Opportunity Watcher

powered by

ADOBE® FLASH® BUILDER™ FOR FORCE.COM

Developer's Documentation

July 14, 2010

salesforce.com®

Table of Contents

1. Introduction 3
   - 1.1. Purpose 3
   - 1.2. Audience 3
   - 1.3. Other Reference Materials 3
2. Setting up Opportunity Watcher 5
   - 2.1. Identify a Salesforce.com Environment 5
   - 2.2. Enable Chatter 6
   - 2.3. Configure the Offline Briefcase 7
   - 2.4. Download and Install Opportunity Watcher 7
   - 2.5. Download the Opportunity Watcher source code 7
   - 2.6. Log in with Opportunity Watcher 8
3. F3 Classes and Components 9
   - 3.1. F3DesktopApplication 9
   - 3.2. F3DesktopWrapper 10
   - 3.3. Connection 11
   - 3.4. DynamicEntity 12
   - 3.5. EntityContainer 12
   - 3.6. FieldContainer 14
   - 3.7. StatusBar 15
   - 3.8. Toaster 16
4. Development Methods and Best Practices 18
   - 4.1. Incremental uploads versus batch uploads 18
   - 4.2. Creating records in offline mode 18
   - 4.3. Using the Connection class 19
   - 4.4. EntityContainer versus FieldContainer 19
   - 4.5. Fiber classes versus the DynamicEntity class 20
   - 4.6. Storing and accessing Salesforce.com records in application memory 20

1. Introduction

1.1. Purpose

The purpose of the Opportunity Watcher Developer's Documentation is to educate a new Adobe Flash Builder for Force.com (F3) developer on the features and benefits of the F3 framework, though examples displayed in the Opportunity Watcher sample application.

This document highlights individual features, describes their usage in Opportunity Watcher, and their possible uses in other applications. It also includes relevant code samples directly from the Opportunity Watcher source code. Finally, a number of development methods and best practices are described in detail, to help developers learn how to get the most out of the F3 platform.

1.2. Audience

This document, as well as the source code available for Opportunity Watcher, is recommended for developers with previous experience in Adobe Flash Builder 4, Adobe AIR, and Salesforce.com. An understanding of Cairngorm (a Flex framework) is also encouraged but not necessary. Reference material and other resources for all of the listed technologies can be found in the next section.

This document will be most useful to developers looking to learn about the F3 framework and its advantages, those building a new F3 application, or extending and modifying the open-source Opportunity Watcher code. Standard Salesforce users who simply want to use or learn more about Opportunity Watcher should instead view the videos available at http://developer.force.com/flashbuilder.

1.3. Other Reference Materials

Adobe Flash Builder for Force.com Developer's Guide (http://developer.force.com/flashbuilder)

The main source of F3 documentation for developers, this guide details the various components and features that F3 has to offer. It is strongly recommended you have this document on hand as a supplement when reviewing this one, and throughout your F3 development.

Adobe Flex Developer Center (http://www.adobe.com/devnet/flex)

Adobe's starting point for new Flex developers, this site features a number of important features, such as Adobe Flex in a Week and Tour de Flex.

Adobe AIR Developer Center (http://www.adobe.com/devnet/air)

Adobe's starting point for new AIR developers, this site walks through the many important benefits of using AIR to publish rich desktop applications.

Developer Force (http://developer.force.com)

Salesforce.com's site for Force.com developers, this site features a great array of resources regarding force.com code, Chatter, and more.

Introducing Cairngorm (http://www.adobe.com/devnet/flex/articles/introducing_cairngorm.html)

An introductory walkthrough of the Cairngorm framework, a Flex framework used in the Opportunity Watcher source code.

2. Setting up Opportunity Watcher

2.1. Identify a Salesforce.com Environment

In order to run the Opportunity Watcher application, you must first have access to a Salesforce.com environment. This can be done in one of two ways:

- **Using an existing environment**

  If you currently have access to a Salesforce.com environment, you may use your username and password to log in with Opportunity Watcher. In order to get the most from the application,

  - your profile should have the ability to read and edit Opportunities and Accounts,
  - your environment should have Chatter enabled, and
  - you should be assigned to an Offline Briefcase Configuration that includes Opportunities and Accounts.

  If any of these are not configured correctly, please contact the administrator of your Salesforce environment. If you are the administrator, more details about enabling Chatter and setting up the Offline Briefcase Configuration are available in the next sections of this document.

- **Setting up a free Developer Edition of Salesforce.com**
  If you do not have access to a Salesforce.com environment, or would like to create one solely for the purpose of trying Opportunity Watcher or exploring its code, you can get a free Developer Edition environment. To do so, visit http://developer.force.com and click the "Get a Free Developer Edition" link, as shown below.

This link will walk you through the steps needed to get a Developer Edition of Salesforce.com, which is a very basic environment with standard settings and some sample records (such as Opportunities and Accounts) already available.

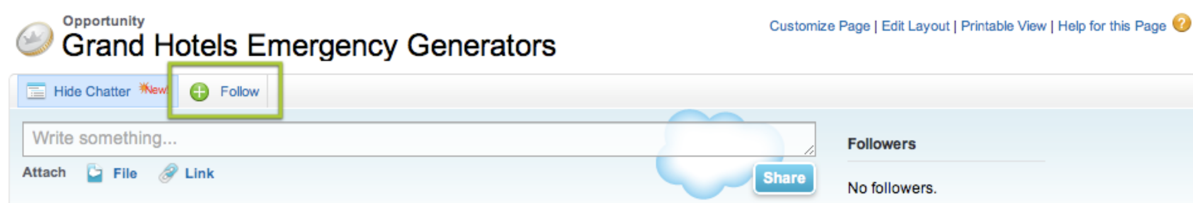
2.2. Enable Chatter

Opportunity Watcher makes use of the Salesforce Chatter feature to notify the user of the latest important changes to their Opportunities. In order to do this, the Salesforce.com environment must first have Chatter enabled.

To enable Chatter:

1. Log in to your Salesforce.com environment.
2. Click the "Setup" link at the top of the page.
3. In the left toolbar, navigate to Customize > Chatter > Settings.
4. If the checkbox is unchecked, click Edit, then check it, then click "Save".

After enabling Chatter, it is recommended that you go to the detail page for one or more Opportunities and click the "Follow" button (as shown below.) This will cause you to be notified to changes to these Opportunities while running Opportunity Watcher.



If you are in your own Developer Edition environment, or are working with test or sample records that other users are not relying upon, it is also recommended that you edit and save the Opportunities that you are following a couple of times. This could be simply changing a field, or perhaps using the “Write something…” prompt (shown above) to enter a custom comment. Doing so will generate a few sample Chatter updates for those Opportunities.

2.3. Configure the Offline Briefcase

Salesforce.com uses the Offline Briefcase to determine which records a user can access with an external application (in this case, Opportunity Watcher.)

To configure the Offline Briefcase for Opportunity Watcher:

1. Log in to your Salesforce.com environment.
2. Click the “Setup” link at the top of the page.
3. In the left toolbar, navigate to Desktop Administration > Offline Briefcase Configurations.
4. Click “New Offline Briefcase Configuration”.
5. Give the new configuration a name, such as “Opportunity Watcher Briefcase”.
6. Click the “Active” checkbox.
7. Use the “Add” arrow buttons to move your name from the “Available Members” list to the “Assigned Members” list.
8. Click “Save”.
9. Under the “Data Sets” section, click the “Edit” button.
10. Click the “Add” button, then select “Opportunity”. Select “Close Date” as the field to order by.
11. Make sure “Opportunity” is selected in the tree to the left. Click the “Add” button, then select “Opportunity Product”.
12. Select “Data Sets” in the tree to the left. Click the “Add” button, then select “Account”. Select “Created Date” as the field to order by.

2.4. Download and Install Opportunity Watcher

If you have not done so already, install Opportunity Watcher from this page.

2.5. Download the Opportunity Watcher source code

If you have not done so already, you can download the Opportunity Watcher source code from this page. To view it in Adobe Flash Builder for Force.com, open your F3 executable, then select File > Import. Point it to the .zip file you’ve downloaded. You will then have a new project called OppWatcher that you can run, explore, and modify however you’d like.

2.6. Log in with Opportunity Watcher

When you begin the Opportunity Watcher application, you will be prompted to log in, as shown below.

Enter your Salesforce.com username and password, then click “Login”. If you receive an error prompting you to use a security token, follow the instructions given in the error message to retrieve your personal security token. Once you have it, append it to your password in order to log in with Opportunity Watcher.

Examining the Opportunity Watcher code

You can add your username, password, and/or token to the application’s source code, so that you don’t need to enter it every time. To do so, examine the code in LoginView.mxml and edit the text properties of the username and password TextInput components.

```
<mx:FormItem label="Username">
    <!--Insert a username into the text property below if you'd like to
    have a default username appear every time.-->
    <s:TextInput id="username"
                 text=""/>
</mx:FormItem>
<mx:FormItem label="Password"
             direction="horizontal">
    <!--Insert a password (and possibly security token) into the text property
    below if you'd like to have a default password appear every time.-->
    <s:TextInput id="password"
                 text=""
                 displayAsPassword="true"/>
```

3. F3 Classes and Components

F3 comes with many custom classes and components right out of the box. These can be used to create powerful online/offline applications with ease, and are individually highlighted in the subsections below. Details about the Opportunity Watcher source code are also included. These details can help a new F3 developer make modifications to Opportunity Watcher, or create their own F3 application.

Further details for all of the classes and components below can be found in the Adobe Flash Builder for Force.com Developer's Guide.

3.1. F3DesktopApplication

This class is at the very core of F3 functionality. It performs several important roles:

- Holds the F3DesktopWrapper class (detailed in a later section) to perform basic Salesforce.com read/write actions.
- Holds the Connection class (detailed in a later section) to perform advanced Salesforce.com read/write actions.
- Can listen for several important events (such as going online or offline, login expiration, etc.)
- Provides status updates to the application.

Examining the Opportunity Watcher code

The instance of F3DesktopApplication in Opportunity Watcher is declared in the main OppWatcher.mxml file. There are also several event listeners set up on the F3DesktopApplication to catch important events. This instance is also available to other classes and components through our ModelLocator class (per standard Cairngorm practices) as "stratus" (the previous codename for F3.)

```
<!--The core of F3, F3DesktopApplication holds the wrapper class used for online/offline briefcase
data storage, the standard Flex toolkit Connection class used for Chatter and other data,
and much more. It also has several event handlers to facilitate login, session expiration, etc.-->
<flexforforce:F3DesktopApplication id="stratus"
                                   statusChanged="statusChangedHandler(event)"
                                   loginComplete="loginCompleteHandler(event)"
                                   loginFailed="loginFailedHandler(event)"
                                   sessionExpired="sessionExpiredHandler(event)"
                                   serverUrl="https://login.salesforce.com/services/Soap/u/18.0"/>
```

3.2. F3DesktopWrapper

Another class vital to F3 functionality is F3DesktopWrapper. It is used to:

- query data from Salesforce.com,
- cache that data to a local database, and
- synchronize changes made locally back up to Salesforce.com.

F3DesktopWrapper handles all caching and synching without the need for any custom code. All of the CRUD operations for objects and records specified in the Offline Briefcase Configuration of your Salesforce.com environment should utilize this class.

Examining the Opportunity Watcher code

Opportunity Watcher uses the F3DesktopWrapper class (by referencing model.stratus.wrapper) in several places, but mainly in the SFDCCommand.query() method.

```
//performs an F3 query
private function query():void {
    model.stratus.wrapper.query(evt.vo, new mx.rpc.Responder(queryComplete, StaticUtils.genericFault));
}

//returns F3 query results to caller, if any
private function queryComplete(rows:ArrayCollection):void {
    notifyCaller(rows);
}
```

This method uses a query string (in this case stored as evt.vo) to perform a query of the local database. This is an asynchronous query, so a Responder is called when the operation is complete. The resulting set of records comes in as an ArrayCollection, and can then be passed back to whichever function fired the query event in the first place.

The wrapper class is also used in many of the “Modal” components to create new records, or to update existing ones. For example, the OppModal component first makes the following call if the record is a new one:

```
model.stratus.wrapper.save(record);
```

This saves the record locally and prepares it to be uploaded to the Salesforce.com servers. After doing this, or if the record already existed and the user is simply making an update, the code calls the syncWithServer() method.

```
if(model.stratus.connected){
    model.stratus.wrapper.syncWithServer([record], new mx.rpc.Responder(saveComplete, StaticUtils.genericFault));
```

Although we are only passing one record to the method, it is possible to send a collection of many, allowing for applications that only save changes locally until the user chooses to make a batch update back to Salesforce.com.

3.3. Connection

Similar to the F3DesktopWrapper class, the Connection class performs CRUD operations between the F3 application and the Salesforce.com servers. It is also a property of the F3DesktopApplication class as well. However, it does not automatically cache data locally, so it is recommended that you use the F3DesktopWrapper class whenever possible.

Developers with previous Flex/Salesforce.com experience may recognize the Connection class as the same one included in the standard Flex Toolkit. In an F3 application, it can be useful for

- performing CRUD operations on advanced objects that cannot be put in the Offline Briefcase Configuration (such as Chatter objects), and
- performing describe calls to get various sets of metadata.

The instance of the Connection class on an application's F3DesktopApplication instance automatically connects when the user logs in through the F3DesktopApplication. It shares the same sessionID and serverURL properties and does not have to establish a separate connection.

Examining the Opportunity Watcher code

Opportunity Watcher uses the Connection class to read Chatter feeds for Opportunity records, and also to add or remove user subscriptions (or "follows") to Opportunity records. This occurs primarily in the SFDCCommand class, as shown below.

```
//peforms a Flex toolkit query to get Chatter records
private function queryChatter():void{
    model.stratus.connection.query(evt.vo, new Callbacks(chatterQueryResultHandler, StaticUtils.genericFault));
}

//peforms a Flex toolkit query to get Chatter subscriptions
private function queryEntity():void{
    model.stratus.connection.query(evt.vo, new Callbacks(entityQueryResultHandler, StaticUtils.genericFault));
}

private function addEntitySubs():void {
    model.stratus.connection.upsert("Id", [evt.vo], new Callbacks(entitySubAddHandler, StaticUtils.genericFault))
}
```

Once the Chatter data (in this case, OpportunityFeed records) is retrieved, it is formatted for display in Opportunity Watcher's ChatterBox component and displayed to the user.

3.4. DynamicEntity

DynamicEntity is a highly useful F3 class that serves as a value object for Salesforce.com records. Similar to the SObject class in the Flex Toolkit, each instance of DynamicEntity represents a record from Salesforce.com, such as an Account or an Opportunity.

Examining the Opportunity Watcher code

In Opportunity Watcher, all Salesforce.com records pulled in by F3DesktopWrapper (either from the local database or from the Salesforce.com servers) remain as instances of DynamicEntity. New records are also created as instances of DynamicEntity, as shown below.

```
private function newOppClicked():void {
    //fires off an event to open a modal for a new Opportunity
    var newOpp:DynamicEntity = new DynamicEntity("Opportunity");
    newOpp.AccountId = parentRecord.Id;
    new ModalEvent(newOpp, ModalEvent.OPEN_MODAL).dispatch();
}
```

3.5. EntityContainer

One of F3's greatest advantages over the Flex Toolkit is the EntityContainer component. It can be used to render a Salesforce.com detail page layout, with very little effort from the developer. EntityContainer can render layouts for almost any standard or custom object, and allows for create, inline edit, and full edit views.

Opportunity Detail

Opportunity Owner	Sara Renberg	**Amount**	$ 1310000.00
Private		**Expected Revenue**	$ 655000.00
Opportunity Name	Dickenson East Installation	**Close Date**	06/09/2010
Account Name	Dickenson PLC	**Next Step**	Call
Type		**Stage**	Value Proposition
Lead Source	Web	**Probability (%)**	50%
		Primary Campaign Source	
Order Number		**Main Competitor(s)**	
Current Generator(s)		**Delivery/Installation Status**	
Tracking Number			
Created By	Sara Renberg, 06/14/2010 2:27 PM	**Last Modified By**	Sara Renberg, 07/06/2010 9:48 AM
Description	Test, Test, Test		

EntityContainer rendering an Opportunity in inline edit view

Opportunity Edit

Opportunity Information | = Required Information

Opportunity Owner	Sara Renberg	Amount	1310000.00
Private		Close Date	06/09/2010 07/13/2010
Opportunity Name	Dickenson East Installation	Next Step	Call
Account Name	Dickenson PLC	Stage	Value Proposition
Type	--None--	Probability (%)	50
Lead Source	Web	Primary Campaign Source	

Additional Information

Order Number		Main Competitor(s)	
Current Generator(s)		Delivery/Installation Status	--None--
Tracking Number			

Description Information

Description	Test, Test, Test

EntityContainer rendering an Opportunity in full edit view

EntityContainer has two important methods that warrant highlighting:

- **render(*dynamicEntity*)** – By passing a record (as an instance of DynamicEntity) to the EntityContainer's render() method, it will automatically generate the create, full edit, or inline edit view for that record.

- **updateObject(*responder*)** – After the user makes any changes to the field values for the record through EntityContainer, the updateObject() method can be used to update the object in the local database. This operation is performed asynchronously. When it is completed, it will call the method specified by the Responder it was passed.

Examining the Opportunity Watcher code

Opportunity Watcher's AccModal and OppModal components both utilize EntityContainer to display their respective objects' detail layouts to the user. In the case of OppModal, there are two, labeled createEC and editEC. One or the other appears at any given time, based on whether the user is creating a new Opportunity record or editing an existing one.

```
<view:CollapsiblePanel title="Opportunity Detail">
    <flexforforce:EntityContainer id="createEC"
                                  startState="Full Edit"
                                  width="100%"
                                  includeIn="create"/>

    <flexforforce:EntityContainer id="editEC"
                                  startState="Inline Edit"
                                  width="100%"
                                  excludeFrom="create"/>
</view:CollapsiblePanel>
```

It may seem redundant to have two separate instances, between which the only difference is their startState value, but this is important due to layout differences that may exist between the different views. Therefore, it is a recommended best practice to set the startState value for an EntityContainer in its MXML declaration, and not to change it afterwards.

NOTE: Even though createEC is used when creating a new Opportunity, it uses the "Full Edit" value for the EntityContainer's startState value. This is because we are creating a new instance of DynamicEntity elsewhere (to represent our new Opportunity record) and passing it in to the OppModal component to be presented to the user.

3.6. FieldContainer

FieldContainer works much like EntityContainer, in that it renders a detail view for a given record, in create, full edit, or inline edit modes. However, FieldContainer does not automatically use the layout metadata provided by Salesforce.com, and instead allows the developer to specify which fields should appear. This can be useful if a custom layout is required, or if one is not available through the layout metadata.

To use a FieldContainer, simply declare one with the proper startState value, then populate its fieldCollection property with one or more instances of LabelAndField (another highly useful F3 component that emulates a Salesforce.com field, complete with label.) Then call the fieldCollection's render() property, just as you would with EntityContainer.

Examining the Opportunity Watcher code

Opportunity Watcher makes use of FieldContainer to render the detail view for OpportunityLineItem records in the OLIModal component. Because there is no standard layout for OpportunityLineItem in the metadata that F3 pulls down from Salesforce.com, FieldContainer should be used to render this form.

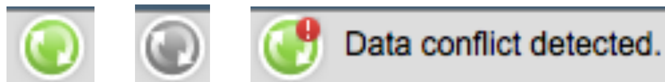
Just as with the EntityContainer instances in OppModal, there are two FieldContainer instances in OLIModal – one for creating a new record and another for editing an existing record. In the code below, standard Flex 4 HGroup components are used to lay out the LabelAndField components properly.

```
<flexforforce:FieldContainer id="createFC"
                             startState="Full Edit"
                             width="100%"
                             includeIn="create">
    <s:HGroup width="100%">
        <flexforforce:LabelAndField id="createQuantityField"
                                    field="OpportunityLineItem.Quantity"/>
        <flexforforce:LabelAndField id="createPriceField"
                                    field="OpportunityLineItem.UnitPrice"/>
    </s:HGroup>
    <s:HGroup width="100%">
        <flexforforce:LabelAndField id="createDateField"
                                    field="OpportunityLineItem.ServiceDate"/>
        <flexforforce:LabelAndField id="createDescField"
                                    field="OpportunityLineItem.Description"/>
    </s:HGroup>
</flexforforce:FieldContainer>

<flexforforce:FieldContainer id="editFC"
                             startState="Inline Edit"
                             width="100%"
                             excludeFrom="create">
    <s:HGroup width="100%">
        <flexforforce:LabelAndField field="OpportunityLineItem.Quantity"/>
        <flexforforce:LabelAndField field="OpportunityLineItem.UnitPrice"/>
    </s:HGroup>
    <s:HGroup width="100%">
        <flexforforce:LabelAndField field="OpportunityLineItem.ServiceDate"/>
        <flexforforce:LabelAndField field="OpportunityLineItem.Description"/>
    </s:HGroup>
</flexforforce:FieldContainer>
```

3.7. StatusBar

Another useful piece of F3 functionality comes in the form of StatusBar, a horizontal bar intended to run across the bottom of the application. This bar provides status messages to the user, and features an icon that informs users at a glance of the application's status, including online/offline modes and data conflicts. Users can click on this icon in order to refresh local data from the Salesforce.com servers, or to resolve data conflicts.



The StatusBar icon while online, offline, and during data conflict.

Examining the Opportunity Watcher code

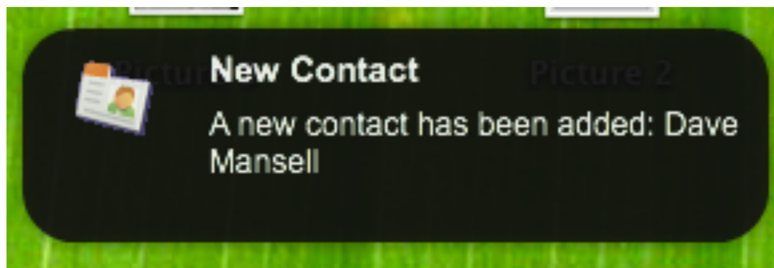
In Opportunity Watcher, the StatusBar component is declared on the parent application, OppWatcher.mxml.

```
<!--The standard F3 StatusBar, used to display network connectivity, conflicts, etc.-->
<flexforforce:StatusBar left="0"
                        bottom="0"/>
```

From there, the component takes care of the rest, including displaying some default status messages to the user. If your application needs to set a custom message, use F3DesktopApplication's setStatus() method.

3.8. Toaster

One powerful advantage of a desktop application (versus a web application) is the ability to notify the user of important events through small popup windows, or toasters. To that end, F3 features a convenient component called Toaster, which displays a brief message in the corner of the user's screen before fading away. The F3 Toaster component can be used for any kind of user alert, and supports the display of multiple toaster images at the same time.



To display a toaster to the user, simply create an instance of the Toaster class, passing it string values for the title and body. You can also provide an icon, several of which are included in F3's StaticAssets class. Then call F3DesktopApplication's showToaster() method, passing in the Toaster instance as a parameter.

Examining the Opportunity Watcher code

In Opportunity Watcher, the Toaster component is used to notify the user of Chatter activity around the Opportunity records they are following. This is achieved through an auto-refresh of the Chatter feed that occurs every 20 minutes while the application is running and online.

```
public function updateChatter(...args):void {
    //called every 20 minutes to update the Chatter feed
    if(model.stratus.connected){
        model.chatter = new ArrayCollection();
        model.chatterById = new Array();
        var updateChatterCallbacks:Callbacks = new Callbacks(chatterUpdateHandler, StaticUtils.genericFault);

        //query for new Chatter information
        new SFDCEvent(StaticUtils.getChatterQueryString(), SFDCEvent.QUERY_CHATTER, updateChatterCallbacks).dispatch();

    }
}
```

If any new Chatters are detected, a toaster alert is popped to notify the user of each new message. (The code snippets above and below both come from OppWatcher.mxml.)

```
private function chatterUpdateHandler(qr:QueryResult):void {
    //if there are new Chatters, pop a toaster alert for each
    var lastChatterDateString:String = StaticUtils.dateToString(lastChatterDate);
    for each(var row:SObject in qr.records){
        if(row.CreatedDate > lastChatterDateString){
            //format a message to display in the ChatterBox component
            var message:String = row.message;
            message = (message as String).replace("<i>", "");
            message = (message as String).replace("</i>", "");
            message = (message as String).replace("<b>", "");
            message = (message as String).replace("</b>", "");
            message = (message as String).replace(new RegExp("<br/>", "g"), "\n");
            var toaster:Toaster = new Toaster(StaticAssets.OPPORTUNITY_IMAGE_32, "Opportunity Watcher", message);
            model.stratus.showToaster(toaster);
        }
    }

    //reset the Chatter last sync date/time
    lastChatterDate = new Date();
}
```

Once the toasters have been shown, the timer is reset and will activate the update again in another 20 minutes.

To test out this (and other) functionality, a number of test buttons are available in the code for MainView.mxml. By uncommenting these buttons, recompiling the application, and clicking the “Update Chatter” button, you can forego the 20-minute timer and call the update manually.

```
<!--Uncomment the buttons below for testing purposes-->
<s:Button label="Update Chatter"
          click="model.application.updateChatter()"/>
<s:Button label="Test Toaster Alert"
          click="model.stratus.showToaster(new Toaster(StaticAssets.CONTACT_IMAGE_32,
                    'New Contact', 'A new contact has been added: Dave Mansell'))"/>
<s:Button label="Go Offline"
          enabled="{model.stratus.connected}"
          click="model.stratus.salesforce_internal::forceConnected(false)"/>
<s:Button label="Go Online"
          enabled="{!model.stratus.connected}"
          click="model.stratus.salesforce_internal::forceConnected(true)"/>
```

Opportunity Watcher
Update Chatter | Test Toaster Alert | Go Offline | Go Online
Opportunities
Opportunity Name | Account | Close Date | Stage | Followed
Chatter
Dickenson East Installation
CloseDate was changed from 2010-06-21 to

4. Development Methods and Best Practices

As with any development framework, there are a number of recommended development methods and best practices that a developer should follow whenever possible, or at least be aware of. A few of these best practices for F3 are detailed in the sections below.

4.1. Incremental uploads versus batch uploads

After a user creates a new record or updates an existing record in an F3 application, it is updated in the local database, usually through calling EntityContainer's updateObject() method. It is not, however, uploaded to the Salesforce.com servers until the code specifically does so through F3DesktopWrapper's syncWithServer() method. Because this method accepts an array of DynamicEntity instances as its first parameter, it is possible to either upload each record as it is created or edited, or to save them all locally and provide the user with some kind of "Save All" button that uploads them in batch form.

It is recommended as a best practice to follow the first methodology by uploading to Salesforce every time a record is created or saved, (provided, of course, that the application is in online mode.) The advantages of batch saves (minimizing bandwidth and server loads) are outweighed by the benefits of keeping local application data in sync with server data. Doing so decreases the number of data conflicts, making the application more useful and efficient.

4.2. Creating records in offline mode

When creating records in offline mode, two important considerations must be taken.

The first is to always call F3DesktopWrapper's save() method on the record, even if EntityContainer's updateObject() method has already been called. If you do not call F3DesktopWrapper.save() for a newly created record, then F3DesktopWrapper.syncWithServer() will subsequently fail.

Secondly, records created offline cannot be used as parent records (as part of a lookup or master-detail relationship) until they have been uploaded. This is because they are not assigned a Salesforce.com ID until they are saved to the server, and this ID field is what is used in a lookup or master-detail relationship. Depending on the type of application you are developing and the frequency with which users are taking it offline, this restriction could be vital to the planning stages of your application.

4.3. Using the Connection class

The Connection class, a carryover from the standard Flex Toolkit, is included as a convenience in the F3 framework, to allow users to access advanced objects and describe metadata. However, it is a recommended best practice to use the F3DesktopWrapper class to perform CRUD operations on all data by default, especially that which you want to persist on the user's local machine.

By setting up the Offline Briefcase Configuration and using F3DesktopWrapper, a developer can easily extend standard and custom Salesforce.com functionality to the desktop. Connection should only be used when objects are not available through the Offline Briefcase Configuration, or data should not persist on the user's local machine.

It is also important to note that the Connection class cannot be used when the user is offline. For that reason, the NetworkStatusChangeEvent class should be used to listen for online/offline mode changes and enable or disable functionality appropriately. Alternately, values can be bound to the F3DesktopApplication.connected property.

Examining the Opportunity Watcher code

In Opportunity Watcher, the ChatterBox component's visible and includeInLayout properties are both bound to model.stratus.connected. This ensures that the Chatter feed is not displayed when the application is in offline mode. Since all Chatter functionality in Opportunity Watcher relies on the Connection class, and the Connection class does not work in offline mode, this is a necessity.

```
visible="{model.stratus.connected}"
includeInLayout="{model.stratus.connected}":
```

4.4. EntityContainer versus FieldContainer

When displaying a full layout of values for a Salesforce.com record in an F3 application, it is almost always best to use EntityContainer over FieldContainer whenever possible. This is because the standard layout for the object in question is automatically pulled down from Salesforce.com every time the F3 application begins, so users always see the latest layout changes that their Salesforce.com administrator has set up.

Using FieldContainer, or one or more instances of LabelAndField, can be useful if only a few key fields are needed, or a custom layout is desired. However, changes to these layouts must be made in the Flex code, and cannot be controlled by a Salesforce.com administrator without editing the code, republishing the F3 application, and updating users' installations of the application.

4.5. Fiber classes versus the DynamicEntity class

One important benefit of F3 is the ability to use Fiber to import enterprise WSDL from your Salesforce.com environment into the F3 project. This will automatically generate classes for each of your environment's objects, both standard and custom, that can be used as value objects in your code.

The alternative to this is to simply use DynamicEntity as the generic value object for all of your application's Salesforce.com records. This allows for greater flexibility and adaptability, but does not follow strict typing and does not ensure that a record has all of the appropriate fields.

Ultimately, there is no best practice recommended for this issue. It is largely left up to the developer's preference, as well as the requirements for a given F3 application's design and development.

4.6. Storing and accessing Salesforce.com records in application memory

When using F3DesktopWrapper to query records from Salesforce.com, they are automatically stored in a local SQLite database. It is also recommended you store them in application memory, usually in Array or ArrayCollection instances. That way, they can be called upon again without the need for an additional query.

For example, imagine an application that queries Account and Contact records. After reading the records from Salesforce.com, we can populate two ArrayCollections called accounts and contacts with the respective records. (In the Cairngorm framework, these would be stored in the ModelLocator class, so that we can access them from any other point in the application's code.) These are useful for populating charts, data grids, and other UI elements meant to display collections of data.

We can also use associative arrays to map each record to its ID. For example, after pulling down Account and Contact records and storing them in the accounts and contacts ArrayCollections, we can initialize two new Arrays called AccountsByID and ContactsByID. For each given record, we can then assign it to a location in the Array determined by its ID. For example, calling

AccountsByID[‘001A000000E6OOp’]

would return us the DynamicEntity instance for Burlington Textiles Corp.

Associative arrays can also help traverse relationship fields. For example, if Jack Rogers is a Contact for Burlington Textiles Corp., and his Contact ID is 003A000000D7pT5, then calling

AccountsById[ContactsByID[‘003A000000D7pT5’].AccountID]

would get us the DynamicEntity instance for Burlington Textiles Corp. With only one ID for one record, we can traverse its relationships as many times as we'd like, as long as each record has been indexed by its ID in an associative array.