**Integrating the Scriptures into the Application**

In our Potential Risks and Solutions document, we identified a feature that we would love to incorporate into our application, but that brought a number of challenges with it. That feature is the integration of the actual scripture texts within the schedule. After doing some research, we have found what seems to us to be a solid solution to not only this, but another risk we identified: calculating the schedule (We will discuss this more later).

We were able to find an incredibly handy resource on scriptures.nephi.org that should make this integration possible. On the above mentioned website, they provide the entire LDS Standard Works compiled into several formats including JSON, SQL, SQLite, CSV, and more. We were most intrigued by the JSON format. This solves our problem of having to work with a proprietary API, as these resources are available in the public domain.

The JSON file contains every verse in the standard works with unique identifiers and metadata. Below is an example of one verse object as listed in this JSON file:

{"volume\_id":1,"book\_id":1,"chapter\_id":1,"verse\_id":1,"volume\_title":"Old Testament","book\_title":"Genesis","volume\_long\_title":"The Old Testament","book\_long\_title":"The First Book of Moses called Genesis","volume\_subtitle":"","book\_subtitle":"","volume\_short\_title":"OT","book\_short\_title":"Gen.","volume\_lds\_url":"ot","book\_lds\_url":"gen","chapter\_number":1,"verse\_number":1,"scripture\_text":"IN the beginning God created the heaven and the earth.","verse\_title":"Genesis 1:1","verse\_short\_title":"Gen. 1:1"}

Our idea is that we can use this data to create verse objects in our application. Each day’s reading will generate the necessary verse objects when loading the scriptures, based on the unique identifiers such as chapter\_id and verse\_id and store them in a container. We can then display the needed scripture text to the user upon request.

**Calculating the Schedule**

This issue has also been largely alleviated due to the system of identifiers provided by the JSON file. Because each chapter has a unique id and we know how many total chapters there, we can use simple arithmetic to find how many chapters to be read per day. We would just need to store the chapter ids of the needed reading within the corresponding day object. When displaying a particular day’s reading, we can just pull the information such as Book, Chapter, and Verse from the object with a matching id. This eliminates any need to store chapter and book in an outside table or file.

**Pushing Notifications to the System**

A standard feature of mobile applications are Notifications. They are application specific messages informing the user that some action(s) need to be taken. This is a required feature of our app. The official documentation on Android Notifications was very helpful. The following code snippet shows how to integrate Notifications into an app.

NotificationCompat.Builder mBuilder =  
        new NotificationCompat.Builder(this)  
        .setSmallIcon(R.drawable.notification\_icon)  
        .setContentTitle("My notification")  
        .setContentText("Hello World!");  
// Creates an explicit intent for an Activity in your app  
Intent resultIntent = new Intent(this, ResultActivity.class);  
  
// The stack builder object will contain an artificial back stack for the  
// started Activity.  
// This ensures that navigating backward from the Activity leads out of  
// your application to the Home screen.  
TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);  
// Adds the back stack for the Intent (but not the Intent itself)  
stackBuilder.addParentStack(ResultActivity.class);  
// Adds the Intent that starts the Activity to the top of the stack  
stackBuilder.addNextIntent(resultIntent);  
PendingIntent resultPendingIntent =  
        stackBuilder.getPendingIntent(  
            0,  
            PendingIntent.FLAG\_UPDATE\_CURRENT  
        );  
mBuilder.setContentIntent(resultPendingIntent);  
NotificationManager mNotificationManager =  
    (NotificationManager) getSystemService(Context.NOTIFICATION\_SERVICE);  
// mId allows you to update the notification later on.  
mNotificationManager.notify(mId, mBuilder.build());

Here is a screenshot from an emulator of the above working code

