src/com/android/providers/telephony/SmsProvider.java - platform/packages/providers/TelephonyProvider - Git at Google

Archived: 2026-04-05 17:15:12 UTC

/* * Copyright (C) 2006 The Android Open Source Project * * Licensed under the Apache License, Version 2.0 (the "License"); * you may not use this file except in compliance with the License. * You may obtain a copy of the License at * * http://www.apache.org/licenses/LICENSE-2.0 * * Unless required by applicable law or agreed to in writing, software * distributed under the License is distributed on an "AS IS" BASIS, * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. * See the License for the specific language governing permissions and * limitations under the License. */package com.android.providers.telephony;import android.content.ContentProvider;import android.content.ContentResolver;import android.content.ContentValues;import android.content.UriMatcher;import android.database.Cursor;import android.database.DatabaseUtils;import android.database.MatrixCursor;import android.database.sqlite.SQLiteDatabase;import android.database.sqlite.SQLiteOpenHelper;import android.database.sqlite.SQLiteQueryBuilder;import android.net.Uri;import android.provider.Contacts;import android.provider.Telephony;import android.provider.Telephony.Mms;import android.provider.Telephony.MmsSms;import android.provider.Telephony.Sms;import android.provider.Telephony.TextBasedSmsColumns;import android.provider.Telephony.Threads;import android.telephony.SmsManager;import android.telephony.SmsMessage;import android.text.TextUtils;import android.util.Log;import java.util.ArrayList;import java.util.HashMap;public class SmsProvider extends ContentProvider { private static final Uri NOTIFICATION_URI = Uri.parse("content://sms"); private static final Uri ICC_URI = Uri.parse("content://sms/icc"); static final String TABLE_SMS = "sms"; private static final String TABLE_RAW = "raw"; private static final String TABLE_SR_PENDING = "sr_pending"; private static final String TABLE_WORDS = "words"; private static final Integer ONE = Integer.valueOf(1); private static final String[] CONTACT_QUERY_PROJECTION = new String[] { Contacts.Phones.PERSON_ID }; private static final int PERSON_ID_COLUMN = 0; /** * These are the columns that are available when reading SMS * messages from the ICC. Columns whose names begin with "is_" * have either "true" or "false" as their values. */ private final static String[] ICC_COLUMNS = new String[] { // N.B.: These columns must appear in the same order as the // calls to add appear in convertIccToSms. "service_center_address", // getServiceCenterAddress "address", // getDisplayOriginatingAddress "message_class", // getMessageClass "body", // getDisplayMessageBody "date", // getTimestampMillis "status", // getStatusOnIcc "index_on_icc", // getIndexOnIcc "is_status_report", // isStatusReportMessage "transport_type", // Always "sms". "type", // Always MESSAGE_TYPE_ALL. "locked", // Always 0 (false). "error_code", // Always 0 "_id" }; @Override public boolean onCreate() { mOpenHelper = MmsSmsDatabaseHelper.getInstance(getContext()); return true; } @Override public Cursor query(Uri url, String[] projectionIn, String selection, String[] selectionArgs, String sort) { SQLiteQueryBuilder qb = new SQLiteQueryBuilder(); // Generate the body of the query. int match = sURLMatcher.match(url); switch (match) { case SMS_ALL: constructQueryForBox(qb, Sms.MESSAGE_TYPE_ALL); break; case SMS_UNDELIVERED: constructQueryForUndelivered(qb); break; case SMS_FAILED: constructQueryForBox(qb,

Sms.MESSAGE_TYPE_FAILED); break; case SMS_QUEUED: constructQueryForBox(qb, Sms.MESSAGE_TYPE_QUEUED); break; case SMS_INBOX: constructQueryForBox(qb, Sms.MESSAGE_TYPE_INBOX); break; case SMS_SENT: constructQueryForBox(qb, Sms.MESSAGE_TYPE_SENT); break; case SMS_DRAFT: constructQueryForBox(qb, Sms.MESSAGE_TYPE_DRAFT); break; case SMS_OUTBOX: constructQueryForBox(qb, Sms.MESSAGE_TYPE_OUTBOX); break; case SMS_ALL_ID: qb.setTables(TABLE_SMS); qb.appendWhere(" (_id = " + url.getPathSegments().get(0) + ")"); break; case SMS_INBOX_ID: case SMS_FAILED_ID: case SMS_SENT_ID: case SMS_DRAFT_ID: case SMS_OUTBOX_ID: qb.setTables(TABLE_SMS); qb.appendWhere("(_id = " + url.getPathSegments().get(1) + ")"); break; case SMS_CONVERSATIONS_ID: int threadID; try { threadID = Integer.parseInt(url.getPathSegments().get(1)); if (Log.isLoggable(TAG, Log.VERBOSE)) { Log.d(TAG, "query conversations: threadID=" + threadID); } } catch (Exception ex) { Log.e(TAG, "Bad conversation thread id: " + url.getPathSegments().get(1)); return null; } qb.setTables(TABLE_SMS); qb.appendWhere("thread_id = " + threadID); break; case SMS_CONVERSATIONS: qb.setTables("sms, (SELECT thread_id AS group_thread_id, MAX(date)AS group_date," + "COUNT(*) AS msg_count FROM sms GROUP BY thread_id) AS groups"); qb.appendWhere("sms.thread_id = groups.group_thread_id AND sms.date =" + "groups.group_date"); qb.setProjectionMap(sConversationProjectionMap); break; case SMS_RAW_MESSAGE: qb.setTables("raw"); break; case SMS_STATUS_PENDING: qb.setTables("sr_pending"); break; case SMS_ATTACHMENT: qb.setTables("attachments"); break; case SMS_ATTACHMENT_ID: qb.setTables("attachments"); qb.appendWhere("(sms_id = " + url.getPathSegments().get(1) + ")"); break; case SMS_QUERY_THREAD_ID: qb.setTables("canonical_addresses"); if (projectionIn == null) { projectionIn = sIDProjection; } break; case SMS_STATUS_ID: qb.setTables(TABLE_SMS); qb.appendWhere("(_id = " + url.getPathSegments().get(1) + ")"); break; case SMS_ALL_ICC: return getAllMessagesFromIcc(); case SMS_ICC: String messageIndexString = url.getPathSegments().get(1); return getSingleMessageFromIcc(messageIndexString); default: Log.e(TAG, "Invalid request: " + url); return null; } String orderBy = null; if (!TextUtils.isEmpty(sort)) { orderBy = sort; } else if (qb.getTables().equals(TABLE_SMS)) { orderBy = Sms.DEFAULT_SORT_ORDER; } SQLiteDatabase db = mOpenHelper.getReadableDatabase(); Cursor ret = qb.query(db, projectionIn, selection, selectionArgs, null, null, orderBy); // TODO: Since the URLs are a mess, always use content://sms ret.setNotificationUri(getContext().getContentResolver(), NOTIFICATION_URI); return ret; } private Object[] convertIccToSms(SmsMessage message, int id) { // N.B.: These calls must appear in the same order as the // columns appear in ICC_COLUMNS. Object[] row = new Object[13]; row[0] = message.getServiceCenterAddress(); row[1] = message.getDisplayOriginatingAddress(); row[2] = String.valueOf(message.getMessageClass()); row[3] = message.getDisplayMessageBody(); row[4] = message.getTimestampMillis(); row[5] = Sms.STATUS_NONE; row[6] = message.getIndexOnIcc(); row[7] = message.isStatusReportMessage(); row[8] = "sms"; row[9] = TextBasedSmsColumns.MESSAGE_TYPE_ALL; row[10] = 0; // locked row[11] = 0; // error_code row[12] = id; return row; } /** * Return a Cursor containing just one message from the ICC. */ private Cursor getSingleMessageFromIcc(String messageIndexString) { try { int messageIndex = Integer.parseInt(messageIndexString); SmsManager smsManager = SmsManager.getDefault(); ArrayList<SmsMessage> messages = smsManager.getAllMessagesFromIcc(); SmsMessage message = messages.get(messageIndex); if (message == null) { throw new IllegalArgumentException("Message not retrieved. ID: " + messageIndexString); } MatrixCursor cursor = new MatrixCursor(ICC_COLUMNS, 1); cursor.addRow(convertIccToSms(message, 0)); return withIccNotificationUri(cursor); } catch

(NumberFormatException exception) { throw new IllegalArgumentException("Bad SMS ICC ID: " + messageIndexString); } } /** * Return a Cursor listing all the messages stored on the ICC. */ private Cursor getAllMessagesFromIcc() { SmsManager smsManager = SmsManager.getDefault(); ArrayList<SmsMessage> messages = smsManager.getAllMessagesFromIcc(); final int count = messages.size(); MatrixCursor cursor = new MatrixCursor(ICC_COLUMNS, count); for (int i = 0; i < count; i++) { SmsMessage message = messages.get(i); if (message != null) { cursor.addRow(convertIccToSms(message, i)); } } return withIccNotificationUri(cursor); } private Cursor withIccNotificationUri(Cursor cursor) { cursor.setNotificationUri(getContext().getContentResolver(), ICC_URI); return cursor; } private void constructQueryForBox(SQLiteQueryBuilder qb, int type) { qb.setTables(TABLE_SMS); if (type != Sms.MESSAGE_TYPE_ALL) { qb.appendWhere("type=" + type); } } private void constructQueryForUndelivered(SQLiteQueryBuilder qb) { qb.setTables(TABLE_SMS); qb.appendWhere(" (type=" + Sms.MESSAGE_TYPE_OUTBOX + " OR type=" + Sms.MESSAGE_TYPE_FAILED + " OR type=" + Sms.MESSAGE_TYPE_QUEUED + ")"); } @Override public String getType(Uri url) { switch (url.getPathSegments().size()) { case 0: return VND_ANDROID_DIR_SMS; case 1: try { Integer.parseInt(url.getPathSegments().get(0)); return VND_ANDROID_SMS; } catch (NumberFormatException ex) { return VND_ANDROID_DIR_SMS; } case 2: // TODO: What about "threadID"? if (url.getPathSegments().get(0).equals("conversations")) { return VND_ANDROID_SMSCHAT; } else { return VND_ANDROID_SMS; } } return null; } @Override public Uri insert(Uri url, ContentValues initialValues) { ContentValues values; long rowID; int type = Sms.MESSAGE_TYPE_ALL; int match = sURLMatcher.match(url); String table = TABLE_SMS; switch (match) { case SMS_ALL: Integer typeObj = initialValues.getAsInteger(Sms.TYPE); if (typeObj != null) { type = typeObj.intValue(); } else { // default to inbox type = Sms.MESSAGE_TYPE_INBOX; } break; case SMS_INBOX: type = Sms.MESSAGE_TYPE_INBOX; break; case SMS_FAILED: type = Sms.MESSAGE_TYPE_FAILED; break; case SMS_QUEUED: type = Sms.MESSAGE_TYPE_QUEUED; break; case SMS_SENT: type = Sms.MESSAGE_TYPE_SENT; break; case SMS_DRAFT: type = Sms.MESSAGE_TYPE_DRAFT; break; case SMS_OUTBOX: type = Sms.MESSAGE_TYPE_OUTBOX; break; case SMS_RAW_MESSAGE: table = "raw"; break; case SMS_STATUS_PENDING: table = "sr_pending"; break; case SMS_ATTACHMENT: table = "attachments"; break; case SMS_NEW_THREAD_ID: table = "canonical_addresses"; break; default: Log.e(TAG, "Invalid request: " + url); return null; } SQLiteDatabase db = mOpenHelper.getWritableDatabase(); if (table.equals(TABLE_SMS)) { boolean addDate = false; boolean addType = false; // Make sure that the date and type are set if (initialValues == null) { values = new ContentValues(1); addDate = true; addType = true; } else { values = new ContentValues(initialValues); if (!initialValues.containsKey(Sms.DATE)) { addDate = true; } if (!initialValues.containsKey(Sms.TYPE)) { addType = true; } } if (addDate) { values.put(Sms.DATE, new Long(System.currentTimeMillis())); } if (addType && (type != Sms.MESSAGE_TYPE_ALL)) { values.put(Sms.TYPE, Integer.valueOf(type)); } // thread_id Long threadId = values.getAsLong(Sms.THREAD_ID); String address = values.getAsString(Sms.ADDRESS); if (((threadId == null) || (threadId == 0)) && (!TextUtils.isEmpty(address))) { values.put(Sms.THREAD_ID, Threads.getOrCreateThreadId(getContext(), address)); } // If this message is going in as a draft, it should replace any // other draft messages in the thread. Just delete all draft // messages with this thread ID. We could add an OR REPLACE to // the insert below, but we'd have to query to find the old _id // to produce a conflict anyway. if (values.getAsInteger(Sms.TYPE) == Sms.MESSAGE_TYPE_DRAFT) { db.delete(TABLE_SMS, "thread_id=? AND type=?", new String[] { values.getAsString(Sms.THREAD_ID),

Integer.toString(Sms.MESSAGE_TYPE_DRAFT) }); } if (type == Sms.MESSAGE_TYPE_INBOX) { // Look up the person if not already filled in. if ((values.getAsLong(Sms.PERSON) == null) && (!TextUtils.isEmpty(address))) { Cursor cursor = null; Uri uri = Uri.withAppendedPath(Contacts.Phones.CONTENT_FILTER_URL, Uri.encode(address)); try { cursor = getContext().getContentResolver().query(uri, CONTACT_QUERY_PROJECTION, null, null, null); if (cursor.moveToFirst()) { Long id = Long.valueOf(cursor.getLong(PERSON_ID_COLUMN)); values.put(Sms.PERSON, id); } } catch (Exception ex) { Log.e(TAG, "insert: query contact uri " + uri + " caught ", ex); } finally { if (cursor != null) { cursor.close(); } } } } else { // Mark all non-inbox messages read. values.put(Sms.READ, ONE); } } else { if (initialValues == null) { values = new ContentValues(1); } else { values = initialValues; } } rowID = db.insert(table, "body", values); // Don't use a trigger for updating the words table because of a bug // in FTS3. The bug is such that the call to get the last inserted // row is incorrect. if (table == TABLE_SMS) { // Update the words table with a corresponding row. The words table // allows us to search for words quickly, without scanning the whole // table; ContentValues cv = new ContentValues(); cv.put(Telephony.MmsSms.WordsTable.ID, rowID); cv.put(Telephony.MmsSms.WordsTable.INDEXED_TEXT, values.getAsString("body")); cv.put(Telephony.MmsSms.WordsTable.SOURCE_ROW_ID, rowID); cv.put(Telephony.MmsSms.WordsTable.TABLE_ID, 1); db.insert(TABLE_WORDS, Telephony.MmsSms.WordsTable.INDEXED_TEXT, cv); } if (rowID > 0) { Uri uri = Uri.parse("content://" + table + "/" + rowID); if (Log.isLoggable(TAG, Log.VERBOSE)) { Log.d(TAG, "insert " + uri + " succeeded"); } notifyChange(uri); return uri; } else { Log.e(TAG,"insert: failed! " + values.toString()); } return null; } @Override public int delete(Uri url, String where, String[] whereArgs) { int count; int match = sURLMatcher.match(url); SQLiteDatabase db = mOpenHelper.getWritableDatabase(); switch (match) { case SMS_ALL: count = db.delete(TABLE_SMS, where, whereArgs); if (count != 0) { // Don't update threads unless something changed. MmsSmsDatabaseHelper.updateAllThreads(db, where, whereArgs); } break; case SMS_ALL_ID: try { int message_id = Integer.parseInt(url.getPathSegments().get(0)); count = MmsSmsDatabaseHelper.deleteOneSms(db, message_id); } catch (Exception e) { throw new IllegalArgumentException("Bad message id: " + url.getPathSegments().get(0)); } break; case SMS_CONVERSATIONS_ID: int threadID; try { threadID = Integer.parseInt(url.getPathSegments().get(1)); } catch (Exception ex) { throw new IllegalArgumentException("Bad conversation thread id: " + url.getPathSegments().get(1)); } // delete the messages from the sms table where = DatabaseUtils.concatenateWhere("thread_id=" + threadID, where); count = db.delete(TABLE_SMS, where, whereArgs); MmsSmsDatabaseHelper.updateThread(db, threadID); break; case SMS_RAW_MESSAGE: count = db.delete("raw", where, whereArgs); break; case SMS_STATUS_PENDING: count = db.delete("sr_pending", where, whereArgs); break; case SMS_ICC: String messageIndexString = url.getPathSegments().get(1); return deleteMessageFromIcc(messageIndexString); default: throw new IllegalArgumentException("Unknown URL"); } if (count > 0) { notifyChange(url); } return count; } /** * Delete the message at index from ICC. Return true iff * successful. */ private int deleteMessageFromIcc(String messageIndexString) { SmsManager smsManager = SmsManager.getDefault(); try { return smsManager.deleteMessageFromIcc(Integer.parseInt(messageIndexString)) ? 1 : 0; } catch (NumberFormatException exception) { throw new IllegalArgumentException("Bad SMS ICC ID: " + messageIndexString); } finally { ContentResolver cr = getContext().getContentResolver(); cr.notifyChange(ICC_URI, null); } } @Override public int update(Uri url, ContentValues values, String where, String[] whereArgs) { int count = 0; String table = TABLE_SMS; String extraWhere = null; SQLiteDatabase db = mOpenHelper.getWritableDatabase(); switch (sURLMatcher.match(url)) { case SMS_RAW_MESSAGE: table = TABLE_RAW; break; case SMS_STATUS_PENDING: table =

TABLE_SR_PENDING; break; case SMS_ALL: case SMS_FAILED: case SMS_QUEUED: case SMS_INBOX: case SMS_SENT: case SMS_DRAFT: case SMS_OUTBOX: case SMS_CONVERSATIONS: break; case SMS_ALL_ID: extraWhere = "_id=" + url.getPathSegments().get(0); break; case SMS_INBOX_ID: case SMS_FAILED_ID: case SMS_SENT_ID: case SMS_DRAFT_ID: case SMS_OUTBOX_ID: extraWhere = "_id=" + url.getPathSegments().get(1); break; case SMS_CONVERSATIONS_ID: { String threadId = url.getPathSegments().get(1); try { Integer.parseInt(threadId); } catch (Exception ex) { Log.e(TAG, "Bad conversation thread id: " + threadId); break; } extraWhere = "thread_id=" + threadId; break; } case SMS_STATUS_ID: extraWhere = "_id=" + url.getPathSegments().get(1); break; default: throw new UnsupportedOperationException("URI " + url + " not supported"); } where = DatabaseUtils.concatenateWhere(where, extraWhere); count = db.update(table, values, where, whereArgs); if (count > 0) { if (Log.isLoggable(TAG, Log.VERBOSE)) { Log.d(TAG, "update " + url + " succeeded"); } notifyChange(url); } return count; } private void notifyChange(Uri uri) { ContentResolver cr = getContext().getContentResolver(); cr.notifyChange(uri, null); cr.notifyChange(MmsSms.CONTENT_URI, null); cr.notifyChange(Uri.parse("content://mms-sms/conversations/"), null); } private SQLiteOpenHelper mOpenHelper; private final static String TAG = "SmsProvider"; private final static String VND_ANDROID_SMS = "vnd.android.cursor.item/sms"; private final static String VND_ANDROID_SMSCHAT = "vnd.android.cursor.item/sms-chat"; private final static String VND_ANDROID_DIR_SMS = "vnd.android.cursor.dir/sms"; private static final HashMap<String, String> sConversationProjectionMap = new HashMap<String, String>(); private static final String[] sIDProjection = new String[] { "_id" }; private static final int SMS_ALL = 0; private static final int SMS_ALL_ID = 1; private static final int SMS_INBOX = 2; private static final int SMS_INBOX_ID = 3; private static final int SMS_SENT = 4; private static final int SMS_SENT_ID = 5; private static final int SMS_DRAFT = 6; private static final int SMS_DRAFT_ID = 7; private static final int SMS_OUTBOX = 8; private static final int SMS_OUTBOX_ID = 9; private static final int SMS_CONVERSATIONS = 10; private static final int SMS_CONVERSATIONS_ID = 11; private static final int SMS_RAW_MESSAGE = 15; private static final int SMS_ATTACHMENT = 16; private static final int SMS_ATTACHMENT_ID = 17; private static final int SMS_NEW_THREAD_ID = 18; private static final int SMS_QUERY_THREAD_ID = 19; private static final int SMS_STATUS_ID = 20; private static final int SMS_STATUS_PENDING = 21; private static final int SMS_ALL_ICC = 22; private static final int SMS_ICC = 23; private static final int SMS_FAILED = 24; private static final int SMS_FAILED_ID = 25; private static final int SMS_QUEUED = 26; private static final int SMS_UNDELIVERED = 27; private static final UriMatcher sURLMatcher = new UriMatcher(UriMatcher.NO_MATCH); static { sURLMatcher.addURI("sms", null, SMS_ALL); sURLMatcher.addURI("sms", "#", SMS_ALL_ID); sURLMatcher.addURI("sms", "inbox", SMS_INBOX); sURLMatcher.addURI("sms", "inbox/#", SMS_INBOX_ID); sURLMatcher.addURI("sms", "sent", SMS_SENT); sURLMatcher.addURI("sms", "sent/#", SMS_SENT_ID); sURLMatcher.addURI("sms", "draft", SMS_DRAFT); sURLMatcher.addURI("sms", "draft/#", SMS_DRAFT_ID); sURLMatcher.addURI("sms", "outbox", SMS_OUTBOX); sURLMatcher.addURI("sms", "outbox/#", SMS_OUTBOX_ID); sURLMatcher.addURI("sms", "undelivered", SMS_UNDELIVERED); sURLMatcher.addURI("sms", "failed", SMS_FAILED); sURLMatcher.addURI("sms", "failed/#", SMS_FAILED_ID); sURLMatcher.addURI("sms", "queued", SMS_QUEUED); sURLMatcher.addURI("sms", "conversations", SMS_CONVERSATIONS); sURLMatcher.addURI("sms", "conversations/*", SMS_CONVERSATIONS_ID); sURLMatcher.addURI("sms", "raw", SMS_RAW_MESSAGE); sURLMatcher.addURI("sms", "attachments", SMS_ATTACHMENT); sURLMatcher.addURI("sms",

"attachments/#", SMS_ATTACHMENT_ID); sURLMatcher.addURI("sms", "threadID", SMS_NEW_THREAD_ID); sURLMatcher.addURI("sms", "threadID/*", SMS_QUERY_THREAD_ID); sURLMatcher.addURI("sms", "status/#", SMS_STATUS_ID); sURLMatcher.addURI("sms", "sr_pending", SMS_STATUS_PENDING); sURLMatcher.addURI("sms", "icc", SMS_ALL_ICC); sURLMatcher.addURI("sms", "icc/#", SMS_ICC); //we keep these for not breaking old applications sURLMatcher.addURI("sms", "sim", SMS_ALL_ICC); sURLMatcher.addURI("sms", "sim/#", SMS_ICC); sConversationProjectionMap.put(Sms.Conversations.SNIPPET, "sms.body AS snippet"); sConversationProjectionMap.put(Sms.Conversations.THREAD_ID, "sms.thread_id AS thread_id"); sConversationProjectionMap.put(Sms.Conversations.MESSAGE_COUNT, "groups.msg_count AS msg_count"); sConversationProjectionMap.put("delta", null); }}
