Extracted from:

Programming Google Glass

The Mirror API

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

The Pragmatic Programmers

Pragmatic exPress

Programming Google Glass

The Mirror API

Eric Redmond

Edited by Jacquelyn Carter

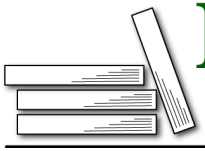
Programming Google Glass

The Mirror API

Eric Redmond

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina



Pragmatic Bookshelf

Our Pragmatic courses, workshops, and other products can help you and your team create better software and have more fun. For more information, as well as the latest Pragmatic titles, please visit us at *http://pragprog.com*.

The team that produced this book includes:

Jacquelyn Carter (editor)
Candace Cunningham (copyeditor)
David J Kelly (typesetter)
Janet Furlow (producer)
Juliet Benda (rights)
Ellie Callahan (support)

Printed in the United States of America.
ISBN-13: 978-1-937785-79-6
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—December 2013

CHAPTER 6

Making Glass Social

Needless to say, computers have evolved tremendously in the past few decades. Twenty years ago, processors were slower and ran hotter, memory was expensive, and displays were heavy and lower quality. But for many consumers, the biggest changes have been around networking. The advent of social networking raised an entire generation into savvy computer users and gave us the ability to express ourselves to friends and the world by sharing photos and videos. This ability is no longer an optional attribute of a mobile computing device.

This chapter covers the remainder of the Mirror API, which allows us to add depth to our application beyond a simple list of cards in a timeline. Contacts and attachments are enhancements to a Glassware. Unlike timeline items or menus, which are the basic necessary building blocks of Glassware, these final puzzle pieces will allow our application to store, retrieve, and share assets with others entirely though Glass. These are the tools we can use to add a social-network dimension to our Glassware.

Creating Contacts

A necessary component of any social activity is other contacts to share with. A contact, in the most general sense, is an individual or a group of things (be they humans or maybe even cats). It's a conceptual construct on who or what you can share with, and not necessarily other Glass users. There are two sides to contacts in the Mirror API: the contacts resource where users can manage their personal *contacts*, and timeline-item integration, where a contact is the *creator* or *recipient* of an item.

A contact is data about a person or group, like `displayName`, `phoneNumber`, or `imageUrls`. It represents someone you can share timeline items with, or someone you wish to more easily message or call on a regular basis.

Since Lunch Roulette is about connecting a potential patron (Glass user) with a restaurant, it stands to reason a user may want to add a restaurant as a contact, especially if the user plans to eat there at a later date. Let's add another custom menu option to a timeline item to allow a user to add the current restaurant timeline item to his contact list. It's just like our last custom menu item, but this time we'll give it the id of ADD_CONTACT to differentiate it from our Alternative menu item from the last chapter.

chapter-6/src/test/book/glass/LunchRoulette.java

```
timelineItem.getMenuItems().add( new MenuItem()
    .setAction( "CUSTOM" )
    .setId( "ADD_CONTACT" )
    .setRemoveWhenSelected( true )
    .setValues( Collections.singletonList( new MenuValue()
      .setState( "DEFAULT" )
      .setDisplayName( "Add As Contact" ) )
    )
);
```

This adds a menu item to tap (and then removes it once it's chosen), but we'll need to know some information about the restaurant in order to create a contact. We have a couple of options for storing that information. Either we can use sourceItemId to set a unique ID on the timeline item to represent a particular restaurant, or we can set the contact to the timeline item.

You may notice that TimelineItem does *not* have a setContact method, but it does allow us to setCreator(). This is a bit of a double punch, since not only does this store our contact, allowing us to retrieve it later, but it also lets us leverage another menu-item action, called VOICE_CALL. This will make a call directly from the timeline item, since it *call*s the creator's phone_number.

chapter-6/src/test/book/glass/LunchRoulette.java

```
timelineItem.getMenuItems().add(
    new MenuItem().setAction( "VOICE_CALL" ) );
```

Next let's set the restaurant contact on the timeline item. Since we have a restaurant's name and phone number, we can add those to the contact. The contact type can be either an INDIVIDUAL (the default) or a GROUP. We'll keep it as INDIVIDUAL, so it will show up in your Call menu-item contact list.

chapter-6/src/test/book/glass/LunchRoulette.java

```
TimelineItem timelineItem = new TimelineItem()
  .setCreator(
    new Contact()
      .setDisplayName( restaurant.getName() )
      .setPhoneNumber( restaurant.getPhone() )
      .setType( "INDIVIDUAL" ) )
```

Mirror will call the TimelineUpdateServlet when the menu item is tapped, due to the subscription we created at the end of Chapter 5, *Tracking Movement and User Responses*, on page ?. Now we need to add another if statement to specify what to do with an ADD_CONTACT payload.

With the menu items and contact set, creation of this timeline item will populate a hidden display name and phone number. You can reference this with a regular timeline.get(), then pass that same contact object to the contact resource's insert() once, adding a unique ID. Unlike for other objects we've created (TimelineItem, Subscription), you are responsible for setting this contact's unique ID.

chapter-6/src/test/book/glass/notifications/TimelineUpdateServlet.java

```
else if( "ADD_CONTACT".equals( userAction.getPayload() ))
{
  Mirror mirror = MirrorUtils.getMirror( userId );
  Timeline timeline = mirror.timeline();
  Contacts contacts = mirror.contacts();

  TimelineItem timelineItem = timeline.get( itemId ).execute();

  Contact contact = timelineItem.getCreator();
  contact.setId( UUID.randomUUID().toString() );

  contacts.insert( contact ).execute();
}
```

Like all other resources with an insert() method, this generates an HTTP PUT with a contact's populated JavaScript Object Notation (JSON) object. You can see all contact fields in Appendix 1, *HTTP and HTML Resources*, on page ?.

After you've deployed and tested your new menu action, you should see a new contact in your list. You can view your list in code via contacts.list().execute(), which returns a ContactListResponse object (similar to LocationsListResponse or 'TimelineListResponse').

```
ContactListResponse list = contacts.list().execute();
for( Contact item : list.getItems() ) {
  System.out.println( item );
}
```

As with other resources, you can individually get, update, or delete each contact.

Sharing Assets with Glassware

Contacts are meant to be more than simply people or businesses. Your Glassware itself can actually be a contact. This means your users can share

items with a Glassware application. In my opinion, this was an odd design choice on Google's part, overloading Contacts to represent living individuals or groups *and* a software application. However, sharing a timeline item, such as images or video, requires a contact to share it with. So let's make do.

If you create a contact, coupled with a subscription to a SHARE event, then your Glassware can respond to notifications from timeline items that your application didn't create, such as photographs.

We'll expand Lunch Roulette again, this time by letting users share photos they've taken of their food. Once they take a photo, they can choose to share it with Lunch Roulette (the Twitter and Facebook apps function similarly). After a shared timeline item is created, our callback URL will be notified. For now we'll just log this notification to expand upon later in this chapter.

Let's begin by creating a new contact that represents Lunch Roulette. Just like we added a subscription after a successful user login, we can add this app as a contact.

chapter-6/src/test/book/glass/LunchRoulette.java

```
public static void addAppAsContact( HttpServletRequest req, String userId )
    throws IOException
{
  Mirror mirror = MirrorUtils.getMirror(req);
  Contacts contacts = mirror.contacts();

  Contact contact = new Contact()
    .setId( "lunch-roulette" )
    .setDisplayName( "Lunch Roulette" )
    .setImageUrls( Collections.singletonList(
        PROD_BASE_URL + "/static/images/roulette.png" ) )
    .setPriority( 0L )
    .setAcceptTypes( Collections.singletonList( "image/*" ) );

  contacts.insert( contact ).executeAndDownloadTo( System.out );
}
```

This code is called in the OAuth2Servlet's doAuth method after a user has successfully logged in.