMENT Data (#PCDATA)>
<!-- SyncML Commands -->
<!-- Add operation. -->
<!ELEMENT Add (CmdID, Item+)>
<!-- Delete operation. -->
<!ELEMENT Delete (CmdID, Item+)>
<!-- Update operation. -->
<!ELEMENT Update (CmdID, Item+)>
<!-- Synchronize Operation. -->
<!ELEMENT Sync (CmdID, Target?, Source?, (Add | Delete | Update)*)>
<!-- End of DTD Definition -->

```

You can download the complete DTD file for SyncML's representation protocol at http://www.syncml.org/docs/syncml_represent_v10_20001207.dtd.

Data Representation

The data format for storing calendar information in MobileScheduler is different from vCalendar format. We refer to the data format for storing data on MIPD devices as *internal data format*, and we refer to the vCalendar format as *external data format*. By separating the two formats, the application will have a better chance to incorporate any future changes made to them. Listing 12.2 is the class for representing the external data format.

LISTING 12.2 CalendarItem.java

```
import java.util.*;
import java.io.*;

/* for simplicity we assume we only deal with one server.
 * Thus, we do not need to store server_id.
 */
public class CalendarItem {
    Location target; //local id
    Location source; //remote id
    String start;
    String end;
    String summary;
    String description;
    String categories;
    String c_class;
    String alarm;

    // Set the default values
    public CalendarItem() {
        target=null;
        source=null;
        start = null;
        end = null;
        summary = null;
        description = null;
        categories = null;
        c_class = null;
        alarm = null;
    }

    public CalendarItem(Appointment app, SynchOption so) {
        this();
        source=new Location("calendarDB",
                           String.valueOf(app.getId()));
        target=new Location(so.getUrl()+":"+so.getUser(),
                           app.getRemoteId());
        start= toTimeString(app.getTime());
        end=toTimeString(app.getTime()+app.getLength()*60000);
        summary= app.getSubject();
    }

    private long fromTimeString(String str) {
        Calendar cal= Calendar.getInstance();
```

LISTING 12.2 Continued

```
//year
cal.set(cal.YEAR, Integer.parseInt(str.substring(0,4)));
//month
cal.set(cal.MONTH, Integer.parseInt(str.substring(4,6))-1);
//day
cal.set(cal.DAY_OF_MONTH,
        Integer.parseInt(str.substring(6,8)));
//hour
cal.set(cal.HOUR_OF_DAY,
        Integer.parseInt(str.substring(9,11)));
//minute
cal.set(cal.MINUTE, Integer.parseInt(str.substring(11,13)));
//second and millisecond
cal.set(cal.SECOND, 0);
cal.set(cal.MILLISECOND, 0);

return cal.getTime().getTime();
}

private String toTimeString(long t) {
    StringBuffer sb= new StringBuffer();
    Calendar cal= Calendar.getInstance();
    cal.setTime(new Date(t));
    //year
    sb.append(cal.get(cal.YEAR));
    //month
    int m=cal.get(cal.MONTH)+1;
    if(m<10) sb.append(0);
    sb.append(m);
    //day
    int d=cal.get(cal.DAY_OF_MONTH);
    if(d<10) sb.append(0);
    sb.append(d);
    //mark it as UTC
    sb.append('T');
    //hour
    int h=cal.get(cal.HOUR_OF_DAY);
    if(h<10) sb.append(0);
    sb.append(h);
    //minute
    int min=cal.get(cal.MINUTE);
    if(min<10) sb.append(0);
    sb.append(min);
    //add second
```

LISTING 12.2 Continued

```
        sb.append("00Z");

        return sb.toString();
    }

    String getAlarm() {
        return alarm;
    }
    String getCategories() {
        return categories;
    }
    String getC_Class() {
        return c_class;
    }
    String getDescription() {
        return description;
    }
    String getEnd() {
        return end;
    }
    long getEndLong() {
        return fromTimeString(end);
    }
    String getStart() {
        return start;
    }
    long getStartLong() {
        return fromTimeString(start);
    }
    String getSummary() {
        return summary;
    }
    Location getTarget() {
        return target;
    }
    Location getSource() {
        return source;
    }
    void setAlarm(String _alarm) {
        alarm = _alarm;
    }
    void setCategories(String _categories) {
        categories = _categories;
    }
}
```

12

LISTING 12.2 Continued

```
void setC_Class(String _cclass) {
    c_class = _cclass;
}
void setDescription(String _description) {
    description = _description;
}
void setEnd(String _end) {
    end = _end;
}
void setStart(String _start) {
    start = _start;
}
void setSummary(String _summary) {
    summary = _summary;
}
void setTarget(Location _target) {
    target = _target;
}
void setSource(Location _source) {
    source=_source;
}

//set data from vCalendar data
void setData(String _data) {
    int startIndex = 0;
    int endIndex = 0;
    int len = _data.length();
    while ((endIndex = data.indexOf('\n', startIndex)) != -1) {
        String aline = _data.substring(startIndex,
                                       endIndex).trim();
        if (aline.startsWith("DTSTART"))
            start = aline.substring(aline.indexOf(":")+1);
        else if (aline.startsWith("DTEND"))
            end = aline.substring(aline.indexOf(":")+1);
        else if (aline.startsWith("SUMMARY"))
            summary = aline.substring(aline.indexOf(":")+1);
        else if (aline.startsWith("DESCRIPTION"))
            description = aline.substring(aline.indexOf(":")+1);
        else if (aline.startsWith("CLASS"))
            c_class = aline.substring(aline.indexOf(":")+1);
        else if (aline.startsWith("CATEGORIES"))
            categories = aline.substring(aline.indexOf(":")+1);
        else if (aline.startsWith("AALARM"))
            alarm = aline.substring(aline.indexOf(":")+1);
    }
}
```

LISTING 12.2 Continued

```

        startIndex = endIndex + 1;
    }
}

//output the item to SynchML string
public String toSyncML() {
    StringBuffer sb = new StringBuffer("<Item>");
    sb.append("<Source><LocURI>").append(source.getLocUri())
        .append("</LocURI></Source>");
    sb.append("<Data>");
    sb.append("BEGIN:VCALENDAR\n");
    sb.append("VERSION:1.0\n");
    sb.append("BEGIN:VEVENT\n");
    sb.append("DTSTART:").append(start).append("\n");
    sb.append("DTEND:").append(end).append("\n");
    sb.append("SUMMARY:").append(summary).append("\n");

    if(description!=null) {
        sb.append("DESCRIPTION;ENCODING=QUOTED-PRITABLE:")
            .append(description).append("\n");
    }
    if(c_class!=null) {
        sb.append("CLASS:").append(c_class).append("\n");
    }
    if(c_class!=null) {
        sb.append("CLASS:").append(c_class).append("\n");
    }
    if(categories!=null) {
        sb.append("CATEGORIES:").append(categories)
            .append("\n");
    }
    if(alarm!=null) {
        sb.append("ALARM:").append(alarm).append("\n");
    }
    sb.append("END:VEVENT\n");
    sb.append("END:VCALENDAR\n");
    sb.append("</Data>");
    sb.append("</Item>");

    return sb.toString();
}
}

```

12**DATA
SYNCHRONIZATION
FOR WIRELESS**

The Location class used in CalendarItem is defined in Listing 12.3.

LISTING 12.3 Location.java

```
public class Location {
    String locuri;
    String locname;

    public Location () {
        locuri=null;
        locname=null;
    }
    public Location (String name, String uri) {
        locname=name;
        locuri=uri;
    }
    String getLocUri() {
        return locuri;
    }
    String getLocName() {
        return locname;
    }
    void setLocUri(String _locuri) {
        locuri = _locuri;
    }
    void setLocName(String _locname) {
        locname = _locname;
    }
}
```

A data field of the internal data format that is not supported in the external data format, such as Appointment's location, is ignored when a CalendarItem object is created. The reverse is also true when an Appointment object of the internal data format is created. For example, CalendarItem's c_class and categories fields are ignored. To support synchronization, two data fields (remoteID and attribute) are added to the Appointment class. Listing 12.4 shows this modification of the Appointment class.

LISTING 12.4 Appointment.java

```
/*
 * Appointment.java
 *
 */

import java.util.Calendar;
import java.util.Date;
import java.util.Vector;
```

LISTING 12.4 Continued

```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import javax.microedition.rms.*;

public class Appointment {
    private int id;
    private String remoteId; //identifier on remove synch server
    private long time;
    private int length;
    private String location;
    private String subject;
    private String attribute; //possible value "dirty", "delete"

    public Appointment () {
        id=0;
        remoteId="";
        time=0;
        length=0;
        location="";
        subject="";
        attribute="";
    }

    public Appointment(Date date) {
        this();
        time=date.getTime();
        length=30;
    }

    public Appointment(int _id,long _time, int _length,
                       String _location, String _subject) {
        this();
        id=_id;
        time=_time;
        length=_length;
        location=_location;
        subject=_subject;
        //set attribute
        attribute="dirty";
    }

    public Appointment (int _id, byte[] rec) {
        this();
```

LISTING 12.4 Continued

```
        id=_id;
        init_app(rec);
    }

    public Appointment (CalendarItem item) {
        this();
        if(item.getTarget() !=null &&
            item.getTarget().getLocUri()!=null) {
            id=Integer.parseInt(item.getTarget().getLocUri());
        }
        if(item.getSource() !=null &&
            item.getSource().getLocUri()!=null) {
            remoteId=item.getSource().getLocUri();
        }
        if(item.getStart()!=null) {
            time=item.getStartLong();
        }
        if(item.getEnd()!=null) {
            length=(int)(item.getEndLong()-time)/60000;
        }
        if(item.getSummary()!=null) {
            subject=item.getSummary();
        }
    }

    public void init_app(byte[] rec) {
        // parse the record
        ByteArrayInputStream bais= new ByteArrayInputStream(rec);
        DataInputStream dis= new DataInputStream(bais);
        try {
            remoteId=dis.readUTF();
            time=dis.readLong();
            length=dis.readInt();
            location=dis.readUTF();
            subject=dis.readUTF();
            attribute=dis.readUTF();
        }catch(Exception e){}
    }

    public byte[] toBytes() {
        byte data[]=null;
        try {
            ByteArrayOutputStream baos= new ByteArrayOutputStream();
            DataOutputStream dos= new DataOutputStream(baos);
```

LISTING 12.4 Continued

```
        dos.writeUTF(remoteId);
        dos.writeLong(time);
        dos.writeInt(length);
        dos.writeUTF(location);
        dos.writeUTF(subject);
        dos.writeUTF(attribute);
        data=baos.toByteArray();

        baos.close();
        dos.close();
    }catch(Exception e) {}
    return data;
}

public String getTimeString() {
    StringBuffer sb= new StringBuffer();
    Calendar cal= Calendar.getInstance();
    cal.setTime(new Date(time));
    sb.append(cal.get(Calendar.MONTH)+1).append("/");
    sb.append(cal.get(Calendar.DAY_OF_MONTH)).append("/");
    sb.append(cal.get(Calendar.YEAR)).append(" ");
    sb.append(cal.get(Calendar.HOUR_OF_DAY)).append(":");
    if(cal.get(Calendar.MINUTE)<10) sb.append(0);
    sb.append(cal.get(Calendar.MINUTE));

    return sb.toString();
}

public int getId() {return id;}
public void setId(int _id) { id=_id;}
public String getRemoteId() {return remoteId;}
public void setRemoteId(String _remoteId) {remoteId=_remoteId;}
public long getTime() {return time;}
public void setTime(long _time){time=_time;}
public int getLength(){return length;}
public void setLength(int _length) {length=_length;}
public String getLocation() {return location;}
public void setLocation(String _location){location=_location;}
public String getSubject() {return subject;}
public void setSubject(String _subject) {subject=_subject;}
public String getAttribute() {return attribute;}
public void setAttribute(String _attribute) {
    attribute = _attribute;
}
}
```

12

Data Flow

The data flow of synchronization consists of many steps, as shown in Figure 12.3. We'll describe these steps in the following sections.

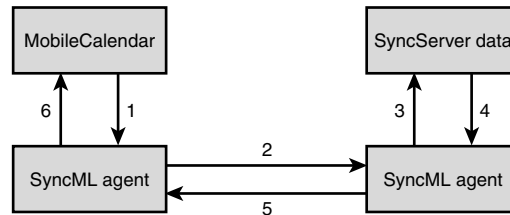


FIGURE 12.3

The data flow in a synchronization session.

Step 1: Collecting Data to be Synced On MIDP Devices

Before the client sync agent initiates a sync session with a sync server, it collects all calendar items that need to be synchronized. The items include newly created items for the Add operation, recently modified items for the Update operation, and items that the user wants to delete. Each item associates with a SyncML operation. The item and its associated SyncML operation are collectively represented by the SyncOperation class in Listing 12.5.

LISTING 12.5 SyncOperation.java

```

import java.util.*;

public class SyncOperation {
    String command_id;
    String command;
    Vector items;

    SyncOperation(String _command) {
        command_id = null;
        command = _command;
        items = new Vector();
    }

    public String toSyncML() {
        StringBuffer sb = new StringBuffer();
        sb.append("<").append(command).append(">");
        sb.append("<CmdID>").append(command_id).append("</CmdID>");
        sb.append("<Meta><mi:Type>text/x-vCalendar</mi:Type></Meta>");
        for(Enumeration e = items.elements(); e.hasMoreElements(); ) {

```

LISTING 12.5 Continued

```
        CalendarItem calItem= (CalendarItem) e.nextElement();
        sb.append(calItem.toSyncML());
    }
    return sb.toString();
}

void addItem(CalendarItem _item) {
    items.addElement(_item);
}

String getCommand() {
    return command;
}

String getCommandId() {
    return command_id;
}

Vector getItems() {
    return items;
}

void setCommand(String _command) {
    command = _command;
}

void setCommandId(String _command_id) {
    command_id = _command_id;
}
}
```

Sync operations are collected into a SyncOperations object, as shown in Listing 12.6.

LISTING 12.6 SyncOperations.java

```
import java.util.*;

public class SyncOperations extends Vector {
    String commandid;
    Location source;
    Location target;

    // Set the default values
    public SyncOperations() {
        super();
        commandid = null;
        source = null;
    }
}
```

LISTING 12.6 Continued

```
        target = null;
    }

    public String toSyncML() {
        StringBuffer sb = new StringBuffer();
        sb.append("<CmdID>").append(commandid).append("</CmdID>");
        sb.append("<Target><LocRUI>").append(target.getLocUri())
            .append("</LocRUI></Target>");
        sb.append("<Source><LocRUI>").append(source.getLocUri())
            .append("</LocRUI></Source>");
        for(Enumeration e = this.elements(); e.hasMoreElements(); ) {
            SyncOperation item = (SyncOperation) e.nextElement();
            sb.append(item.toSyncML());
        }
        return sb.toString();
    }

    String getCommandId() {
        return commandid;
    }
    Location getSource() {
        return source;
    }
    Location getTarget() {
        return target;
    }
    void setCommandId(String _commandid) {
        commandid = _commandid;
    }
    void setSource(Location _source) {
        source = _source;
    }
    void setTarget(Location _target) {
        target = _target;
    }
}
```

The sync operations are converted a SyncML message using the `toSyncML()` method of `SyncOperations` and `SyncOperation`. Each SyncML message has a header that contains sessionid, source, target locations, and authentication information. The header of a SyncML message is represented by a `SyncHeader` object, as shown in Listing 12.7.

LISTING 12.7 SyncHeader.java

```
/* SyncHeader is used for storing Header information
 */
public class SyncHeader {
    String sessionid;
    String msgid;
    String msgref;
    Location source;
    String sourceref;
    Location target;
    String targetref;
    String verdttd;
    String verproto;
    String user;
    String passwd;

    // Set the default values
    public SyncHeader() {
        sessionid = null;
        msgid = null;
        msgref = null;
        source = null;
        sourceref = null;
        target = null;
        targetref = null;
        verdttd = null;
        verproto = null;
        user=null;
        passwd=null;
    }

    public void setDefault(SynchOption so) {
        sessionid=Integer.toString(1);
        msgid=Integer.toString(1);
        verdttd="1.0";
        verproto="SynchML/1.0";
        target= new Location();
        target.setLocUri(so.getUrl());
        source = new Location();
        source.setLocUri("my_phone");
        user=so.getUser();
        passwd=so.getPasswd();
    }

    public String toString() {
        StringBuffer sb= new StringBuffer();
```

LISTING 12.7 Continued

```
        sb.append("<SyncHdr>");
        sb.append("<VerDTD>").append(verdtd).append("</VerDTD>");
        sb.append("<SessionID>").append(sessionid)
            .append("</SessionID>");
        sb.append("<MsgID>").append(msgid).append("<MsgID>");
        sb.append("<Target>");
        sb.append("<LocURI>").append(target.getLocUri())
            .append("</LocURI>");
        sb.append("</Target>");
        sb.append("<Source>");
        sb.append("<LocURI>").append(source.getLocUri())
            .append("</LocURI>");
        sb.append("</Source>");
        sb.append("<Cred>");
        sb.append("<Data>").append(user).append(":").append(passwd)
            .append("</Data>");
        sb.append("</Cred>");
        sb.append("</SyncHdr>");

        return sb.toString();
    }

    String getMsgId() {
        return msgid;
    }

    String getMsgRef() {
        return msgref;
    }

    String getSessionId() {
        return sessionid;
    }

    Location getSource() {
        return source;
    }

    String getSourceRef() {
        return sourceref;
    }

    Location getTarget() {
        return target;
    }

    String getTargetRef() {
        return targetref;
    }

    String getVerDTD() {
```

LISTING 12.7 Continued

```
        return verDTD;
    }
    String getVerProto() {
        return verproto;
    }
    void setMsgId(String _msgid) {
        msgid = _msgid;
    }
    void setMsgRef(String _msgref) {
        msgref = _msgref;
    }
    void setSessionId(String _sessionid) {
        sessionid = _sessionid;
    }
    void setSource(Location _source) {
        source = _source;
    }
    void setSourceRef(String _sourceref) {
        sourceref = _sourceref;
    }
    void setTarget(Location _target) {
        target = _target;
    }
    void setTargetRef(String _targetref) {
        targetref = _targetref;
    }
    void setVerDTD(String _verDTD) {
        verDTD = _verDTD;
    }
    void setVerProto(String _verproto) {
        verproto = _verproto;
    }
}
```

Step 2: Sending SyncML Messages to Sync Servers

After obtaining all the sync operations that the client wants the sync server to perform, the client sync agent establishes an HTTP connection with the sync server. It then passes the SyncML message in an HTTP request to the sync server.

Step 3: Updating Data on Sync Servers

The sync agent on the server side extracts the SyncML operations and updates the data on the sync server accordingly.

Step 4: Collecting Data to be Synced on Servers

The server sync agent collects information that needs to be passed to the client and converts it into a SyncML message. This message is sent in the HTTP response.

Step 5: Extracting SyncML Operations

The client sync agent waits for a server response after sending a request to the server until it reads an HTTP response that contains a SyncML message. A SAX XML parser (discussed in Chapter 10, “Using XML in Wireless Applications”) is used to parse the SyncML message and extract SyncML operations. The SyncML handler is defined in Listing 12.8.

LISTING 12.8 SyncMLHandler.java

```
import java.util.*;
import java.io.*;
import org.xml.sax.*;

/* this version of SyncML handler supports three
 * sync operations: Add, Delete, Update
 */
public class SyncMLHandler extends HandlerBase {

    // the Vector used for storing SyncOperations
    private SyncOperations operationList;
    // the current XML element being processed
    private String currTag;
    //the type of item (vcard or vcalendar)
    private String miType;
    // the current CalendarItem being parsed
    private CalendarItem currItem;
    // the current SyncOperation object being constructed
    private SyncOperation currOperation;
    // XML element tracker
    private Stack elementStack;
    // syncHeader
    private SyncHeader syncHeader;
    // source
    private Location currSource;
    // target
    private Location currTarget;

    SyncMLHandler() {
        elementStack = new Stack();
        syncHeader = new SyncHeader();
        operationList = new SyncOperations();
    }
}
```

LISTING 12.7 Continued

```
public SyncOperations getOperations() {
    return operationList;
}
public void startElement (String name, AttributeList attributes)
    throws SAXException {
    currTag = name;
    if (currTag.equals("SyncHdr")) {
        elementStack.push(currTag);
    } else if (currTag.equals("SyncBody")) {
        elementStack.push(currTag);
    } else if (currTag.equals("Sync")) {
        elementStack.push(currTag);
    } else if (currTag.equals("Add") ||
        currTag.equals("Delete") ||
        currTag.equals("Update")){
        currOperation = new SyncOperation(currTag);
        elementStack.push(currTag);
    } else if (currTag.equals("Item")) {
        currItem = new CalendarItem();
        elementStack.push(currTag);
    } else if (currTag.equals("Source")) {
        currSource = new Location();
        elementStack.push(currTag);
    } else if (currTag.equals("Target")) {
        currTarget = new Location();
        elementStack.push(currTag);
    }
}
public void endElement (String name)
    throws SAXException {
    currTag = name;
    if (currTag.equals("SyncHdr")) {
        Object tmp = elementStack.pop();
    } else if (currTag.equals("SyncBody")) {
        Object tmp = elementStack.pop();
    } else if (currTag.equals("Sync")) {
        Object tmp = elementStack.pop();
    } else if (currTag.equals("Add") ||
        currTag.equals("Delete") ||
        currTag.equals("Update")) {
        Object tmp = elementStack.pop();
        operationList.addElement(currOperation);
        currOperation = null;
    }
}
```

LISTING 12.8 Continued

```

    } else if (currTag.equals("Item")) {
        Object tmp= elementStack.pop();
        if(miType.equals("text/x-vCalendar")) {
            currOperation.addItem(currItem);
        }
        currItem = null;
    } else if (currTag.equals("Source")) {
        Object tmp = elementStack.pop();
        String parent = (String) elementStack.peek();
        if (parent.equals("SyncHdr"))
            syncHeader.setSource(currSource);
        else if (parent.equals("Item"))
            currItem.setSource(currSource);
        else if (parent.equals("Sync"))
            operationList.setSource(currSource);

        currSource = null;
    } else if (currTag.equals("Target")) {
        Object tmp = elementStack.pop();
        String parent = (String) elementStack.peek();
        if (parent.equals("SyncHdr"))
            syncHeader.setTarget(currTarget);
        else if (parent.equals("Item"))
            currItem.setTarget(currTarget);
        else if (parent.equals("Sync"))
            operationList.setTarget(currTarget);
        currTarget = null;
    }
    currTag = "NULL";
}
public void characters (char ch[], int start, int length)
    throws SAXException {
    if (!currTag.equals("NULL")) {
        String contents = new String(ch, start, length);
        // populate the fields of current Book with parsed data
        if (currTag.equals("SessionID")) {
            syncHeader.setSessionId(contents);
        } else if (currTag.equals("MsgID")) {
            syncHeader.setMsgId(contents);
        } else if (currTag.equals("MsgRef")) {
            syncHeader.setMsgRef(contents);
        } else if (currTag.equals("VerDTD")) {
            syncHeader.setVerDTD(contents);
        } else if (currTag.equals("VerProto")) {

```

LISTING 12.8 Continued

```

        syncHeader.setVerProto(contents);
    } else if (currTag.equals("mi:Type")) {
        miType=contents;
    } else if (currTag.equals("Data")) {
        //we only implement vcalendar here. Ignore vcard
        if(miType.equals("text/x-vCalendar")) {
            currItem.setData(contents);
        }
    } else if (currTag.equals("LocURI")) {
        String parent = (String) elementStack.peek();
        if (parent.equals("Source"))
            currSource.setLocUri(contents);
        else if (parent.equals("Target"))
            currTarget.setLocUri(contents);
    } else if (currTag.equals("LocName")) {
        String parent = (String) elementStack.peek();
        if (parent.equals("Source"))
            currSource.setLocName(contents);
        else if (parent.equals("Target"))
            currTarget.setLocName(contents);
    } else if (currTag.equals("CmdID")) {
        String parent = (String) elementStack.peek();
        if (parent.equals("Sync"))
            operationList.setCommandId(contents);
        else if (parent.equals("Add") ||
            parent.equals("Delete") ||
            parent.equals("Update"))
            currOperation.setCommandId(contents);
    }
    }
}
}
public SyncOperations getOperationList() {
    return operationList;
}
}
}

```

Step 6: Updating Client Data

At last, the client sync agent updates the Mobile Calendar information according to the extracted SyncML operations.

Steps 3 and 4 are functions of a sync server. The sync server can be implemented using a Java servlet; this implementation is out of the scope of this chapter.

Steps, 1, 2, 5, and 6 are functions of a client sync agent. They are implemented in `SyncAgent.java`, shown in Listing 12.9.

LISTING 12.9 `SyncAgent.java`

```
import java.io.*;
import java.util.*;
import javax.microedition.midlet.*;
import javax.microedition.io.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;

public class SyncAgent {
    SyncOperations toServer;
    SyncOperations fromServer;
    CalendarDB     calendarDB;
    SynchOption    so;

    public SyncAgent(CalendarDB calDB, SynchOption s) {
        calendarDB= calDB;
        so=s;
    }

    public boolean startSync() {
        boolean success=false;
        HttpURLConnection hc = null;
        DataInputStream dis = null;
        OutputStream os = null;

        try {
            hc = (HttpURLConnection)
                Connector.open(so.getUrl(), Connector.READ_WRITE);
            hc.setRequestMethod(HttpURLConnection.POST);

            StringBuffer sb= new StringBuffer("<SyncML>");
            //create SyncHeader
            SyncHeader syncHdr= new SyncHeader();
            syncHdr.setDefault(so);
            sb.append(syncHdr.toString());
            //create syncBody
            sb.append("<SyncBody>");
            sb.append("<sync>");
            int cmdid=1;
            toServer=calendarDB.getToSync(cmdid);
            toServer.setSource(new Location("", "CalendarDB"));
            toServer.setTarget(new Location("", so.getUser()+".nsf"));
        }
    }
}
```

LISTING 12.9 Continued

```

sb.append(toServer.toSyncML());
sb.append("</Sync>");
//end
sb.append("<Final/>");
sb.append("</SyncBody>");
sb.append("</SyncML>");

os = hc.openOutputStream();
os.write(sb.toString().getBytes());
os.flush();
os.close();

/* clear the attribute of local copies if the operation
 * is Add or Update. Physically delete the local copy if
 * the operation is Delete.
 */
for(Enumeration e = toServer.elements();
    e.hasMoreElements(); ) {
    SyncOperation syncOp = (SyncOperation) e.nextElement();
    Vector items = syncOp.getItems();
    for(Enumeration eo = items.elements();
        eo.hasMoreElements();){
        CalendarItem calItem =
            (CalendarItem) eo.nextElement();
        Appointment app=calendarDB.getAppointmentById(
            Integer.parseInt(
                calItem.getSource().getLocUri()));
        app.setAttribute("");
        if(syncOp.getCommand().equals("Delete")) {
            calendarDB.delete(app);
        }
        else if(syncOp.getCommand().equals("Add") ||
            syncOp.getCommand().equals("Update")) {
            calendarDB.update(app);
        }
    }
}

dis = hc.openDataInputStream();
//dis = new DataInputStream(new
    ByteArrayInputStream(data.getBytes()));

//set xml parser
Parser parser =

```

LISTING 12.9 Continued

```
ParserFactory.makeParser("com.microstar.xml.SAXDriver");
SyncMLHandler syncmlHandler = new SyncMLHandler();
parser.setDocumentHandler(syncmlHandler);

//get all the operations
InputStream inputSource = new InputStream(dis);
parser.parse(inputSource);
fromServer = syncmlHandler.getOperationList();

// print out parsing results
System.out.println("operationList_size:" +
    fromServer.size());
System.out.println("operationList_commandid:" +
    fromServer.getCommandId());
Location loc1;
if ( (loc1 = fromServer.getSource())!= null )
    System.out.println("operationList_sourcelocuri:" +
        loc1.getLocUri());
if ( (loc1 = fromServer.getTarget())!= null )
    System.out.println("operationList_targetlocuri:" +
        loc1.getLocUri());
Enumeration e = fromServer.elements();

// print out sync operations
while (e.hasMoreElements()) {
    SyncOperation syncOperation =
        (SyncOperation) e.nextElement();
    System.out.println(" command:" +
        syncOperation.getCommand());
    System.out.println(" command_id:" +
        syncOperation.getCommandId());
    Vector items = syncOperation.getItems();

    // print out data items in operations
    for (int i = 0; i < items.size(); i++) {
        CalendarItem item =
            (CalendarItem) items.elementAt(i);
        if(syncOperation.getCommand().equals("Add")) {
            /* we need map operation to complete this
             * operation. Without it, we will synch this
             * record later to pass local id to server.
             */

            calendarDB.mark4Synch(new Appointment(item));
        }
    }
}
```

LISTING 12.9 Continued

```
        else if(syncOperation.getCommand()
                .equals("Delete")) {
            Appointment app = new Appointment(item);
            app.setId(calendarDB.findByIdByRemoteId(app));
            calendarDB.delete(app);
        }
        else if(syncOperation.getCommand()
                .equals("Update")) {
            Appointment app = new Appointment(item);
            app.setId(calendarDB.findByIdByRemoteId(app));
            calendarDB.update(app);
        }
        System.out.println("  item_summary:" +
                item.getSummary());
        System.out.println("  item_description:" +
                item.getDescription());
        System.out.println("  item_start:" +
                item.getStart());
        System.out.println("  item_end:" + item.getEnd());
    }
    }
    success=true;
} catch (IOException ie) {
    System.err.println("IO Error:" + ie);
} catch (SAXException se) {
    System.err.println("XML Error:" + se);
} catch (Exception e) {
    System.err.println("Other Error:" + e);
} finally {
    // freeing up i/o streams and http connection
    try { if (hc != null) hc.close();
    } catch (IOException ignored) {}
    try { if (dis != null) dis.close();
    } catch (IOException ignored) {}
    try { if (os != null) os.close();
    } catch (IOException ignored) {}
}
return success;
}
}
```

Linking with the Rest of the MobileScheduler Functions

The synchronization function is added to the main menu of the MobileScheduler application. When appointments are listed, appointments that need to be synced are marked with an asterisk (*), and appointments that are deleted locally are marked with a letter *d*. The updated Scheduler.java appears in Listing 12.10.

LISTING 12.10 Scheduler.java

```
import java.util.Calendar;
import java.util.Vector;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class Scheduler extends MIDlet implements CommandListener{
    private Calendar      calendar;
    private List          menu;
    private AppointmentForm appForm=null;
    private List          appList=null;
    private SynchOptionForm soForm=null;
    private MyCalendar    mycalendar=null;
    private String[]      options={"Add Appointment",
                                  "Retrieve Appointments",
                                  "Synch Setup",
                                  "Start sync",
                                  "Calendar view"};

    private Display      display;
    private Command      backCommand= new Command("Back",
                                                    Command.BACK, 1);
    private Command      exitCommand = new Command("Exit",
                                                    Command.EXIT, 1);

    private CalendarDB   calendarDB;
    private Vector       apps;
    private SynchOption  so;
    public Scheduler() {
        //create an implicit choice list, and use it as start menu
        menu= new List("Scheduler", List.IMPLICIT,options,null);
        menu.addCommand(exitCommand);
        menu.setCommandListener(this);

        //get a calendar
        calendar=Calendar.getInstance();
    }
}
```

LISTING 12.10 Continued

```
calendarDB = new CalendarDB();
so = new SynchOption();

//retrieve display
display=Display.getDisplay(this);
}
public void startApp() throws MIDletStateChangeException {
    display.setCurrent(menu);
}

/**
 * Pause the MIDlet
 */
public void pauseApp() {
}

/**
 * Called by the framework before the application is unloaded
 */
public void destroyApp(boolean unconditional) {
    //clear everything
    menu= null;
    calendar=null;
    display=null;
    appForm = null;
    appList =null;
    apps=null;
    soForm = null;
    mycalendar=null;
}

public void commandAction(Command c, Displayable d) {
    if(d==menu && c==List.SELECT_COMMAND) {
        switch(menu.getSelectedIndex()) {
            case 0: //Add appointment
                //create a new appointment from
                appForm = new AppointmentForm(display, menu,
                    calendarDB);
                appForm.setAppointment(new
                    Appointment(calendar.getTime()));
                display.setCurrent(appForm);
                break;
            case 1: //retrieve appointments
                //create an appointment list
```

LISTING 12.7 Continued

```
appList = new List("Appointments", List.IMPLICIT);
appList.addCommand(backCommand);
appList.setCommandListener(this);

//retrieve all the appointments
apps= calendarDB.retrieveAll();
for(int i=0; i<apps.size(); i++) {
    Appointment app= (Appointment) apps.elementAt(i);
    StringBuffer sb = new StringBuffer();
    if (app.getAttribute().equals("dirty")) {
        sb.append("* ");
    }
    else if(app.getAttribute().equals("delete")) {
        sb.append("d ");
    }
    else {
        sb.append(" ");
    }
    sb.append(app.getTimeString()).append(" ")
        .append(app.getSubject());
    appList.append(sb.toString(),null);
}
display.setCurrent(appList);
break;
case 2: //synchronization set up
if(soForm==null) {
    //synchsetting
    soForm = new SynchOptionForm(display,menu,so);
}
display.setCurrent(soForm);
break;
case 3: //start synch
Alert alert = new Alert ("Sync Info.");
alert.setTimeout(Alert.FOREVER);
if(new SyncAgent(calendarDB, so).startSync()) {
    alert.setString("Success!");
}
else {
    alert.setString("Failed!");
}
display.setCurrent(alert,menu);
break;
```

LISTING 12.7 Continued

```
        case 4: // calendar view
            if(mycalendar==null) {
                //mycalendar
                mycalendar= new MyCalendar(display, menu,
                                           calendar);
            }
            display.setCurrent(mycalendar);
            break;
        default:
        }
    }
}
else if(d==menu && c==exitCommand ) {
    calendarDB.close();
    destroyApp(true);
    notifyDestroyed();
}
else if(d==appList) {
    if(c==List.SELECT_COMMAND) {
        //create a new appointment from
        appForm = new AppointmentForm(display, menu,
                                      calendarDB);

        appForm.setAppointment(
            (Appointment)apps.elementAt(
                appList.getSelectedIndex()));
        display.setCurrent(appForm);
    }
    else if(c==backCommand) {
        display.setCurrent(menu);
    }
}
}
}
```

12**DATA
SYNCHRONIZATION
FOR WIRELESS**

In the preceding chapters, you developed functionality for adding, modifying, and deleting appointments. With the synchronization issue in mind, deleting an appointment on the MIDP device will only mark the record for deletion. The record will not be physically removed from data storage until a synchronization completes successfully. In addition, new methods are provided to collect appointments to be synchronized and to search appointments by data identification on a sync server. The updated version of `CalendarDB.java` is shown in Listing 12.11.

LISTING 12.11 CalendarDB.java

```
/*
 * CalendarDB.java
 *
 */

import java.io.*;
import java.util.*;
import javax.microedition.rms.*;

public class CalendarDB {
    RecordStore rs=null;
    public CalendarDB () {
        //the file to store the db is "calendarDB"
        String file="calendarDB";
        try {
            // open a record store named file
            rs = RecordStore.openRecordStore(file,true);
        }catch(Exception e) {
            System.out.println("Error: "+e.getMessage());
        }
    }

    //add new record
    public synchronized boolean add(Appointment app) {
        if(rs==null) return false;
        boolean success=false;
        try {
            byte data[]=app.toBytes();
            rs.addRecord(data,0,data.length);
            success=true;
        }catch(Exception e) {
            System.out.println("Error: "+e.getMessage());
        }
        return success;
    }

    //close the record store
    public void close() {
        if(rs!=null) {
            try {
                rs.closeRecordStore();
            }catch (Exception e){}
        }
    }
}
```

LISTING 12.11 Continued

```
    }
}

//delete a record
public synchronized boolean delete(Appointment app) {
    boolean success=false;
    int id=app.getId();
    if(id==0) return false;

    try {
        rs.deleteRecord(id);
        success=true;
    }catch(Exception e) {}

    return success;
}

//find the id of record whose remoteId match input's remoteId
public int findIdByRemoteId(Appointment app) {
    if(rs==null) return 0;

    RecordEnumeration re=null ;
    int id=0;
    String remoteId= app.getRemoteId();
    try {
        if(rs.getNumRecords()==0) return 0;
        //try to find out the record id
        re= rs.enumerateRecords((RecordFilter)null,
                               (RecordComparator)null, false);
        while(re.hasNextElement() &&id==0) {
            int i= re.nextRecordId();
            Appointment a = new Appointment(i, rs.getRecord(i));
            if(remoteId.equals(a.getRemoteId())) {
                id=i;
            }
        }
    }
    }catch(Exception e) {}
    finally{
        //clear
        try {
            if(re!=null) re.destroy();
        }catch(Exception e) {}
    }
}
```

LISTING 12.11 Continued

```
        return id;
    }

    public Appointment getAppointmentById(int id) {
        Appointment app=null;
        try {
            byte data[]=rs.getRecord(id);
            app= new Appointment (id, data);
        }catch(Exception e) {}
        return app;
    }

    /* Mark for delete. If there is a remote copy, the record
    *will not be physically removed until synchronization completes.
    */
    public boolean mark4Delete(Appointment app) {
        if(app.getRemoteId().length()==0) { //there is only local copy
            return delete(app);
        }
        app.setAttribute("delete");
        return update(app);
    }

    //set attribute to dirty. Mark for Synchronization
    public boolean mark4Synch(Appointment app) {
        app.setAttribute("dirty");
        if(app.getId(>)>0) return update(app);
        else return add(app);
    }

    public Vector retrieveAll() {
        RecordEnumeration re=null;
        Vector apps= new Vector();
        try {
            //cutoff is 90 days old
            long cutoff=System.currentTimeMillis()-
                new Integer(90).longValue()*24*60*60000;
            RecordFilter rf = new AppointmentFilter(cutoff);
            RecordComparator rc = new AppointmentComparator();
            re = rs.enumerateRecords(rf,rc,false);
            while(re.hasNextElement()) {
                int rec_id=re.nextRecordId();
                apps.addElement(
                    new Appointment(rec_id,rs.getRecord(rec_id)));
            }
        }
    }
}
```

LISTING 12.11 Continued

```

    }catch(Exception e) {}
    finally{
        //close the record store
        if(re!=null) re.destroy();
    }
    return apps;
}

public SyncOperations getToSync(int startCmdId) {
    SynchOption so = new SynchOption();
    SyncOperations synOps= new SyncOperations();
    synOps.setCommandId(Integer.toString(startCmdId));
    int cmdId=startCmdId;
    RecordEnumeration re=null;
    try {
        re = rs.enumerateRecords(null, null, false);
        while(re.hasNextElement()) {
            int rec_id=re.nextRecordId();
            Appointment app= new
                Appointment(rec_id,rs.getRecord(rec_id));
            if(app.getAttribute().equals("dirty") ||
                app.getAttribute().equals("delete")) {
                SyncOperation synOp=null;
                cmdId++;
                if(app.getAttribute().equals("dirty")) {
                    if(app.getRemoteId().length(>0) { //for update
                        synOp = new SyncOperation("Update");
                    }
                    else {
                        synOp = new SyncOperation("Add");
                    }
                }
                else if(app.getAttribute().equals("delete")) {
                    synOp = new SyncOperation("Delete");
                }
                synOp.addItem(new CalendarItem(app, so));
                synOps.setCommandId(Integer.toString(cmdId));
                synOps.addElement(synOp);
            }
        }
    }catch(Exception e) {}
    finally{
        //close the record store
        if(re!=null) re.destroy();
    }
}

```

LISTING 12.11 Continued

```
        return synOps;
    }

    public synchronized boolean update(Appointment app) {
        if(rs==null) return false;
        int id= app.getId();
        if(id==0) return false;

        boolean success=false;
        try {
            byte[] data= app.toBytes();
            rs.setRecord(id, data, 0, data.length);
            success=true;
        }catch(Exception e){}
        return success;
    }
}
```

Other classes used in the MobileScheduler applications are defined in AppointmentComparator.java, AppointmentFilter.java, SynchOption.java, and SynchOptionForm.java. These files are the same as the ones used in previous chapters, so they are not listed here.

The XML parser class is included in the xm1.jar file. To compile and execute the application, you should include xm1.jar in your classpath.

Sun's emulator contains bugs relating to RecordStore. These bugs will affect the execution of this application. Thus, the examples are shown using Motorola's emulator.

NOTE

As the internal data format is changed to include synchronization information, the old appointment data you stored in the record store needs to be deleted before you save new appointments.

For example, suppose you start the scheduler and insert two appointments. You can retrieve them as shown in Figure 12.4. There is an asterisk in front of each appointment that indicates the appointment needs to be synchronized with the sync server. You can retrieve one appointment and delete it. The appointment will be physically removed from local storage because no entry on the sync server needs to be synchronized.

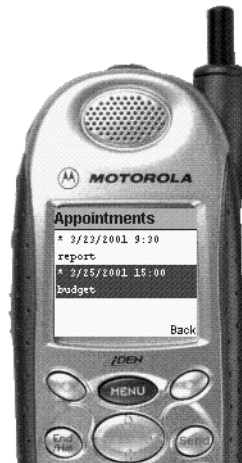


FIGURE 12.4

Retrieving appointments from the scheduler.

Now you want to sync appointments with a sync server. A test sync server is set up for this book. You can use the Synch Setting operation to set the SynchOption's URL as follows

```
http://64.28.105.108/servlets/webyu/SamsbookSyncAgent
```

which is served by www.webyu.com. After the URL has been set, you can start synchronization. If everything goes well, you will see a screen showing a successful message. After dismissing the screen, you can retrieve all appointments. You will see a screen like Figure 12.5. An appointment is added by the sync server. Because this appointment needs to be synced with the server again to update its local id to the server, it is marked with an asterisk.

You have now completed the MobileScheduler application, which contains four components: the GUI in which the user can modify and view appointments; persistent storage to hold all appointments; a calendar view for displaying monthly appointments; and a sync agent for data synchronization.



FIGURE 12.5

Retrieving all appointments after synchronization.

Summary

This chapter discussed the need for universal data synchronization. We talked about the SyncML open standard for data synchronization, and we implemented a subset of the SyncML protocol in the MobileScheduler example.