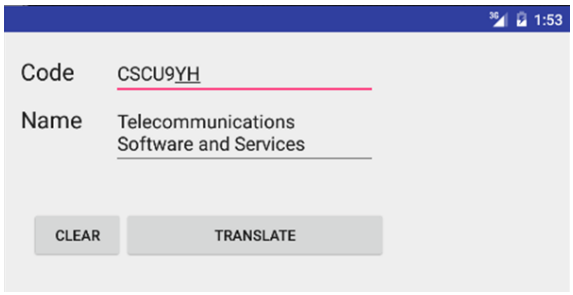
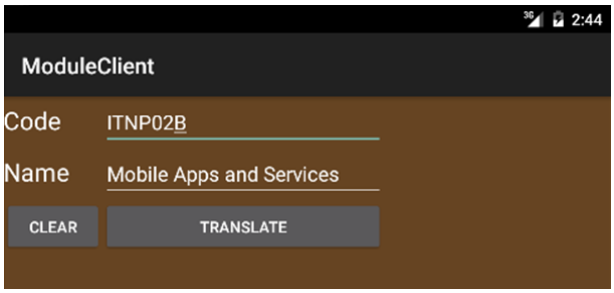
University of Stirling
Computing Science
Mobile App Development

Android Practical 3: Module Database

Checkpoint at the End

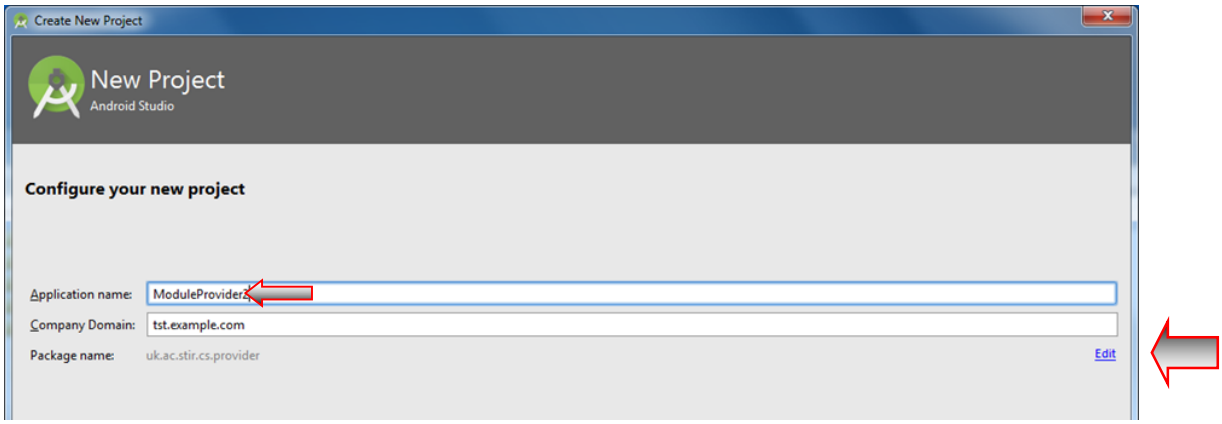
In this practical you will learn about Android content providers. The aim is to create a content provider that translates module codes into names (e.g. CSCU9YE is translated into 'Artificial Intelligence'). You will also create a client activity for this provider.



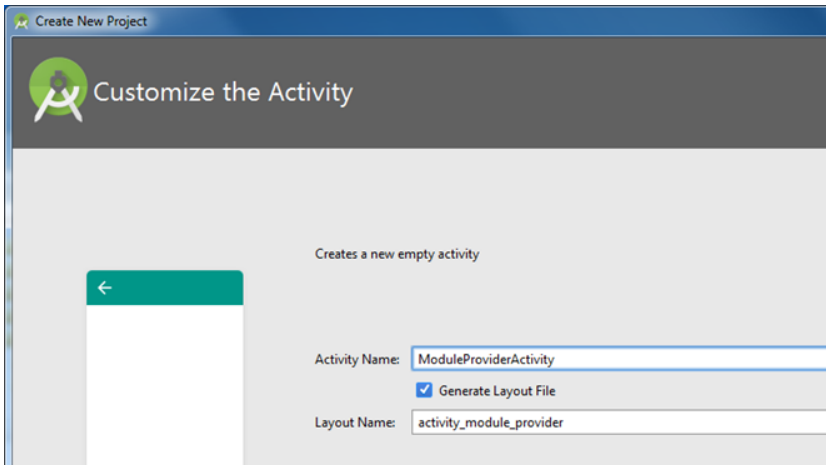


Create the *Module Provider Activity*

To do this, create an Android project much as you did before, except that you should ensure the package name is *uk.ac.stir.cs.provider*. When you create the new project you will need to edit the package name.



Call your project *ModuleProvider*. On the next window, name the activity *ModuleProviderActivity.java*.



Then using the Explorer on the left hand pane open *res\values\strings.xml* and edit *hello* to say 'Module database initialised' as the message to be used once the activity has set up the database. Then open res\layout\activity_module_provider and edit the *text* within *TextView* to 'Module database initialised'.

In the code that follows below, the *onCreate* method sets up the database by calling the *addModule* method for each module. Add whatever modules you wish here.

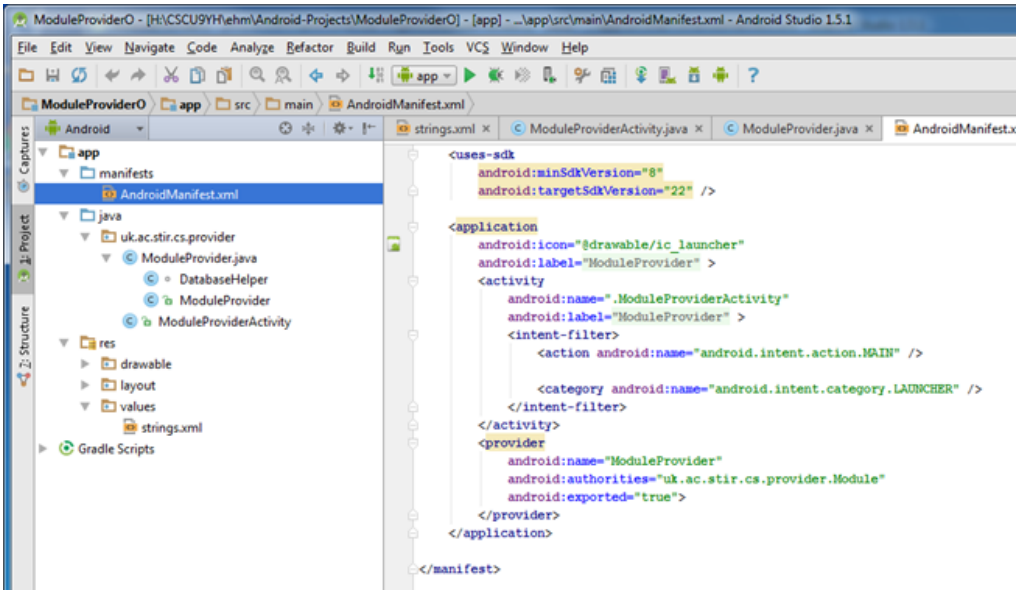
Notice the use of method *Log.v* (short for verbose) which is called to log each module as it is added. This is a useful for debugging Android code (much like inserting print statements in a Java program). The log output appears under the *Android Monitor* tab. This should be visible, alternatively use *ALT+6*.

```
package uk.ac.stir.cs.provider;

import android.app.Activity;                    // import activity
import android.os.Bundle;                       // import bundle
import android.content.ContentResolver;         // import content resolver
import android.content.ContentValues;           // import content values
import android.net.Uri;                         // import URI
import android.util.Log;                        // import log


public class ModuleProviderActivity extends Activity {

  private ContentResolver contentResolver;

  /**
    Add module to database. If updating the database table does not work
    (because the module does not exist), simply insert it into the table.

    @param code           module code
    @param name           module name
  */
  public void addModule(String code, String name) {
    Log.v("adding module", "code " + code + " name " + name);
    ContentValues values = new ContentValues();
    Uri uri = Uri.withAppendedPath(ModuleProvider.CONTENT_URI, code);
    values.put("code", code);
    values.put("name", name);
    int rows = contentResolver.update(uri, values, "", null);
    if (rows == 0) {
      uri = contentResolver.insert(uri, values);
    }
  }
```

```
  /**
    Set up modules in database and report completion.

    @param savedInstanceState    saved instance state
  */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    contentResolver = getContentResolver();

    // delete any existing module table
    contentResolver.delete(ModuleProvider.CONTENT_URI, null, null);

    // call addModule("Code", "Name") for each module to be added

     setContentView(R.layout.activity_module_provider); // report completion
  }
}
```

Create the Module Provider

Open *AndroidManifest.xml* and add the provider tab as shown below. This edit can be speeded up by clicking just under *activity* end tab, then *right click> generate…>XML Tag* and select *provider.* Set the *Name* to *ModuleProvider* and *Authorities* to *uk.ac.stir.cs.provider.Module* (the name by which the provider will be known). Then save the manifest file ignoring any error, e.g. *ModuleProvider* not resolved. Such errors will disappear when the provider class is defined later.



In the Explorer, select *java\uk.ac.stir.cs.provider* and right-click *New* > *Class* and set the *Name* to *ModuleProvider*. Open *ModuleProvider.java* from the Explorer and put in the code given below.

ModuleProvider extends the standard *ContentProvider*. You will see that there are constants for the database table name, the provider name, the URI for accessing this, and URI codes corresponding to all modules or just one module. The *onCreate* method calls *getContentResolver* for access to provider content and also creates a database reference.

It is necessary to provide implementations for the *delete*, *getType*, *insert* and *update* database methods. These provide the interface between client apps and the content provider. A separate *DatabaseHelper* class provides an interface to the built-in *SQLite* database. For each content provider URI, the *getType*

method returns the resulting MIME type (Multipurpose Internet Mail Extensions). This names the type of data returned by the content provider when asked for module details; *vnd* stands for vendor (i.e. developer-defined). The *onCreate* method creates a content resolver (for accessing content) and creates a reference to the database.

Once the content provider has been created, run *ModuleProvider* from Android Studio and check that it reports initialisation of the database.

```
package uk.ac.stir.cs.provider;

import android.content.ContentUris;           // import content URI
import android.content.Context;               // import context
import android.content.ContentResolver;       // import content resolver
import android.content.ContentProvider;       // import content provider
import android.content.ContentValues;         // import content values
import android.content.UriMatcher;            // import URI matcher
import android.database.Cursor;               // import database cursor
import android.database.SQLException;         // import SQL exception
import android.database.sqlite.SQLiteDatabase; // import SQLite database
import android.database.sqlite.SQLiteOpenHelper;// import SQLite helper
import android.database.sqlite.SQLiteQueryBuilder; // import SQLite query
import android.net.Uri;                       // import URI
import android.text.TextUtils;                // import text utilities

public class ModuleProvider extends ContentProvider {

  /* ------------------------------ Constants ------------------------------ */

  /** Database modules table */
  public final static String DATABASE_TABLE = "modules";

  /** Content provider name */
  public final static String PROVIDER_NAME = "uk.ac.stir.cs.provider.Module";

  /** Content provider URI */
  public final static Uri CONTENT_URI =
    Uri.parse("content://" + PROVIDER_NAME + "/" + DATABASE_TABLE);

  /** URI code for all modules */
  private final static int ALL_MODULES = 1;

  /** URI code for one module */
  private final static int ONE_MODULE = 2;

  /* ------------------------------ Variables ------------------------------ */

    private ContentResolver contentResolver;
    private SQLiteDatabase modulesDatabase;
    private UriMatcher uriMatcher; /** Content provider name */

  /* ------------------------------ Methods ------------------------------ */

  /**
    Delete a selection from the module table.

    @param uri            URI for deletion
    @param selection      optional filter on rows to be deleted
    @param arguments      selection arguments (argument replaces '?' in selection)
    @return               number of rows deleted
  */
  @Override
  public int delete(Uri uri, String selection, String[] arguments) {
```

```
  int count = 0;
  switch (uriMatcher.match(uri)) {
    case ALL_MODULES:
      count = modulesDatabase.delete(DATABASE_TABLE, selection, arguments);
      break;
    case ONE_MODULE:
      String code = uri.getPathSegments().get(1);
      code = "code = '" + code + "'";
      if (!TextUtils.isEmpty(selection))
        code += " AND (" + selection + ")";
      count = modulesDatabase.delete(DATABASE_TABLE, code, arguments);
      break;
    default:
     throw(new IllegalArgumentException("Unknown URI '" + uri + "'"));
  }
  contentResolver.notifyChange(uri, null);
  return(count);
}

/**
  Return MIME type for a module query.

  @param uri            URI for querying
  @return               MIME return type
*/
@Override
public String getType(Uri uri) {
  String mimeType;
  switch (uriMatcher.match(uri)) {
    case ALL_MODULES:
     mimeType = "vnd.uk.ac.stir.cs.cursor.dir/modules";
     break;
    case ONE_MODULE:
     mimeType = "vnd.uk.ac.stir.cs.cursor.item/modules";
     break;
    default:
     throw(new IllegalArgumentException("Invalid URI '" + uri + "'"));
  }
  return(mimeType);
}

/**
  Insert values into the module table.

  @param uri            URI for insertion
  @param values         column name/value pairs to be inserted
  @return               URI of the newly inserted items
*/
@Override
public Uri insert(Uri uri, ContentValues values) {
  Uri newUri;
  long rowIdentifier = modulesDatabase.insert(DATABASE_TABLE, "", values);
  if (rowIdentifier > 0) {
    newUri = ContentUris.withAppendedId(CONTENT_URI, rowIdentifier);
    contentResolver.notifyChange(newUri, null);
  }
  else
    throw(new SQLException("Failed to insert row into '" + uri + "'"));
  return(newUri);
}

/**
  Create modules database.
```

```
  @return               true if database was created successfully
*/
@Override
public boolean onCreate() {
  uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
  uriMatcher.addURI(PROVIDER_NAME, "modules", ALL_MODULES);
  uriMatcher.addURI(PROVIDER_NAME, "modules/*", ONE_MODULE);

  Context context = getContext();
  contentResolver = context.getContentResolver();
  DatabaseHelper databaseHelper = new DatabaseHelper(context);
  modulesDatabase = databaseHelper.getWritableDatabase();
  boolean result = modulesDatabase != null;
  return(result);
}

/**
  Query database.

  @param uri            URI for querying
  @param projection     list of columns for database cursor (null = all columns)
  @param selection      optional filter on rows to be queried (null = all rows)
  @param arguments      selection arguments (argument replaces '?' in selection)
  @param sortOrder      sorting order (null or empty = sort by code)
  @return               database cursor position
*/
@Override
public Cursor query(Uri uri, String[] projection, String selection,
 String[] arguments, String sortOrder) {
  SQLiteQueryBuilder sqlBuilder = new SQLiteQueryBuilder();
  sqlBuilder.setTables(DATABASE_TABLE);
  if (uriMatcher.match(uri) == ONE_MODULE) {
    String code = uri.getPathSegments().get(1);
    sqlBuilder.appendWhere("code = '" + code + "'");
  }
  if (sortOrder == null || sortOrder == "") {
    sortOrder = "code";
  }

  Cursor cursor = sqlBuilder.query(modulesDatabase, projection, selection,
    arguments, null, null, sortOrder);
  cursor.setNotificationUri(contentResolver, uri);
  return(cursor);
}

/**
  Update database.

  @param uri            URI for updating
  @param values         mapping from column names to values
  @param selection      optional filter on rows to be updated
  @param arguments      selection arguments (argument replaces '?' in selection)
  @return               number of updated rows
*/
@Override
public int update(Uri uri, ContentValues values, String selection,
 String[] arguments) {
  int count = 0;
  switch (uriMatcher.match(uri)) {
    case ALL_MODULES:
     count = modulesDatabase.update(DATABASE_TABLE, values, selection,
       arguments);
```

```
        break;
      case ONE_MODULE:
        String code = uri.getPathSegments().get(1);
        code = "code = '" + code + "'";
        if (!TextUtils.isEmpty(selection))
          code += " AND (" + selection + ")";
        count = modulesDatabase.update(DATABASE_TABLE, values, code,
          arguments);
        break;
      default:
       throw(new IllegalArgumentException("Unknown URI " + uri));
    }
    contentResolver.notifyChange(uri, null);
    return(count);
  }
}

class DatabaseHelper extends SQLiteOpenHelper {

  private final static String DATABASE_NAME = "courses";
  private final static int DATABASE_VERSION = 1;

  /**
    Database helper class.

    @param context  context
  */
  DatabaseHelper(Context context) {
    super(context, DATABASE_NAME, null, DATABASE_VERSION);
  }

  /**
    Handle database creation.

    @param database database
  */
  @Override
  public void onCreate(SQLiteDatabase database) {
    database.execSQL(
      "CREATE TABLE " + ModuleProvider.DATABASE_TABLE + "(" +
        "id INTEGER PRIMARY KEY AUTOINCREMENT, " +
       "code TEXT NOT NULL," +
       "name TEXT NOT NULL" +
      ");"
    );
  }

  /**
    Handle database upgrade.

    @param database       database
    @param oldVersion     old version number
    @param newVersion     new version number
  */
  @Override
  public void onUpgrade(SQLiteDatabase database, int oldVersion,
   int newVersion) {
    database.execSQL("DROP TABLE IF EXISTS " + ModuleProvider.DATABASE_TABLE);
    onCreate(database);
  }
}
```

Create the Module Client

Create an Android project much as you did before, with package name *uk.ac.stir.cs.android*. Call your project *ModuleClient*. Name the activity *ModuleClientActivity.java*. Then using the Explorer edit the following:

Open *res\layout\activity_module_client.xml* (or whatever your layout file is called) and edit this to look like the screenshot at the beginning of the practical. The layout is similar to the second practical, with the following rows in the table layout:

- In the first *TableRow* on the left hand side place *Plain Text* and again following the procedure in the second lab, set the name to *codeLabel* and the value to *Code*. In the next column place *Plain Text* and name it *codeValue* and give it a value (width) of *250dp*.
- In the second *TableRow* on the left hand side place *Plain Text* and set the name to *nameLabel* and the value to *Name*. In the next column place *textMultiLine* and (as before through the *width* property) name it *nameValue* and give it a value (width) of *250dp*.
- In the third row place a *Button* and name it *clearButton* with value '*Clear'*; and then place a second *Button* called *translateButton* with value '*Translate'*.

When the Translate button is clicked, the content provider is asked to translate the given module code into its name. A managed query is used as this keeps track of the database cursor automatically. If the module name is missing or invalid (one row is not returned in the cursor), and an error is displayed. Run your code and check that it behaves as expected (including error cases such as no module code or an invalid one).

```
package uk.ac.stir.cs.android;

import android.app.Activity;                    // import activity
import android.database.Cursor;                 // import database cursor
import android.net.Uri;                         // import URIs
import android.os.Bundle;                       // import bundle
import android.view.View;                       // import view
import android.view.View.OnClickListener;       // import click listener
import android.widget.Button;                   // import button
import android.widget.EditText;                 // import edit text
import android.widget.Toast;                    // import toast

public class ModuleClientActivity extends Activity {

  /* ------------------------------ Constants ------------------------------ */

  /** Database modules table */
  public final static String DATABASE_TABLE = "modules";

  /** Content provider name */
  public final static String PROVIDER_NAME = "uk.ac.stir.cs.provider.Module";

  /** Content provider URI */
  public final static Uri CONTENT_URI =
    Uri.parse("content://" + PROVIDER_NAME + "/" + DATABASE_TABLE);

  /* ------------------------------ Variables ------------------------------ */

  EditText codeText;  /** Module code text field */
  EditText nameText;  /** Module name text field */

  /* ------------------------------ Methods ------------------------------ */
```

```
  /**
    Create user interface and set up listeners for buttons.

    @param savedInstanceState    previously saved state
  */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    // create basic interface
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_module_provider);

    // set up module fields, with only code being editable
    codeText = (EditText) findViewById(R.id.codeValue);
    nameText = (EditText) findViewById(R.id.nameValue);
    nameText.setFocusable(false);

    // when clear is clicked, empty the module fields
    Button clearButton = (Button) findViewById(R.id.clearButton);
    clearButton.setOnClickListener(new OnClickListener() {
      public void onClick(View view) {
        codeText.setText("");
        nameText.setText("");
      }
    });

    // when translate is clicked, get module name corresponding to module code
    Button translateButton = (Button) findViewById(R.id.translateButton);
    translateButton.setOnClickListener(new OnClickListener() {
      public void onClick(View view) {
        try {
          String moduleCode = codeText.getText().toString();
          Uri uri = Uri.parse(CONTENT_URI + "/" + moduleCode);
          Cursor cursor = managedQuery(uri, null, null, null, null);
          if (cursor.getCount() == 1) {
            cursor.moveToFirst();
            String moduleName = cursor.getString(cursor.getColumnIndex("name"));
            nameText.setText(moduleName);
          }
          else {
            nameText.setText("");
            throw(new Exception("module code invalid"));
          }
        }
        catch (Exception exception) {
          // report problem in pop-up window
          Toast.makeText(view.getContext(),
            "Invalid data - " + exception.getMessage(),
            Toast.LENGTH_SHORT).show();
        }
      }
    });
  }
}
```

You have now reached a checkpoint that you should show to a lab demonstrator. Also be prepared to answer the following questions:

- **Why are different packages used for the provider and the client?**
- **Does the provider really need an associated activity?**
- **What might a provider activity usefully do in general?**