Part 2 - Managing Fragments

To help with managing Fragments, Android provides the `FragmentManager` class. Each Activity has an instance of `Android.App.FragmentManager` that will find or dynamically change its Fragments. Each set of these changes is known as a *transaction*, and is performed by using one of the APIs contained in the class `Android.App.FragmentTransation`, which is managed by the `FragmentManager`. An Activity may start a transaction like this:

```
FragmentTransaction fragmentTx =
this.FragmentManager.BeginTransaction();
```

These changes to the Fragments are performed in the `FragmentTransaction` instance by using methods such as `Add()`, `Remove()`, and `Replace()`. The changes are then applied by using `Commit()`. The changes in a transaction are not performed immediately. Instead, they are scheduled to run on the Activity's UI thread as soon as possible.

The following example shows how to add a Fragment to an existing container:

```
// Create a new fragment and a transaction.
FragmentTransaction fragmentTx =
this.FragmentManager.BeginTransaction();
DetailsFragment aDifferentDetailsFrag = new DetailsFragment();

// The fragment will have the ID of Resource.Id.fragment_container.
fragmentTx.Add(Resource.Id.fragment_container,
aDifferentDetailsFrag);

// Commit the transaction.
fragmentTx.Commit();
```

If a transaction is committed after `Activity.OnSaveInstanceState()` is called, an exception will be thrown. This happens because when the Activity saves its state, Android also saves the state of any hosted Fragments. If any Fragment transactions are committed after this point, the state of these transactions will be lost when the Activity is restored.

It's possible to save the Fragment transactions to the Activity's back stack by making a call to `FragmentTransaction.AddToBackStack()`. This allows the user to navigate backwards through Fragment changes when the Back button is pressed. Without a call to this method, Fragments that are removed will be destroyed and will be unavailable if the user navigates back through the Activity.

The following example shows how to use the `AddToBackStack` method of a `FragmentTransaction` to replace one Fragment, while preserving the state of the first Fragment on the back stack:

```
// Create a new fragment and a transaction.
FragmentTransaction fragmentTx =
this.FragmentManager.BeginTransaction();
DetailsFragment aDifferentDetailsFrag = new DetailsFragment();

// Replace the fragment that is in the View fragment_container (if
applicable).
fragmentTx.Replace(Resource.Id.fragment_container,
aDifferentDetailsFrag);

// Add the transaction to the back stack.
fragmentTx.AddToBackStack(null);

// Commit the transaction.
fragmentTx.Commit();
```

Communicating with Fragments

The *FragmentManager* knows about all of the Fragments that are attached to an Activity and provides two methods to help find these Fragments:

- **FindFragmentById** – This method will find a Fragment by using the ID that was specified in the layout file or the container ID when the Fragment was added as part of a transaction.
- **FindFragmentByTag** – This method is used to find a Fragment that has a tag that was provided in the layout file or that was added in a transaction.

Both Fragments and Activities reference the `FragmentManager`, so the same techniques are used to communicate back and forth between them. An application may find a reference Fragment by using one of these two methods, cast that reference to the appropriate type, and then directly call methods on the Fragment. The following snippet provides an example:

It is also possible for the Activity to use the `FragmentManager` to find Fragments:

```
var emailList = FragmentManager.FindFragmentById<EmailListFragment>
(Resource.Id.email_list_fragment);
emailList.SomeCustomMethod(parameter1, parameter2);
```

Communicating with the Activity

It is possible for a Fragment to use the `Fragment.Activity` property to reference its host. By casting the Activity to a more specific type, it is possible for an Activity to call methods and properties on its host, as shown in the following example:

```
var myActivity = (MyActivity) this.Activity;
myActivity.SomeCustomMethod();
```