Accessing PIM Databases

USAGE RESTRICTIONS:

You are being provided these materials solely as a member of Palm's PluggedIn Program (the "PIP").

Your right to access and use these "PluggedIn Materials" is subject to and governed by the terms and conditions of the PluggedIn Program License Agreement which you agreed to at the time you became a member of PIP.

TABLE OF CONTENTS

Target Audience

Palm OS developers who have or intend to access the new PIM databases directly to retrieve or insert data programmatically from third-party applications.

Revision History

Date	**Revision #**	**Description of changes**
April 18, 2005	1.00	Created application notes.
November 30, 2005	2.00	Updated to include contacts changes.
May 15, 2006	3.00	Updated template.
Sept. 11, 2006	4.00	Copyedited for clarity and inclusion in the *Palm OS Platform Developer Guide*

1. Introduction

1.1. The PIM SDK

The PIM SDK includes the latest information on PIM databases and structures, including header files, documentation, and sample code. It must be downloaded separately from the latest Palm OS SDK from **http://pluggedin.palm.com**.

IMPORTANT: Header files in the PIM SDK are not meant to be used as they are. The purpose of distributing the PIM SDK is to illustrate the changes in the database structures since the previous version, and explain how to use the new fields.

1.2. Overview

Personal information management (PIM) applications are the core feature of Palm mobile devices. Users rely on PIM applications to store and access their personal data to perform business and personal tasks. Since late 2004, existing PIM applications (Address Book, Datebook, ToDo, and Memo) have been enhanced to add new fields in the database. These changes were made by creating new PIM applications with different creator IDs (Contacts, Calendar, Tasks, and Memos). So the current PIM applications include both the old and new creator IDs.

This document will explain the changes to the PIM application database structures and code samples in the PIM SDK since Palm OS version 5.4 R3.

To build applications that access the PIM databases, such as Contacts, Calendars, Tasks, and Memos, refer to the header files and documentation included with the PIM SDK.

The PIM code samples that are distributed with the SDK are a reference on how to access the databases and how to get and set data outside the PIM applications. With the introduction of the new database fields, some of the functions in the source code of the code samples must be changed to adapt to the new structures. The purpose of the sections on specific code samples is to identify and document these changes.

2. PIM Database Structures

IMPORTANT: Applications should always use the APIs, and not access database fields directly. PIM application database structures may change in the future, and if your application accesses fields directly, it may break.

The content of the legacy PIM databases is accessible through a compatibility layer that simulates the old databases. This layer assumes that PalmSource's format is respected. For this reason, using different content such as an encrypted blob, will cause a device to crash.

The following tables include the old and new creator IDs for the PIM applications, as well as the types for applications, databases, and the compatibility layer:

Legacy		**New**	
Application	**Creator ID**	**Application**	**Creator ID**
Address Book	'addr'	**Contacts**	**'PAdd'**
Datebook	'date'	**Calendar**	**'PDat'**
Todo	'todo'	**Tasks**	**'PTod'**
Memo	'memo'	**Memos**	**'PMem'**
		Compatibility Later	**'pdmE'**

Item	**Type**
Applications	'appl'
Databases	'DATA'
Compatibility layer	'aexo'

Legacy PIM databases exist on the device, but they are empty when not accessed. Records are created on the fly when an application reads or writes to the databases. For this reason, the legacy structures have some impact on performance, especially when a large number of records are present. For example, you may see a performance issue when AddressDB is opened, because data is copied from the legacy ContactsDB-PAdd, and records are created in the new AddressDB.

If your application will provide a different PIM interface, first make sure the new PIM applications and the compatibility layer exist on the device. Where the new enhanced databases are available, use them instead of the legacy databases in your application.

Another change in the database structures is the use of binary large objects (blobs). A blob is a collection of binary data stored as a single item in a database, such as a multimedia files like images, video, or sound. In the old PIM database structures, blobs used to follow at the end of records. With the new PIM database structures, blobs are now part of the record structure and are easier to access. Blobs are available for use by developers, and follow the format of creator ID, size, and data.

IMPORTANT: The data of a blob can contain anything as long as it does not exceed 1K in size.

Blobs generally use the following database structure:

```
typedef struct
{
     UInt32 creatorID;
     UInt16 size;
     void * content;
} BlobType;
```

For more specific information on the structures of the enhanced PIM databases, see the header files in the PIM SDK.

2.1. Contacts

This section describes the changes in the database structures for the Contacts application since Palm OS version 5.4 R3. It also illustrates the changes that must be made to use the code samples for packing and unpacking records.

The enhancements to the database structures for the Contacts application are shown here:

```
typedef struct
{
      AddrOptionsType        options;
      Char *                 fields[addrNumStringFields];
      BirthdayInfo           birthdayInfo;               // NEW
      AddressDBPictureInfo   pictureInfo;                // NEW
       Release2BlobType      rel2blobInfo;               // NEW
      UInt16                 numBlobs;                   // NEW
      BlobType               blobs[apptMaxBlobs];        // NEW

} AddrDBRecordType;
```

Two structures have been added to the Birthday and Picture fields in the database: `birthdayInfo` and `pictureInfo`. As the result of these additions, the number of fields was also increased from `addressFieldsCount` in the old `AddressDB.h` to `addrNumStringFields` in the `AddressDB.h` included in this SDK.

`rel2blobInfo` is also new, and uses the following type:

```
typedef struct
{
    UInt16                      dirty;
    BirthdayInfo                anniversaryInfo;
    ToneIdentifier              ringtoneInfo;

} Release2BlobType;
```

For more specific information on the new database structures, refer to the header files in the PIM SDK.

To adapt to the changes in the Contacts database structures, the record pack and unpack functions in the code samples must be modified as well. These changes are described in the following sections.

2.1.1. Contacts - Packing Records Code Sample

```
void PrvAddrDBPack(AddrDBRecordPtr s, void * recordP)
{
      Int32 offset;
      AddrDBRecordFlags flags;
      Int16 index;
      PrvAddrPackedDBRecord* d = 0;
      UInt16 len;
      void * srcP;
      UInt8 companyFieldOffset;
      UInt16 blobCount = 0;
      UInt16 size = 0;

      flags.allBits = 0;
      flags.allBits2 = 0;

      DmWrite(recordP, (Int32)&d->options, &s->options, sizeof(s->options));
      offset = (Int32)&d->firstField;

      for (index = firstAddressField; index < addrNumStringFields; index++)
      {
            if (s->fields[index] != NULL)
            {
                  if ((s->fields[index][0] != '\0') || (index == note))
                  {
                        srcP = s->fields[index];
                        len = StrLen((Char*)srcP) + 1;
                        DmWrite(recordP, offset, srcP, len);
                        offset += len;
                        SetBitMacro(flags, index);
                  }
            }
      }

      len = sizeof(DateType);

      if(s->birthdayInfo.birthdayDate.month && s->birthdayInfo.birthdayDate.day)
      {
            DmWrite(recordP, offset, &(s->birthdayInfo.birthdayDate), len);
            offset += len;
            SetBitMacro(flags, birthdayDate);

            len = sizeof(AddressDBBirthdayFlags);
            DmWrite(recordP, offset, &(s->birthdayInfo.birthdayMask), len);
            offset += len;
            SetBitMacro(flags, birthdayMask);

            if(s->birthdayInfo.birthdayMask.alarm)
            {
                  len = sizeof(UInt8);
                  DmWrite(recordP, offset, &(s->birthdayInfo.birthdayPreset), len);
                  offset += len;
                  SetBitMacro(flags, birthdayPreset);
            }
      }

      if(s->pictureInfo.pictureSize && s->pictureInfo.pictureData)
      {
            UInt32 blobId = addrPictureBlobId;
            UInt16 blobSize = 0;

            len = sizeof(blobId);
            DmWrite(recordP, offset, &blobId, len);
            offset += len;

            len = sizeof(blobSize);
            blobSize = s->pictureInfo.pictureSize + sizeof(s->pictureInfo.pictureDirty);
            DmWrite(recordP, offset, &blobSize, len);
            offset += len;

            len = sizeof(s->pictureInfo.pictureDirty);
            DmWrite(recordP, offset, &(s->pictureInfo.pictureDirty), len);
            offset += len;

            DmWrite(recordP, offset, (s->pictureInfo.pictureData),
                  s->pictureInfo.pictureSize);
            offset += s->pictureInfo.pictureSize;

            blobCount++;
      }

      if( (s->rel2blobInfo.anniversaryInfo.birthdayDate.month
           && s->rel2blobInfo.anniversaryInfo.birthdayDate.day)
         || s->rel2blobInfo.ringtoneInfo.id )
      {
            UInt32 blobId = addrRelease2BlobId;
            UInt16 blobSize = 0;

            len = sizeof(blobId);
            DmWrite(recordP, offset, &blobId, len);
            offset += len;

            len = sizeof(blobSize);
            blobSize = sizeof(ToneIdentifier) + sizeof(BirthdayInfo);
            DmWrite(recordP, offset, &blobSize, len);
            offset += len;

            DmWrite(recordP, offset, &(s->rel2blobInfo.anniversaryInfo), blobSize);
            offset += len;

            blobCount++;
      }

      ErrNonFatalDisplayIf(blobCount + s->numBlobs > apptMaxBlobs, "Too many blobs");
      for (index = 0; index < s->numBlobs; index++)
      {
            size = sizeof(s->blobs[index].creatorID);
            DmWrite(recordP, offset, &(s->blobs[index].creatorID), size);
            offset += size;
            size = sizeof(s->blobs[index].size);
            DmWrite(recordP, offset, &(s->blobs[index].size), size);
            offset += size;
            DmWrite(recordP, offset, s->blobs[index].content, s->blobs[index].size);
            offset += s->blobs[index].size;
      }

      ErrNonFatalDisplayIf(offset > MemHandleSize(MemPtrRecoverHandle(recordP)),
             "Not enough room for packed record");

      DmWrite(recordP, (Int32)&d->flags.allBits, &flags.allBits, sizeof(flags.allBits));
      DmWrite(recordP, (Int32)&d->flags.allBits2, &flags.allBits2, sizeof(flags.allBits2));

      if (s->fields[company] == NULL)
            companyFieldOffset = 0;
      else
      {
            index = 1;
            if (s->fields[name] != NULL)
                  index += StrLen(s->fields[name]) + 1;
            if (s->fields[firstName] != NULL)
                  index += StrLen(s->fields[firstName]) + 1;
            companyFieldOffset = (UInt8) index;
      }
      DmWrite(recordP, (Int32)(&d->companyFieldOffset), &companyFieldOffset,
sizeof(companyFieldOffset));
}
```

2.1.2. Contacts - Unpacking Records Code Sample

```
void PrvAddrDBUnpack(PrvAddrPackedDBRecord *src, AddrDBRecordPtr dest)
{
      Int16 index;
      AddrDBRecordFlags     flags;
      Char * p;
      UInt16 recordSize = 0;
      Char * blobStart;
      Char * blobEnd;
      UInt32 blobId;
      UInt16 blobSize;

      MemMove(&(dest->options), &(src->options), sizeof(AddrOptionsType));

      MemMove(&flags, &(src->flags), sizeof(AddrDBRecordFlags));
      p = &src->firstField;

      for (index = firstAddressField; index < addrNumStringFields; index++)
      {
            if (GetBitMacro(flags, index) != 0)
            {
                  dest->fields[index] = p;
                  p += StrLen(p) + 1;
            }
            else
                  dest->fields[index] = NULL;
      }

      MemSet(&(dest->birthdayInfo), sizeof(BirthdayInfo), 0 );

      if(GetBitMacro(flags, birthdayDate))
      {
            MemMove(&(dest->birthdayInfo.birthdayDate), p, sizeof(DateType));
            p+= sizeof(DateType);
      }

      if(GetBitMacro(flags, birthdayMask))
      {
            MemMove(&(dest->birthdayInfo.birthdayMask), p, sizeof(AddressDBBirthdayFlags));
            p+= sizeof(AddressDBBirthdayFlags);
      }

      if(GetBitMacro(flags, birthdayPreset))
      {
            dest->birthdayInfo.birthdayPreset = *((UInt8*)p);
            p+= sizeof(UInt8);
      }

      dest->pictureInfo.pictureDirty = 0;
      dest->pictureInfo.pictureSize = 0;
      dest->pictureInfo.pictureData = NULL;
      dest->numBlobs = 0;

      MemSet(&(dest->rel2blobInfo), sizeof(Release2BlobType), 0 );
      dest->numBlobs = 0;


      blobStart = p;
      recordSize = MemPtrSize(src);

      while (blobStart < (Char *)src + recordSize)
      {
            p = blobStart;

            ErrNonFatalDisplayIf((Char *)src + recordSize - blobStart <= sizeof (blobId) +
                  sizeof (blobSize),"Invalid blob encountered");
            MemMove(&blobId, p, sizeof (blobId));
            p += sizeof (blobId);

            MemMove(&blobSize, p, sizeof (blobSize));
            p += sizeof (blobSize);

            blobEnd = p + blobSize;

            switch (blobId)
            {
                  case addrPictureBlobId:
                  {
                        UInt16  pictureDirtySize;

                        pictureDirtySize = sizeof(dest->pictureInfo.pictureDirty);

                        dest->pictureInfo.pictureSize = blobSize - pictureDirtySize;

                        MemMove(&(dest->pictureInfo.pictureDirty), p, pictureDirtySize);

                        p+= pictureDirtySize;

                        if(dest->pictureInfo.pictureSize)
                              dest->pictureInfo.pictureData = p;

                        p+= dest->pictureInfo.pictureSize;
                        break;
                  }

                  case addrRelease2BlobId:
                  {
                        MemMove(&dest->rel2blobInfo.anniversaryInfo, p, sizeof(BirthdayInfo));

                        p += sizeof(BirthdayInfo);

                        MemMove(&dest->rel2blobInfo.ringtoneInfo, p, sizeof(ToneIdentifier));

                        p += sizeof(ToneIdentifier);
                        break;
                  }


                  default:
                  {
                        ErrNonFatalDisplayIf(dest->numBlobs >= apptMaxBlobs, "Too many blobs");

                        dest->blobs[dest->numBlobs].creatorID = blobId;
                        dest->blobs[dest->numBlobs].size = blobSize;
                        dest->blobs[dest->numBlobs].content = p;

                        dest->numBlobs++;
                        p = blobEnd;
                        break;
                  }
            }

            ErrNonFatalDisplayIf(p != blobEnd, "Blob size does not agree with contents");

            blobStart = blobEnd;
      }

      ErrNonFatalDisplayIf(blobStart != (Char *)src + recordSize,
      "Last blob not aligned with end of record - don't let fields edit records
      directly!");
}
```

2.2. Calendar

This section describes the changes in the database structures for the Calendar application since Palm OS version 5.4 R3. It also illustrates the changes that must be made to use the code samples for packing, unpacking, and sorting records.

The enhanced database structures for the Calendar application are shown here:

```
typedef struct
{
      ApptDateTimeType      *when;
      AlarmInfoType         *alarm;
      RepeatInfoType        *repeat;
      ExceptionsListType    *exceptions;
      char                  *description;
      char                  *note;
      char                  *location;            // NEW
      ApptTimeZoneType      timeZone;             // NEW
      ApptMeetingInfo       meetingInfo;          // NEW
      UInt16                numBlobs;             // NEW
      BlobType              blobs[apptMaxBlobs];  // NEW
} ApptDBRecordType;
```

In addition to the inclusion of blobs in the record, new data such as `location`, `timeZone`, and `meetingInfo` are also introduced to the Calendar database structure.

- The `location` string provides information about the venue of the appointment.
- The `ApptTimeZoneType` structure enables the use of a time zone feature when setting up appointments. This controls Daylight Saving Time start and end dates.
- The `ApptMeetingInfo` structure contains information about whether a meeting is accepted or declined, number of attendees, etc.

It is important to note that the `timeZone` and `meetingInfo structures` are stored as blobs in the database with blob IDs `dateTimeZoneBlobId` and `dateMeetingBlobId` respectively. However, they are not counted towards the maximum number of blobs allowed (`apptMaxBlobs`).

2.2.1. Calendar - Packing Records Code Sample

```
static void ApptPack(ApptDBRecordPtr s, ApptPackedDBRecordPtr d)
{
      ApptDBRecordFlags      flags;
      UInt16                 size;
      ApptDateTimeType       when;
      UInt16                 len;
      UInt16                 index;
      UInt32                 offset = 0;
      UInt32                 offsetForSize;
      UInt32                 blobId;
      UInt16                 blobSize;
      UInt16                 blobCount = 0;

      *(UInt8 *)&flags = 0;

      offset = 0;

      when = *s->when;
      if (when.endTime.hours == hoursPerDay)
            when.endTime.hours = 0; // Never store hours = 24 in database.

      DmWrite(d, offset, &when, sizeof(ApptDateTimeType));
      offset += sizeof (ApptDateTimeType) + sizeof (ApptDBRecordFlags);

      if (s->alarm != NULL)
      {
            DmWrite(d, offset, s->alarm, sizeof(AlarmInfoType));
            offset += sizeof (AlarmInfoType);
            flags.alarm = 1;
      }

      if (s->repeat != NULL)
      {
            DmWrite(d, offset, s->repeat, sizeof(RepeatInfoType));
            offset += sizeof (RepeatInfoType);
            flags.repeat = 1;
      }

      if (s->exceptions != NULL)
      {
            size = sizeof (UInt16) + (s->exceptions->numExceptions * sizeof (DateType));
            DmWrite(d, offset, s->exceptions, size);
            offset += size;
            flags.exceptions = 1;
      }

      if (s->description != NULL)
      {
            size = StrLen(s->description) + 1;
            DmWrite(d, offset, s->description, size);
            offset += size;
            flags.description = 1;
      }

      if (s->note != NULL)
      {
            size = StrLen(s->note) + 1;
            DmWrite(d, offset, s->note, size);
            offset += size;
            flags.note = 1;
      }

      if (s->location != NULL)
      {
            size = StrLen(s->location) + 1;
            DmWrite(d, offset, s->location, size);
            offset += size;
            flags.location = 1;
      }

      DmWrite(d, sizeof(ApptDateTimeType), &flags, sizeof(flags));

      // Include a time zone blob if needed.
      if (s->timeZone.name[0] != chrNull)
      {
            blobId = dateTimeZoneBlobId;

            // Write the 4 byte blob ID.
            len = sizeof(blobId);
            DmWrite(d, offset, &blobId, len);
            offset += len;

            // Don't write the size yet, but remember where it goes.
            offsetForSize = offset;
            offset += sizeof(blobSize);

            // Write the blob content.
            DmWrite(d, offset, &s->timeZone.uTC, sizeof(s->timeZone.uTC));
            offset += sizeof(s->timeZone.uTC);

            DmWrite(d, offset, &s->timeZone.dSTStart, sizeof(s->timeZone.dSTStart));
            offset += sizeof(s->timeZone.dSTStart);

            DmWrite(d, offset, &s->timeZone.dSTEnd, sizeof(s->timeZone.dSTEnd));
            offset += sizeof(s->timeZone.dSTEnd);

            DmWrite(d, offset, &s->timeZone.dSTAdjustmentInMinutes,
                  sizeof(s->timeZone.dSTAdjustmentInMinutes));
            offset += sizeof(s->timeZone.dSTAdjustmentInMinutes);

            DmWrite(d, offset, &s->timeZone.country, sizeof(s->timeZone.country));
            offset += sizeof(s->timeZone.country);

            DmWrite(d, offset, (UInt8 *)&s->timeZone.name - 1 /*custom flag*/, sizeof(UInt8));
            offset += sizeof(UInt8); /*custom flag*/

            ErrNonFatalDisplayIf(StrLen(s->timeZone.name) > apptMaxTimeZoneNameLen – 1,
                  "Time zone name too long");

            DmWrite(d, offset, s->timeZone.name, StrLen(s->timeZone.name) + 1);
            offset += StrLen(s->timeZone.name) + 1;

            // Now go back and fill in the blob size.
            blobSize = offset - offsetForSize - sizeof(blobSize);
            DmWrite(d, offsetForSize, &blobSize, sizeof(blobSize));
            blobCount++;
      }


      // Include a meeting blob if needed, have any attendees or set the appt status to
      // something other than default
      if (s->meetingInfo.numAttendees != 0 || s->meetingInfo.apptStatus)
      {
            blobId = dateMeetingBlobId;

            // Write the 4 byte blob ID.
            len = sizeof(blobId);
            DmWrite(d, offset, &blobId, len);
            offset += len;

            // Don't write the size yet, but remember where it goes.
            offsetForSize = offset;
            offset += sizeof(blobSize);

            // Write the blob content.
            DmWrite(d, offset, &s->meetingInfo.meetingStatus,
                sizeof(s->meetingInfo.meetingStatus));
            offset += sizeof(s->meetingInfo.meetingStatus);

            DmWrite(d, offset, &s->meetingInfo.apptStatus, sizeof(s->meetingInfo.apptStatus));
            offset += sizeof(s->meetingInfo.apptStatus);

            DmWrite(d, offset, &s->meetingInfo.numAttendees,
                sizeof(s->meetingInfo.numAttendees));
            offset += sizeof(s->meetingInfo.numAttendees);

            for (index = 0; index < s->meetingInfo.numAttendees; index++)
            {
                DmWrite(d, offset, &s->meetingInfo.attendees[index].role,
                    sizeof(s->meetingInfo.attendees[index].role));
                offset += sizeof(s->meetingInfo.attendees[index].role);

                DmWrite(d, offset, s->meetingInfo.attendees[index].name,
                    StrLen(s->meetingInfo.attendees[index].name) + 1);
                offset += StrLen(s->meetingInfo.attendees[index].name) + 1;

                DmWrite(d, offset, s->meetingInfo.attendees[index].email,
                    StrLen(s->meetingInfo.attendees[index].email) + 1);
                offset += StrLen(s->meetingInfo.attendees[index].email) + 1;
            }

            // Now go back and fill in the blob size.
            blobSize = offset - offsetForSize - sizeof(blobSize);
            DmWrite(d, offsetForSize, &blobSize, sizeof(blobSize));
            blobCount++;
        }

        // Include any other blobs we don't understand.
        ErrNonFatalDisplayIf(blobCount + s->numBlobs > apptMaxBlobs, "Too many blobs");
        for (index = 0; index < s->numBlobs; index++)
        {
            size = sizeof(s->blobs[index].creatorID);
            DmWrite(d, offset, &(s->blobs[index].creatorID), size);
            offset += size;
            size = sizeof(s->blobs[index].size);
            DmWrite(d, offset, &(s->blobs[index].size), size);
            offset += size;
            DmWrite(d, offset, s->blobs[index].content, s->blobs[index].size);
            offset += s->blobs[index].size;
        }

        ErrNonFatalDisplayIf(offset > MemHandleSize(MemPtrRecoverHandle(d)),
            "Not enough room for packed record");

        ErrNonFatalDisplayIf(offset < MemHandleSize(MemPtrRecoverHandle(d)),
            "Too much room for packed record");
}
```

2.2.2. Calendar - Unpacking Record Code Sample

```
static void ApptUnpack(ApptPackedDBRecordPtr src, ApptDBRecordPtr dest)
{
      ApptDBRecordFlags flags;
      Char *p;
      Char *blobStart;
      Char *blobEnd;
      UInt16 recordSize;
      UInt32 blobId;
      UInt16 blobSize;
      UInt16 index;
      flags = src->flags;
      p = &src->firstField;

      recordSize = MemPtrSize(src);
      dest->when = (ApptDateTimeType *) src;

      ErrNonFatalDisplayIf(dest->when->endTime.hours == hoursPerDay,
            "Hours = 24 found in database");

      if (flags.alarm)
      {
            dest->alarm = (AlarmInfoType *) p;
            p += sizeof (AlarmInfoType);
      }
      else
            dest->alarm = NULL;


      if (flags.repeat)
      {
            dest->repeat = (RepeatInfoType *) p;
            p += sizeof (RepeatInfoType);
      }
      else
            dest->repeat = NULL;

      if (flags.exceptions)
      {
            dest->exceptions = (ExceptionsListType *) p;
            p += sizeof (UInt16) +
                  (((ExceptionsListType *) p)->numExceptions * sizeof (DateType));
      }
      else
            dest->exceptions = NULL;

      if (flags.description)
      {
            dest->description = p;
            p += StrLen(p) + 1;
      }
      else
            dest->description = NULL;


      if (flags.note)
      {
            dest->note = p;
            p += StrLen(p) + 1;
      }
      else
            dest->note = NULL;

      if (flags.location)
      {
            dest->location = p;
            p += StrLen(p) + 1;
```

```
}
else
     dest->location = NULL;

// There may also be blob data on the end of the record.
// First set everything as if there were no blobs.

// This indicates that there is no time zone info for this meeting.
dest->timeZone.name[0] = chrNull;

dest->meetingInfo.meetingStatus = unansweredMeeting;
dest->meetingInfo.apptStatus = showAsBusy;

// This indicates that it is not a meeting.
dest->meetingInfo.numAttendees = 0;

dest->numBlobs = 0; // Start by assuming no blobs we don't understand.


// Then iterate through the blobs, ignoring any we don't understand.
blobStart = p;   // First blob starts where last non-blob data ends.
while (blobStart < (Char *)src + recordSize)
{
     p = blobStart;

     // If a field is using edit in place to directly edit a database record at
     // the time this routine is called, or if the device was reset while an
     // edit in place was in progress, the record can be left with junk data on
     // the end. FldCompactText would clean up this junk, but it either wasn't
     // called or simply hasn't yet been called. We will attempt to parse this
     // junk data as blobs, but more than likely these blobs will appear to be
     // invalid. On release builds we want to recover gracefully from this
     // situation by ignoring the junk data.

     if ((Char *)src + recordSize - blobStart <= sizeof (blobId) + sizeof (blobSize))
     {
          ErrNonFatalDisplay("Blob goes beyond end of record - don't let fields edit
          records directly!");
          return;
     }

     MemMove(&blobId, p, sizeof (blobId));
     p += sizeof (blobId);
     MemMove(&blobSize, p, sizeof (blobSize));
     p += sizeof (blobSize);

     // Blob size excludes space to store ID and size of blob.
     blobEnd = p + blobSize;

     if (blobEnd > (Char *)src + recordSize)
     {
          ErrNonFatalDisplay("Blob goes beyond end of record - don't let fields
          edit records directly!");
          return;
     }

     switch (blobId)
     {
          case dateTimeZoneBlobId:

               ErrNonFatalDisplayIf(dest->timeZone.name[0] != chrNull,
                    "Duplicate time zone blob");

               MemMove(&dest->timeZone.uTC, p, sizeof (dest->timeZone.uTC));
               p += sizeof (dest->timeZone.uTC);

               MemMove(&dest->timeZone.dSTStart, p, sizeof (dest->timeZone.dSTStart));
               p += sizeof (dest->timeZone.dSTStart);

               MemMove(&dest->timeZone.dSTEnd, p, sizeof (dest->timeZone.dSTEnd));
```

```
                  p += sizeof (dest->timeZone.dSTEnd);

                  MemMove(&dest->timeZone.dSTAdjustmentInMinutes, p,
                        sizeof (dest->timeZone.dSTAdjustmentInMinutes));
                  p += sizeof (dest->timeZone.dSTAdjustmentInMinutes);

                  MemMove(&dest->timeZone.country, p, sizeof (dest->timeZone.country));
                  p += sizeof (dest->timeZone.country);

                  MemMove((Char *)&dest->timeZone.name - 1 /*custom flag*/, p,
                        sizeof (UInt8));
                  p += sizeof (UInt8);

                  ErrNonFatalDisplayIf(StrLen(p) > apptMaxTimeZoneNameLen – 1,
                        "Time zone name too long");
                  StrCopy(dest->timeZone.name, p);
                  p += StrLen(dest->timeZone.name) + 1;

                  break;

            case dateMeetingBlobId:
                  ErrNonFatalDisplayIf(dest->meetingInfo.numAttendees != 0,
                        "Duplicate meeting blob");

                  MemMove(&dest->meetingInfo.meetingStatus, p,
                        sizeof (dest->meetingInfo.meetingStatus));
                  p += sizeof (dest->meetingInfo.meetingStatus);

                  MemMove(&dest->meetingInfo.apptStatus, p,
                        sizeof (dest->meetingInfo.apptStatus));
                  p += sizeof (dest->meetingInfo.apptStatus);

                  MemMove(&dest->meetingInfo.numAttendees, p,
                        sizeof (dest->meetingInfo.numAttendees));
                  p += sizeof (dest->meetingInfo.numAttendees);

                  for (index = 0; index < dest->meetingInfo.numAttendees; index++)
                  {
                        MemMove(&dest->meetingInfo.attendees[index].role, p,
                              sizeof(dest->meetingInfo.attendees[index].role));
                        p += sizeof(dest->meetingInfo.attendees[index].role);

                        dest->meetingInfo.attendees[index].name = p;
                        p += StrLen(dest->meetingInfo.attendees[index].name) + 1;

                        dest->meetingInfo.attendees[index].email = p;
                        p += StrLen(dest->meetingInfo.attendees[index].email) + 1;

                        dest->meetingInfo.attendees[index].AttendeePos = index;
                  }
                  break;

            default:
            {
                  ErrNonFatalDisplayIf (dest->numBlobs >= apptMaxBlobs,"Too many blobs");

                  dest->blobs[dest->numBlobs].creatorID = blobId;
                  dest->blobs[dest->numBlobs].size = blobSize;
                  dest->blobs[dest->numBlobs].content = p;
                  dest->numBlobs++;
                  p = blobEnd;
                  break;
            }
      }

      ErrNonFatalDisplayIf(p != blobEnd, "Blob size does not agree with contents");
      blobStart = blobEnd;  // Next blob starts where last blob ends.
}

ErrNonFatalDisplayIf(blobStart != (Char *)src + recordSize, "Last blob not aligned with
      end of record – don't let fields edit records directly!");

      ErrNonFatalDisplayIf(ApptPackedSize(dest) != recordSize, "Blob size mismatch");
}
```

2.2.3. Calendar - Sorting Records Code Sample

With the changes to the PIM application database structures, the Calendar application has implemented a stricter sort order, which improves efficiency. The new sort function is as follows:

- Repeating items are sorted before non-repeating items in the database.
- Repeating items are sorted by their end date (oldest to newest).
- In the case where two items have the same repeat end date, the application compares start times.

```
static Int16 ApptComparePackedRecords (ApptPackedDBRecordPtr r1, ApptPackedDBRecordPtr r2,
      Int16 extra, SortRecordInfoPtr info1, SortRecordInfoPtr info2, MemHandle appInfoH)
{
#pragma unused (extra, info1, info2, appInfoH)

      Int16 result;

      if ((r1->flags.repeat) || (r2->flags.repeat))
      {
            if ((r1->flags.repeat) && (r2->flags.repeat))
            {
                  // In the past, two repeating events were considered equal. Now we
                  // sort them by their end date in order to more efficiently iterate
                  // over the repeating events on a given date or date range. First
                  // step is to find the repeat info in each of the records so we can
                  // compare their end dates. No end date is represented as -1, which
                  // will sort last, as desired.

                  result = DateCompare (ApptGetRepeatInfo(r1)->repeatEndDate,
                        ApptGetRepeatInfo(r2)->repeatEndDate);

                  if (result == 0)

                  // Two events than end on the same date are sorted by their
                  // start time. We don't in fact rely on this, but might in
                  // the future.
                  result = TimeCompare (r1->when.startTime, r2->when.startTime);
            }

            else if (r1->flags.repeat)
                  result = -1;
            else
                  result = 1;
      }
      else
      {
            result = DateCompare (r1->when.date, r2->when.date);
            if (result == 0)
                  result = TimeCompare (r1->when.startTime, r2->when.startTime);
      }
      return result;
}
```

2.3. Tasks

This section describes changes in the database structures for the Tasks application since Palm OS version 5.4 R3. It also illustrates the changes that must be made to use the code samples for updating records.

The enhanced database structures for the Tasks application are shown here:

```
typedef struct {
      ToDoDBDataFlags  dataFlags;       // NEW
      UInt16           recordFlags;     // NEW
      UInt16           priority;        // NEW
      char             optionalData[];  // NEW
} ToDoDBRecord;
```

In the enhanced ToDo (Tasks) application, in addition to due date, priority and description, each task is given its own completion date, alarm time, note, and recurrence information. These fields are stored in the variable length container that is pointed by `optionalData` in the structure. You may retrieve the fields based on the flag and by adjusting the offset.

As new data is added or deleted, the order of data or the offsets must be adjusted to reflect the changes. For example, if the task previously had a due date, `optionalData + 0(zero)` pointed to the due date and `optionalData + sizeof(DateType)` pointed to the completion date (if applicable). When the due date is removed, the data for completion date is moved forward and `optionalData + 0(zero)` will now point to it.

2.3.1. Tasks - Updating Record Code Sample

```
Err ToDoChangeRecord(DmOpenRef dbP, UInt16 *index, UInt16 filter, UInt16 subFilter,
      ToDoRecordFieldType changedField, const void * data)
{
      Char *                c;
      MemHandle             recordH = 0;
      ToDoDBRecordPtr       src;
      UInt32                offset;
      ToDoDBDataFlags       newFlags;
      Err                   err;
      Int16                 cLen;
      UInt16                attr;
      UInt16                curSize;
      UInt16                newSize;
      UInt16                descriptionOffset;
      UInt16                newIndex;
      UInt16                priority;
      UInt16                recordFlags;

      // Get the record which we are going to change
      recordH = DmQueryRecord(dbP, *index);
      src = MemHandleLock (recordH);

      newFlags = src->dataFlags;



      // If the record is being changed such that its sort position will
      // change, move the record to its new sort position.
      if (  (changedField == toDoRecordFieldCategory) ||
            (changedField == toDoRecordFieldPriority) ||
            (changedField == toDoRecordFieldDueDate)  ||
            (changedField == toDoRecordFieldCompletionDate) )
      {
            SortRecordInfoType    sortInfo;
            MemHandle             tempRecH = 0;
            ToDoDBRecordPtr       tempRecP;
            UInt32                tempRecSize;
            DateType              srcDueDate, srcCompletionDate;
            Boolean               newHasDueDate, newHasCompletionDate;

            MemSet( &sortInfo, sizeof( sortInfo ), 0 );
            DmRecordInfo( dbP, *index, &attr, NULL, NULL );
            sortInfo.attributes = attr;

            // we don't bother adding alarm info to the temporary new record,
            // since it doesn't matter to sorting.
            if ( toDoRecordFieldDueDate == changedField )
            {
                  newHasDueDate = (toDoNoDueDate != *( UInt16 * ) data);
            }
            else
            {
                  newHasDueDate = src->dataFlags.dueDate;
            }

            if ( toDoRecordFieldCompletionDate == changedField )
            {
                  newHasCompletionDate = (toDoNoCompletionDate != *( UInt16 * ) data);
            }
            else
            {
                  newHasCompletionDate = src->dataFlags.completionDate;
            }

            tempRecSize = sizeDBRecord;
            if ( newHasDueDate )
```

```
{
     tempRecSize += sizeof( DateType );
}
if ( newHasCompletionDate )
{
     tempRecSize += sizeof( DateType );
}
if ( src->dataFlags.repeat )
{
     tempRecSize += sizeof( ToDoRepeatInfoType );
}

tempRecH = MemHandleNew( tempRecSize );
if ( !tempRecH )
{
     goto exit;
}

tempRecP = ( ToDoDBRecordPtr ) MemHandleLock( tempRecH );
MemSet( tempRecP, tempRecSize, 0 );

// set data flags before the direct data manipulation below
tempRecP->dataFlags.dueDate            = newHasDueDate;
tempRecP->dataFlags.completionDate     = newHasCompletionDate;
tempRecP->dataFlags.repeat             = src->dataFlags.repeat;

if ( tempRecP->dataFlags.repeat && src->dataFlags.repeat )
{
     MemMove(ToDoDBRecordGetFieldPointer(tempRecP, toDoRecordFieldRepeat ),
          ToDoDBRecordGetFieldPointer(src, toDoRecordFieldRepeat ),
          sizeof( ToDoRepeatInfoType ) );
}

if ( src->dataFlags.dueDate )
{
     srcDueDate = *(DateType *)ToDoDBRecordGetFieldPointer(src,
          toDoRecordFieldDueDate );
}
else
{
     DateToInt( srcDueDate ) = toDoNoDueDate;
}

if ( src->dataFlags.completionDate )
{
     srcCompletionDate = *( DateType * ) ToDoDBRecordGetFieldPointer( src,
          toDoRecordFieldCompletionDate );
}
else
{
     DateToInt( srcCompletionDate ) = toDoNoCompletionDate;
}

if ( changedField == toDoRecordFieldCategory )
{
     tempRecP->priority    = src->priority;
     if ( newHasDueDate )
     {
          *( DateType * ) ToDoDBRecordGetFieldPointer( tempRecP,
               toDoRecordFieldDueDate ) = srcDueDate;
     }
     if ( newHasCompletionDate )
     {
          *( DateType * ) ToDoDBRecordGetFieldPointer( tempRecP,
               toDoRecordFieldCompletionDate ) = srcCompletionDate;
     }
     sortInfo.attributes = *( UInt16 * ) data;
}
else if ( changedField == toDoRecordFieldPriority )
{
```

```
            tempRecP->priority    = *( UInt16 * ) data;
            if ( newHasDueDate )
            {
                  *( DateType * ) ToDoDBRecordGetFieldPointer( tempRecP,
                        toDoRecordFieldDueDate ) = srcDueDate;
            }
            if ( newHasCompletionDate )
            {
                  *( DateType * ) ToDoDBRecordGetFieldPointer( tempRecP,
                        toDoRecordFieldCompletionDate ) = srcCompletionDate;
            }
            sortInfo.attributes = attr;
      }
      else if ( changedField == toDoRecordFieldDueDate )
      {
            tempRecP->priority    = src->priority;
            if ( newHasDueDate )
            {
                  *( DateType * ) ToDoDBRecordGetFieldPointer( tempRecP,
                        toDoRecordFieldDueDate ) = *(( DatePtr ) data);
            }
            if ( newHasCompletionDate )
            {
                  *( DateType * ) ToDoDBRecordGetFieldPointer( tempRecP,
                        toDoRecordFieldCompletionDate ) = srcCompletionDate;
            }
            sortInfo.attributes = attr;
      }
      else if ( changedField == toDoRecordFieldCompletionDate )
      {
            tempRecP->priority    = src->priority;
            if ( newHasDueDate )
            {
                  *( DateType * ) ToDoDBRecordGetFieldPointer( tempRecP,
                        toDoRecordFieldDueDate ) = srcDueDate;
            }
            if ( newHasCompletionDate )
            {
                  *( DateType * ) ToDoDBRecordGetFieldPointer( tempRecP,
                        toDoRecordFieldCompletionDate ) = *(( DatePtr )
                                                               data);
            }
            sortInfo.attributes = attr;
      }

      newIndex = ToDoFindSortPosition( dbP, tempRecP, &sortInfo, filter, subFilter );
      DmMoveRecord( dbP, *index, newIndex );

      if ( newIndex > *index )
      {
            newIndex--;
      }

      *index = newIndex;

      MemHandleUnlock( tempRecH );
      MemHandleFree( tempRecH );
}

if ( changedField == toDoRecordFieldCategory )
{
      attr = (attr & ~dmRecAttrCategoryMask) | *( UInt16 * ) data;

      DmSetRecordInfo( dbP, newIndex, &attr, NULL );

      goto exit;
}

if ( changedField == toDoRecordFieldPriority )
{
      priority = *( UInt16 * ) data;

      DmWrite( src, OffsetOf( ToDoDBRecord, priority ), &priority, sizeof( UInt16 ) );

      goto exit;
}


if ( changedField == toDoRecordFieldComplete )
{
      recordFlags = src->recordFlags;

      if ( *( UInt16 * ) data )
      {
            recordFlags |= TODO_RECORD_FLAG_COMPLETE;
      }
      else
      {
            recordFlags &= ~TODO_RECORD_FLAG_COMPLETE;
      }

      DmWrite(src, OffsetOf( ToDoDBRecord, recordFlags), &recordFlags, sizeof(UInt16));

      goto exit;
}

if ( toDoRecordFieldDueDate == changedField )
{
      MemPtrUnlock( src );

      if ( toDoNoDueDate == *( UInt16 * ) data )
      {
            ToDoDBRecordClearDueDate( dbP, *index );
      }
      else
      {
            ToDoDBRecordSetDueDate( dbP, *index, ( DateType * ) data );
      }

      goto exitNoUnlock;
}

if ( toDoRecordFieldCompletionDate == changedField )
{
      MemPtrUnlock( src );

      if ( toDoNoCompletionDate == *( UInt16 * ) data )
      {
            ToDoDBRecordClearCompletionDate( dbP, *index );
      }
      else
      {
            ToDoDBRecordSetCompletionDate( dbP, *index, ( DateType * ) data );
      }

      goto exitNoUnlock;
}

if ( toDoRecordFieldAlarm == changedField )
{
      MemPtrUnlock( src );

      ToDoDBRecordSetAlarmInfo( dbP, *index, ( ToDoAlarmInfoType * ) data );

      goto exitNoUnlock;
}

if ( toDoRecordFieldRepeat == changedField )
{
      goto exit;
```

```
}

// Calculate the size of the changed record. First,
// find the size of the data used from the old record.
newSize =  sizeof( ToDoDBDataFlags ) + // dataFlags
           sizeof( UInt16 ) +          // recordFlags
           sizeof( UInt16 );           // priority

offset = OffsetOf( ToDoDBRecord, optionalData );
c = ( char * ) &src->optionalData;

if ( src->dataFlags.dueDate )
{
     newSize += sizeof( DateType );
}

if ( src->dataFlags.completionDate )
{
     newSize += sizeof( DateType );
}

if ( src->dataFlags.alarm )
{
     newSize += sizeof( ToDoAlarmInfoType );
}

if ( src->dataFlags.repeat )
{
     newSize += sizeof( ToDoRepeatInfoType );
}

descriptionOffset = newSize;

// Now, add in the size of the new data
newSize += StrLen( ( Char * ) data ) + 1;

// Now, add in the size of whichever of the description and note will
// remain unchanged.
c = ( Char * ) src + descriptionOffset;
cLen = StrLen( c ) + 1;

if ( changedField != toDoRecordFieldDescription )
{
     newSize += cLen;
}

if ( changedField != toDoRecordFieldNote )
{
     c += cLen;
     newSize += StrLen( c ) + 1;
}

// Change the description field.
if ( changedField == toDoRecordFieldDescription )
{
     newFlags.description = (0 != StrLen( ( Char * ) data ));

     if ( newFlags.description != src->dataFlags.description )
     {
          DmWrite( src, OffsetOf( ToDoDBRecord, dataFlags ), &newFlags,
               sizeof( ToDoDBDataFlags ) );
     }

     // If the new description is longer, expand the record.
     curSize = MemPtrSize( src );
     if ( newSize > curSize )
     {
          MemPtrUnlock( src );
          err = MemHandleResize( recordH, newSize );
```

```
                if ( err )
                {
                     return err;
                }

                src = MemHandleLock( recordH );
           }

           // Move the note field.

           // Calculate new offset of note field
           offset = descriptionOffset + StrLen( ( Char * ) data ) + 1;

           // Point c at current note field
           c = ( Char * ) src + descriptionOffset;
           c += StrLen( c ) + 1;

           DmWrite( src, offset, c, StrLen( c ) + 1 );

           // Write the new description field.
           offset = descriptionOffset;
           DmStrCopy( src, offset, ( Char * ) data );

           // If the new description is shorter, shrink the record.
           if ( newSize < curSize )
           {
                MemHandleResize( recordH, newSize );
           }

           goto exit;
      }

      // Change the note field
      if ( changedField == toDoRecordFieldNote )
      {
           newFlags.note = (0 != StrLen( ( Char * ) data ));

           if ( newFlags.note != src->dataFlags.note )
           {
                DmWrite( src, OffsetOf( ToDoDBRecord, dataFlags ), &newFlags,
                     sizeof( ToDoDBDataFlags ) );
           }

           c = ( Char * ) src + descriptionOffset;
           offset = descriptionOffset + StrLen( c ) + 1;

           MemPtrUnlock( src );
           err = MemHandleResize( recordH, newSize );
           if ( err )
           {
                return err;
           }

           src = MemHandleLock( recordH );

           DmStrCopy( src, offset, data );

           goto exit;
      }

exit:
      MemPtrUnlock( src );

exitNoUnlock:

#if ERROR_CHECK_LEVEL == ERROR_CHECK_FULL
      ECToDoDBValidate( dbP );
#endif

      return 0;
```

```
}
```

2.4. Memos

There are no changes to the structure of the record in the Memo database, except for the name and creator ID.

2.5. Known Issues

2.5.1. Record Remainders

When an application opens a legacy PIM database, it is filled with records that are copied from the new databases. Then, when the legacy database is closed, data is removed, but 1-byte records are left behind.

2.5.2. DmDeleteRecord Causes Devices to Crash

An NVFS bug exists where when `DmDeleteRecord` is called after `DmAttachRecord`, a device will crash. This will cause a problem if an application is currently modifying the old PIM databases and relies on the compatibility layer to remove that record from the new databases.

To work around this issue, make sure your application looks for the unique ID of the record and delete the record from the new database directly. The unique ID of the record in both old and new databases should be the same.