# Force.com Flex Developer's Guide

Version 1, 15.0

’21

@salesforcedocs

Last updated: November 12, 2020

# CONTENTS

# CHAPTER 1 Getting Started with Flex

In this chapter ...



Flex is a framework for creating Flex-based desktop and Web applications that leverage Salesforce logic, data, and security. The framework provides:

- Seamless handling of connected and disconnected states for desktop apps, including the detection of Internet connectivity
- Adobe's Data Management Services (DMS), which creates and manages local databases for desktop applications that operate online and offline
- Data synchronization between Salesforce and desktop applications
- Generated ActionScript classes that mirror your Salesforce enterprise WSDL and provide access to Salesforce objects
- MXML components to simplify the implementation of user interfaces that render Salesforce data, Flex status bars, and popup notifications called *toasts*
- An engine and user interface for resolving data conflicts and errors in desktop applications

The Flex framework installs as a standalone instance of Adobe® Flash® Builder for Lightning Platform—an enhanced version of Adobe's Eclipse-based integrated development environment (IDE) for developing Flex applications. Adobe Flash Builder for Lightning Platform includes classes that facilitate the development of Flex-based applications for Salesforce. These applications create, update, and delete Salesforce data via asynchronous requests to the Lightning Platform API. Mobile and tablet applications created using Flash Builder version 4.5 or newer aren't supported.

Note: Adobe Flash Builder for Lightning Platform also includes the previously-released Flex Toolkit for Lightning Platform, which provides direct access to the Lightning Platform API.

# Audience

Flex is for developers with a working knowledge of the API and the following Adobe technologies:

**ActionScript**
Adobe's scripting language for developing websites and software.

**Adobe Integrated Runtime (AIR)**
Adobe's cross-platform runtime environment for building desktop applications. Developers deploy Flex desktop applications as AIR files.

**Flash**
Adobe's multimedia platform for creating visually enhanced Web pages. Developers deploy Flex Web applications as Flash files, which run in browsers with the Adobe Flash Player.

**Flash Builder 4**
Adobe's Eclipse-based integrated development environment (IDE).

**Flex**
Adobe's free, open source framework for building dynamic Web applications that deploy consistently across browsers, desktops, and operating systems by leveraging the Adobe Flash Player and Adobe AIR runtimes.

**MXML**
Adobe's XML-based user interface markup language.

For information on the API, see *SOAP API Developer Guide*. For information on Adobe technologies, see `www.adobe.com`.

# Configuring Salesforce for Flex Development

Before developing Flex applications, log into Salesforce and do the following:

- Access your personal information page and select the `Offline User` permission. See *Editing Your Personal Information* in the Salesforce online help.

  **Important:** Anyone who develops or uses a Flex application must have the `Offline User` permission.

- Create an offline briefcase configuration that includes the Salesforce objects in your Flex application; assign the configuration to the Salesforce users who will use your application. See *Lightning Platform Connect Offline Overview* in the Salesforce online help.
- Generate and save a local copy of your Salesforce enterprise WSDL so you can import it into your Flex environment and generate ActionScript classes according to your Salesforce data model.

# Installing Flex

Flex installs in a separate Eclipse environment. You can't access Flex from instances of Eclipse other than the one that installs with Flex. Flex doesn't affect existing Eclipse installations.

To install Flex:

1. Verify that your machine meets these system requirements:
   - One of the following operating systems:
     - Windows XP Pro SP2, SP3
     - Windows Vista–32 bit

  - Mac OS 10.5.6 (Leopard) Intel only; PPC not supported
  - Mac 10.6 (Snow Leopard)
- One of the following JAVA Runtime Environments:
  - Windows—Sun 1.5 and 1.6. (32 bit), IBM 1.5 and 1.6 (32 bit)
  - Mac OS X—Sun 1.5 (32 bit)
- 1 GB of memory (2 GB recommended)

2. Go to `developer.salesforce.com/flashbuilder`.
3. Download the appropriate Flex installer for your operating system.
4. Run the installer. It displays a message when finished.

Warning:

- The Flex installer invokes a separate Adobe Flash Builder installer. Let both installers complete without interruption.
- The Adobe Flash Builder installer for Windows runs in the background and may appear to pause without progress for up to 30 minutes.
- If you need to reinstall Flex, first uninstall it. If you run the installer twice on the same machine without uninstalling the first instance, the installation fails.
- Don't change the locale in the installer. The Flex Mac installer provides an option to change your locale, but doing so causes an error when you build an AIR package.

# Additional Resources

## Flex Quick Start Tutorial

The *Flex Quick Start Tutorial* walks you through the process of building a basic desktop application with Flex. It provides all the code you need plus information on configuring Salesforce for use with Flex, installing Flex, and more.

## Built-In API Reference Documentation

ASDoc is an Adobe command-line tool that creates HTML versions of API language reference documentation, such as the *Adobe Flex Language Reference*, from the classes in Flash Builder. Flex leverages ASDoc to create API reference documentation for Flex. The documentation includes descriptions of every standard Flex class, including its properties and methods.

Relevant snippets of the Flex API reference documentation appear contextually as you code. To access the entire reference, click the ASDoc tab in Adobe Flash Builder or view it online at Salesforce.

## Sample Flex Projects and Applications

Sample Flex projects and applications are available at Salesforce.

# What's New Since the Developer Preview?

The following table shows how this release of Flex differs from the developer preview release, referred to as *Adobe Flash Builder for Force.com*.

| Developer Preview | Current Version |
| --- | --- |
| Framework named Stratus | Framework named Flex |
| No support for Web applications | Supports Web applications |
| Applications cannot dynamically detect and handle schema changes | Flex applications can dynamically detect and handle schema changes |
| Applications can only contain Fiber-generated classes | Flex applications can consists of Fiber-generated classes or be completely dynamic |
| No user interface component for rendering Salesforce detail or edit pages | `EntityContainer` user interface component renders Salesforce detail and edit pages |
| Applications must be contained within a top-level component | Applications are contained inside a standard Flex component, making them easier to integrate into other applications |
| Users must handle data conflicts and errors using the built-in data conflict and error resolution interface | Developers can programmatically handle data conflicts and errors without the built-in interface |
| Users must handle log in using the built-in log in interface | Developers can programmatically log in users by passing session IDs or user credentials |
| SOQL queries support only lowercase keywords | SOQL queries in Flex desktop applications are case-insensitive. |

# CHAPTER 2 Using Flex

In this chapter ...



Installing Flex creates a Flex project type in Adobe Flash Builder. Use this type of project to create Flex applications.

Flex projects are Flex projects with additional ActionScript classes that add the following functionality to Flex desktop applications:

- Data storage and synchronization
- Online and offline detection
- Data conflict and error resolution
- Status bars
- Toaster alerts

Flex desktop applications projects also have ActionScript classes and MXML components that let you add user interface components with the Salesforce look and feel to your both desktop and Web applications.

As with other Flex projects, Flex projects let you import your Salesforce enterprise WSDL as a data service. When you import your WSDL, Flex:

- Generates ActionScript classes for every standard and custom object in your Salesforce WSDL
- Renders a model of your WSDL that you can visually introspect
- Provides code hinting and compiler warnings in Adobe Flash Builder based on your WSDL

To create a Flex project:

1. Launch Flex.
2. Select **File** > **New Project**.
3. Open the Flash Builder folder, select `Flex Project`, and click **Next**.
4. Enter a project name, and select `Desktop (runs in Adobe AIR)` to create a desktop application or `Web` to create a browser-based Flash application. Click **Next**.
5. Choose a folder for your application output and click **Next**.
6. Fill out the fields as you would for a standard Flex project, and click **Finish**.

# Standard Flex Classes

Flex includes classes that facilitate the development of desktop and Web applications. The built-in API documentation that installs with Flex contains details about each class, including its properties and methods. The following table provides a high-level description of each class, and indicates if the class if for use in desktop applications, Web applications, or both.

| Class | Description | Desktop Only | Web Only | Desktop and Web |
|---|---|---|---|---|
| `BaseApplication` | The base implementation of a Flex desktop and Web application. | | | X |
| `BaseContext` | The base implementation class for context. | X | | |
| `ComponentType` | The enumeration that identifies the layout component type. | | | X |
| `ConflictContext` | The context that the conflict handler uses in `IF3ErrorHandler`. | X | | |
| `ConnectionStatus` | The widget that reports on the connection status of a desktop application. | X | | |
| `Dispatcher` | The class that passes messages from components. | | | X |
| `DynamicEntity` | The representation of a dynamic entity, which supports a dynamic set of properties. | | | X |
| `EntityContainer` | The representation of an entire Salesforce detail or edit page. | | | X |
| `ErrorContext` | The context that the error handler uses in `IF3ErrorHandler`. | X | | |
| `F3DesktopApplication` | The Flex desktop application container. | X | | |
| `F3DesktopWrapper` | The class that manages the interactions between Salesforce and the local database. | X | | |
| `F3Message` | The standard contextual error and information messages for Flex applications. | | | X |
| `F3WebApplication` | The Flex Web application container. | | X | |
| `F3WebWrapper` | The class that manages the create, update, and delete interactions between Flex Web applications and Salesforce. | | X | |

| | | | |
|---|---|---|---|
| `FieldCollection` | The container for a collection of `FieldElement` and the methods that work with them. | | X |
| `FieldContainer` | The visual container for the contents of a `FieldCollection`. | | X |
| `FieldElement` | The representation of a Salesforce field value without its label. | | X |
| `FieldErrors` | The errors returned by the `validate()` method. | | X |
| `IF3Application` | The interface that provides the contract for Flex applications. | | X |
| `IF3ErrorHandler` | The interface that provides the contract for handling errors and conflicts. | X | |
| `IToaster` | The interface that toaster classes implement. | X | |
| `LabelAndField` | The representation of a Salesforce field value and its label. | | X |
| `LayoutSection` | The enumeration that identifies the layout for rendering or building queries. | | X |
| `LoginFaultEvent` | The event that fires when a login fails. | | X |
| `LoginResultEvent` | The event that fires when a login is successful. | | X |
| `MetadataUtil` | The utility that provides access to metadata and functions. | | X |
| `NetworkOperationEvent` | The event that indicates if a create, update, or delete operation is in progress. | X | |
| `NetworkStatusChangeEvent` | The event that indicates whether a desktop application is online or offline. | X | |
| `PageStyle` | The utility class for page style data. | | X |
| `SessionExpiredEvent` | The event that fires when a Salesforce session expires. | | X |
| `StatusBar` | The desktop application status bar on page 12. | X | |
| `StatusChangedEvent` | The event that fires when the status bar status message changes. | X | |
| `Toaster` | The contents of a toaster alert. | X | |
| `ToasterEvent` | The event that fires when a toaster alert displays. | X | |

# WSDL Introspection and Code Generation with Fiber

Fiber is an Adobe Flash Builder feature that generates ActionScript code for remote objects and, if necessary, for remote method invocations. Fiber creates an application model that lets developers write applications at a higher level, reduces the amount of code necessary to develop applications, and simplifies data integration.

Flex leverages Fiber to import your Salesforce enterprise WSDL into your development project. Fiber then generates ActionScript classes for all of your Salesforce standard and custom objects, and exposes the properties as variables with getters and setters. Each property is bindable, meaning your application can listen for changes to its value and receive notification through a `PropertyChange` event.

To import your WSDL:

1. Generate your enterprise WSDL in Salesforce and save a copy.
2. In Flash Builder, click **Data** > **Connect to Data/Service**.
3. Choose `Salesforce` and click **Next**.
4. Navigate to your WSDL and click **Finish**.

To view the generated ActionScript classes, open your Flex project in the Flash Builder Package Explorer, and navigate to the `services.flexforforce` package under `src`.

To introspect the Fiber model, expand the DataTypes node in the Flash Builder Data/Services tab.

To use a Fiber-generated class in your code, import it and add at least one explicit definition, such as a variable or parameter, in your code. The Flex compiler removes unused classes, so the Fiber-generated class is removed if it is only generated at run time.

Note:

- Flex projects can't import WSDLs other than the Salesforce enterprise WSDL.
- After importing your enterprise WSDL, changes to your Salesforce metadata don't automatically appear in Flex. If you add, remove, or update an object, you must re-import your WSDL to see the changes in Flex.
- Generating the Fiber model and ActionScript classes is optional—you can build Flex apps without Fiber code generation. See Dynamic Data Model Support on page 8.
- You can't use Flex to modify your Salesforce metadata.

# Dynamic Data Model Support

A Flex Web application bases its data model on your Salesforce global metadata, while a Flex desktop application bases its data model on a subset of Salesforce metadata specified in an associated offline briefcase configuration. Flex implements these data models in ActionScript.

ActionScript classes, including the Flex Fiber-generated ActionScript classes, are dynamic, meaning that the properties of a class can change during runtime. As a result, Flex data models are dynamic—they can change during runtime to reflect changes in your Salesforce metadata. For example, if you add a `Social Security Number` field to the Contact object after deploying a Flex desktop application, the application can get an instance of the Contact class, and get or set the property `contact.SocialSecurityNumber` even though the Contact class compiled without this property.

In Flex Web applications, changes to the data model appear instantly. In Flex desktop applications, a popup message alerts users when the Salesforce metadata changes. Upon restarting the application, users have the option to fully synchronize with Salesforce. A full synchronization involves recreating the database and other operations that make the application is unavailable until the synchronization completes.

Note:

- While Flex applications update their data models to reflect Salesforce metadata changes, the Flex environment doesn't automatically update its Fiber-generated ActionScript classes. To update your Flex environment, re-import your Salesforce enterprise WSDL.
- Salesforce metadata changes only affect Flex desktop applications if the metadata change involves an object in the offline briefcase configuration associated with the Flex desktop application.
- If you remove a field from Salesforce and the code for your Flex application explicitly accesses the field, the Flex application sets the value to null without throwing an exception or warning the user.

## DynamicEntity Class

Flex includes a `DynamicEntity` class that lets you get and set any property at runtime. The class is similar to the Salesforce `sObject` class, but has the annotations necessary for Adobe Data Management Services (DMS).

Flex desktop applications can use the `DynamicEntity` class to create an empty object or receive data from an API request, allowing you to write code that works with entity types not present at compile time. For example, the `DynamicEntity` class can represent objects you add to an offline briefcase configuration after compiling a Flex desktop application.

When a query returns an object, Flex checks the code for a Fiber-generated version of the object. If the code contains a Fiber-generated version, Flex converts the object to that Fiber type; otherwise, Flex converts the object to a `DynamicEntity`.

The following table describes how the `DynamicEntity` class differs from Fiber-generated ActionScript classes:

| **Fiber-generated objects...** | `DynamicEntity` **Classes...** |
|---|---|
| Have predefined fields | Don't have predefined fields |
| Doesn't allow the deletion of fields; instead, sets fields to their default values | Allow the deletion of fields |
| Assigns a value to every field when you do a query | Returns only the fields you specify |

Important: When querying Fiber-generated objects, select all fields using `SELECT *`. If you do not use `SELECT *`, fields within the `SELECT` clause are set to their specified values while all other fields are set to their default values. When saving the object, these default values may override the values on the server.

# Developing User Interfaces for Flex Applications

Flex includes the following MXML components to facilitate the development of desktop and Web application user interfaces with the Salesforce look and feel:

- `FieldElement`
- `LabelAndField`
- `FieldContainer`
- `EntityContainer`

Flex also includes classes that let you easily add a status bar and toaster alerts to your desktop applications.

## FieldElement

The `FieldElement` component renders the value of a Salesforce field without its label. The value appears and functions the same way it does in Salesforce—a field that's a checkbox in Salesforce appears as a checkbox in Flex applications, date fields display a data picker when clicked, lookup fields let you choose related records, and so forth. The values also support field dependencies, field format validation, hover details, error notification, and information icon ( ) help text.

Users can edit the value the same way they do in Salesforce with inline editing enabled. When hovering over the value, a pencil icon ( ) appears if the value is editable, and a lock icon ( ) appears if the value is read-only. Double-clicking the value changes it to edit mode, and clicking after you change a value reverts it.

Note: The values that `FieldElement` components render differ from Salesforce in the following ways:

- In Flex, the values of compound fields, like addresses, first and last names, and dependent picklists appear in a dialog box when you edit them.
- The `FieldElement` component doesn't support field-level security. `EntityContainer` is the only Flex component that supports field-level security.
- Flex validates field formats but doesn't support validation rules. Salesforce runs validation rules on the data when Flex synchronizes.

To indicate which value the `FieldElement` component renders, pass a fully-qualified field name in the following format:

```
Object.Field
```

For example, the fully qualified name for the account `Website` field is

```
Account.Website
```

The following MXML demonstrates how to use the `FieldElement` component to include the value of the account `Website` field in a user interface:

```
<flexforforce:FieldElement id="_fieldElement" field="Account.Website" />
```

The following ActionScript code renders the actual value of the `Website` field provided that "account" is an instance of the Account object.

```
_fieldElement.render(account);
```

The field type of the `Account.Website` field in Salesforce is a hyperlink, so the Flex application automatically renders the value as a hyperlink instead of plain text.

Important: Field names are case-sensitive. Using the incorrect case results in an error that reads `Invalid fully qualified name.`

## LabelAndField

The `LabelAndField` component is exactly like the `FieldElement` component, only it renders the field label to the left value. If the label is translated in Salesforce, the translated version also appears in the Flex application.

Note: Flex doesn't support the localization of dates, currency, and numbers, nor does it support right-to-left languages.

The following MXML demonstrates how to use the `LabelAndField` component to include the label and value of the account `Website` field in a user interface:

```
<flexforforce:LabelAndField field="Account.Website" />
```

**LabelAndField**

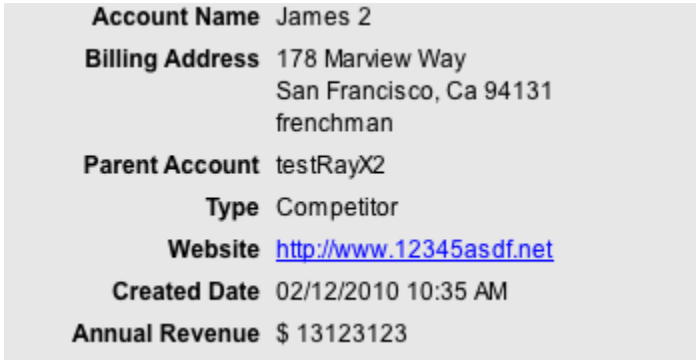

## FieldContainer

The `FieldContainer` component provides a way to collectively manage `FieldElement` and `LabelAndField` components that render data from the same object. Developers can simply call render or isValid on a `FieldContainer` instead of managing each individual `FieldElement` and `LabelAndField` component.

For example, the following MXML demonstrates how to create a container for the `Name`, `Parent ID`, `Type`, `Website`, and `Annual Revenue` fields of the Account object.

```
<flexforforce:FieldContainer id="_editFieldContainer" width="100%">
    <flexforforce:LabelAndField field="Account.Name" />
    <flexforforce:LabelAndField field="Account.BillingStreet[Group]" />
    <! -- Account.BillingStreet is a virtual field that expands to include all of the
subfields in a billing address -->
    <flexforforce:LabelAndField field="Account.ParentId" />
    <flexforforce:LabelAndField field="Account.Type" />
    <flexforforce:LabelAndField field="Account.Website" />
    <flexforforce:LabelAndField field="Account.CreatedDate" />
    <flexforforce:LabelAndField field="Account.AnnualRevenue" />
</flexforforce:FieldContainer>
```

**Field Container**




## EntityContainer

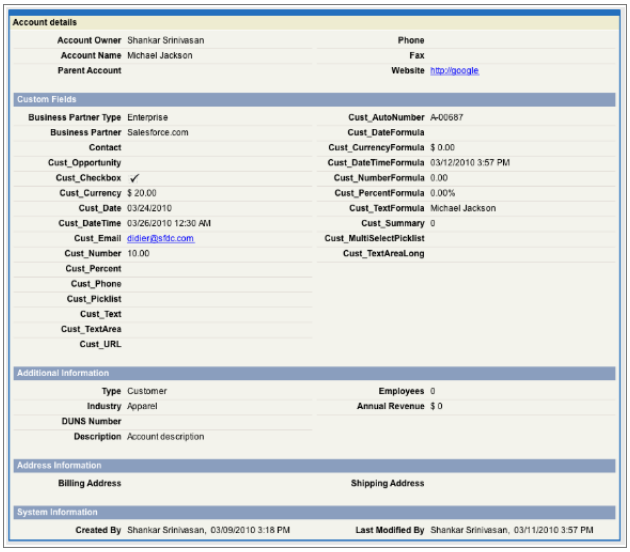
The `EntityContainer` component renders the entire detail or edit page of a Salesforce record, including its section headers, and manages the `FieldElement`, `LabelAndField`, and `FieldContainer` components that the `EntityContainer` includes. The `EntityContainer` uses layout metadata from your Salesforce enterprise WSDL to determine the layout of these components.

The following MXML sets up the `EntityContainer`:

```
<flexforforce:EntityContainer id="_editFieldContainer" width="100%">
```

The MXML doesn't specify the object. Instead, after setting the `EntityContainer`, call the Render method with any object to render the detail page for that object. For example, calling `render()` with an Account object renders the account detail page.

**EntityContainer**




The following ActionScript shows how to bind data to the `EntityContainer` and commit the data to the database:

```
private function onEditSaveClick() : void {
_editFieldContainer.fieldCollection.updateObject( new mx.rpc.Responder( commitToDB,
// The existing entity has been updated in memory. Commit it to the database.
saveFaultHandler ));
}
```

The `EntityContainer` component can render detail and edit pages for an object regardless of whether it's a Fiber-generated class or a `DynamicEntity` class.

**Important:** When using the `EntityContainer` component to render the edit page of an object that a `DynamicEntity` class represents, the Flex application saves a value for every field on the object. If the field exists on the object but not in the `DynamicEntity` class, the Flex application adds the field and its value to the `DynamicEntity` class, even if the user didn't change the value of the field.

The `EntityContainer` component doesn't support record types or related lists.

## Status Bar

Flex desktop applications can include a status bar that displays information according to intercepted events, such as programmatic responses. For example, the status bar can notify users when the application is online, or it can indicate the number of data conflicts and errors between the desktop application and Salesforce.

**Status Bar**

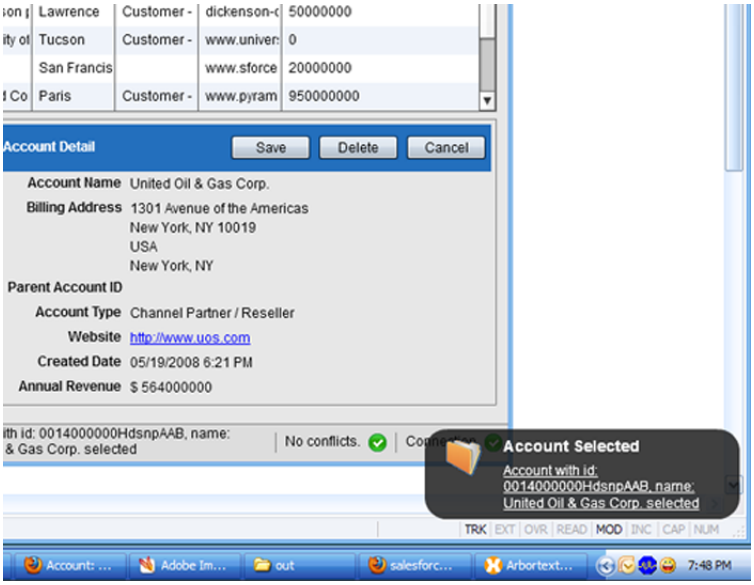

To implement a status bar, use the following MXML component:

```
<flexforforce:StatusBar/>
```

## Toaster Alerts

Flex desktop applications can include *toaster alerts*—popup messages that typically appear in the System Tray on Windows operating systems or the Dock on Mac OS X. Toaster alerts can inform users when actions occur in either the desktop application or Salesforce. For example, you can use toaster alerts to let users know when an opportunity closes or account information changes.

**Toaster Alert**




To implement a toaster alert, use the following ActionScript:

```
var toaster : Toaster = new Toaster( StaticAssets.REPORT_IMAGE_32, "Record deleted",
status.description );
```

# CHAPTER 3 Developing Flex Desktop Applications

In this chapter ...



Flex desktop applications run on your workstation or laptop using Adobe® Integrated Runtime (AIR®), and operate both online and offline. The ability to operate offline makes them ideal for users who don't always have an Internet connection but still need to access Salesforce data. For example, traveling sales teams can use Flex desktop applications to update contacts, opportunities, and accounts while visiting customer sites.

Every Flex desktop application is contained within an `F3DesktopApplication` component. This component connects the Flex desktop application to Salesforce, provides Salesforce login and authentication functionality, and handles the initial data synchronization between Salesforce and the Flex application.

```
        <flexforforce:F3DesktopApplication />
```

To implement a Flex desktop application, use the `F3DesktopWrapper` class:

```
private var _wrapper : F3DesktopWrapper =
F3DesktopApplication.getInstance().wrapper;
```

**Important:** Before logging into a Flex desktop application that's offline, a user must log into the application online at least once and synchronize the application with Salesforce.

# Making Salesforce Data Available Offline

A Flex desktop application can only access the data the user's offline briefcase configuration includes. An *offline briefcase configuration* is a set of parameters in Salesforce that controls which records users can access offline. Organizations can have multiple briefcase configurations and assign them to different profiles and users to simultaneously suit the needs of various types of offline users. For example, one configuration might have leads and opportunities for users with a Sales Representative profile, while another configuration has accounts and related opportunities for users with an Account Executive profile.

Important: Offline briefcase configurations limit the number of records client applications (including Flex desktop applications) can download from Salesforce. This limit can vary.

If a Flex desktop application needs access to objects that the current user's briefcase does not include, the application might not function correctly. To prevent this from happening, specify the objects the application needs in the `requiredTypes` attribute on the `F3DesktopApplication` component. The Flex desktop application verifies it can access the objects specified in the `requiredTypes` attribute before launching the applications.

# Storing and Synchronizing Offline Data

When offline, Flex desktop applications store data in local databases. Flex creates a different database for each user and names the database after the user's Salesforce user ID. This allows multiple users to log in to the same Flex desktop application installation without accessing other users' data.

When online, Flex desktop applications can exchange data with Salesforce using the following methods:

- `fetchChangesFromServer`—Retrieves data from Salesforce.
- `syncWithServer`—Commits local data to Salesforce and, if there is data to commit, also attempts to retrieve data from Salesforce.

Tip: Salesforce may limit the number of times an application can call `fetchChangesFromServer()` and `syncWithServer()`. To accommodate these possible limitations, call these methods at intervals greater than 20 minutes.

Flex desktop applications use Adobe's Data Management Services (DMS) to manage all interactions with the local database, including synchronization with Salesforce. DMS has ActionScript APIs that perform standard create, update, and delete capabilities and facilitate interacting with online and offline data.

Note: Records you save in a Flex application may take a few minutes to appear in Salesforce.

# Detecting Internet Connectivity and Web Services

Flex desktop applications automatically detect both Internet connectivity and the availability of Web services. If the Internet or Web service becomes unavailable, Flex desktop applications route create, update, and delete operations to the local database. The operation executes when the Internet or Web service becomes available again, providing a seamless online and offline experience for users.

Note: Flex desktop applications can take up to ten seconds to detect a change in Internet connectivity.

# Accessing Data in Flex Deskop Applications

To access data in Flex desktop applications, use the `query()` and `getItemById()` methods of the `F3DesktopWrapper` class. The `F3DesktopWrapper` class facilitates all interactions between the Flex desktop application and DMS.

The `F3DesktopWrapper` class also has methods that interact with the Salesforce server. Some of these methods, such as `fetchChangesFromServer()` and `syncWithServer()`, interact directly with Salesforce. Other methods, such as `save()` and `deleteItem()`, interact indirectly with Salesforce. All of these methods have responders. Use the responders to determine the status of the server operation (result or fault), and when the operation completes.

When designing how your Flex desktop application accesses data, consider the following:

- Each Salesforce organization supports up to 500 object synchronizations per 24-hour period. For example, if your offline briefcase has 10 objects, the desktop application can synchronize 50 times in a 24-hour period. If your offline briefcase has 1 object, the desktop application can synchronize 500 times in a 24-hour period. If the desktop application hits the limit for your organization, Salesforce returns a `REQUEST_LIMIT_EXCEEDED` error and delays the synchronization until the 24-hour period ends.
- Attempting a large number of simultaneous server interactions without waiting for operations to complete can put the application in an error state. So, wait for responder results before continuing execution.
- Calling `syncWithServer()` with a `null` parameter sends all pending local changes to Salesforce. Alternatively, you can pass an array of specific items to synchronize.
- Saving more than 400 records at a time can cause performance issues.
- Divide your batch synchronizations into reasonable sizes to avoid one large operation. Transferring a thousand records from Salesforce to your Flex application (`fetchChangesFromServer`) can take up to a minute. Creating or modifying a thousand records on the server (`syncWithServer`) can take up to ten minutes.
- To load a large number of records, consider using the Data Loader. See the Salesforce online help for information on loading records.
- If a Flex application crashes, you may need to delete its local database and re-synchronize with Salesforce.
- Your Salesforce account may have limits on the number of API calls allowed in a 24–hour period. If your Flex application performs a large number of save or delete calls, you may exceed that limit.

# Querying Data in Flex Deskop Applications

Flex desktop applications execute all queries offline, and convert SOQL to SQL before querying the database. Data that the Flex desktop application doesn't fetch from Salesforce isn't available to query.

DMS manages results from queries, and listens for changes to the entities. If you delete an entity, DMS queues a delete. If you change an entity field, DMS queues up an update.

Flex desktop applications support the following SQL keywords in queries:

- SELECT
- FROM
- ORDER BY
- LIMIT
- WHERE

Keywords can be lower and upper case.

Flex desktop applications only support the use of SOQL SELECT queries without joins and subqueries. The following are examples of the SOQL SELECT queries that Flex desktop applications support:

- `SELECT Name FROM Account`
- `SELECT LastName, CreatedDate FROM Contact WHERE FirstName='fred'`
- `SELECT * from Account`

Flex desktop applications don't support SOQL functions or nested SOQL SELECT queries, such as:

```
SELECT count() FROM Contact
SELECT Name, (SELECT LastName FROM Contacts) FROM Account
```

Call `releaseQueryResults()` when you're done with the results of a query.

# Resolving Data Conflicts and Errors in Flex Desktop Application

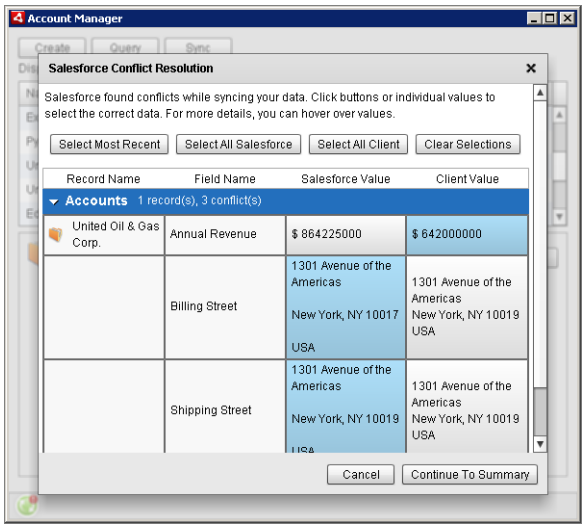
When a Flex desktop application synchronizes with Salesforce, values in the desktop application might violate validation rules or conflict with changes by other users in Salesforce. To resolve such issues, the Flex framework includes data conflict and error resolution functionality in desktop applications.

In Flex desktop applications that use the standard, out-of-the-box data conflict and error resolution functionality, the status bar displays the number of unresolved conflicts or errors that occur when synchronizing data. The application also renders a **Resolve** button when a conflict or error occurs. Users can click the button to launch an interface that lets users quickly resolve data issues in Flex desktop applications.

If data conflicts exists, the interface displays the Salesforce values and their conflicting Flex application values in a table. Users can resolve each conflict by selecting the value to keep in either the `Salesforce Value` column or `Client Value` column. Alternatively, buttons at the top allow users to quickly select the most recent values, all Salesforce values, or all client values. Users can hover over values for details.

If the Flex desktop application has data errors to resolve, the interface lists the fields with errors and their values. Users can double-click the incorrect value and enter the correct one.

**Standard Conflict Resolution Interface**




The following code shows the standard functionality for handling data conflict and errors. The code has a class that implements the `IF3ErrorHandler` interface, and registers the class using the `errorHandler` property on the `F3DesktopWrapper`.

```
// Create a class that implements the IF3ErrorHandler for handling errors and conflicts.

import com.salesforce.data.ConflictContext;
      import com.salesforce.data.ErrorContext;
      import com.salesforce.data.IF3ErrorHandler;
```

```
      public class F3ErrorHandler implements IF3ErrorHandler
      {

            public function handleConflict(conflictContext:ConflictContext):void {

                  conflictContext.raiseConflict();

            }

            public function handleError(errorContext:ErrorContext):void {
                  errorContext.raiseError();
            }
      }


// Register your class in the F3DesktopWrapper.

var handler:F3ErrorHandler  = new F3ErrorHandler();
F3DesktopApplication.getInstance().wrapper.errorHandler = handler;
```

When a conflict occurs, Flex calls the `handleConflict` method with a `ConflictContext` object. `ConflictContext` contains the server version of the item (record) and conflicting fields. It also contains the `raiseConflict` method, which adds the conflict to a list you can access through the `F3DesktopWrapper.conflicts` property.

When an error occurs, Flex calls the `handleError` method with an `ErrorContext` object. The `ErrorContext` object contains the operation (create, update, or delete) that caused the error, and the error message, which contains the fault code, description, and details about what caused the error. It also contains the `raiseError` method, which adds the conflict to a list you can access through the `F3DesktopWrapper.errors` property.

If you want to undo the changes in the desktop application that cause the conflicts or errors, call `revertChangesOnItem`.

If you do not want to use the standard Flex functionality, you can use the classes to implement your own data conflict and error resolution functionality. For example, you can create a custom user interface for resolving the data, or configure your desktop application to automatically override conflicting values with the values in Salesforce if a certain number of conflicts exist.

The following sample code shows how to automatically accept Salesforce values if the object is an account while displaying conflicts for all other objects.

```
public function handleConflict(conflictContext:ConflictContext):void {
   if (conflictContext.localItem() is Account) {
      // Always select the values on the server for account conflicts
      conflictContext.revertChanges();
   }
   else {
      // Raise a standard conflict for objects that aren't accounts
      conflictContext.raiseConflict();
   }
}
```

# CHAPTER 4 Developing Flex Web Applications

In this chapter ...



Flex Web applications run in browsers with Adobe® Flash® Player, and can operate as standalone Web applications or as embedded user interface components on Salesforce pages. For example, Flex Web applications can render animated charts and graphs based on Salesforce data.

To implement a Flex Web application, use the `F3WebWrapper` class:

```
private var _wrapper : F3WebWrapper =
F3WebApplication.getInstance().wrapper;
```

Like standard Web applications, Flex Web applications operate in the browser and with the expectation that an Internet connection is always available. They don't support offline and desktop application features, such as:

- Data caching
- Data conflict and error resolution
- DMS and local databases
- Internet connection detection
- Native events
- Native windows
- Toaster alerts

Also, Flex Web applications don't use offline briefcase configurations. Instead, use the `RequiredTypes` attribute on the `F3WebApplication` component to indicate which objects the application requires.

# Accessing Data in Flex Web Applications

To access data in Flex Web applications, use the `query()` method in the `F3Wrapper` interface implemented by the `F3WebWrapper` class. The `F3WebWrapper` class handles all interactions with the database.

In Flex Web applications, `query()`, `save()`, and `deleteItem()` operations directly access the API. The direct access enables Flex Web applications to use the full SOQL syntax; however, Flex Web applications don't support the `*` shortcut.

# Adding Flex Web Applications to Visualforce Pages

To include a Flex Web application on a Visualforce page, upload the application to Salesforce as a static resource, and use the `<apex:flash>` component to reference the static resource form a Visualforce page. For example, to include the fiscalprojector static resource on a Visualforce page:

```
<apex:flash src="{!$Resource.fiscalprojector}" width="1200" height="600"/>
```

Use the `flashvars` attribute on the `<apex:flash>` component to pass in the user's session ID. For example:

```
<apex:flash src="{!$Resource.fiscalprojector}"
flashvars="sid={!MySessionId}&surl={!$Api.Partner_Server_URL_90}"/>
```

HTTP/SOAP server sessions expire if there is no activity. Flex applications detect the expiration of HTTP/SOAP server sessions only when synchronizing or committing data. When a session expires, Flex applications display the login screen. The user must log in to continue using the application. See the *SOAP API Developer Guide* for information on session expiration and session IDs.

# APPENDIX A Troubleshooting

| Problem | Solution |
| --- | --- |
| Flex desktop not displaying the correct objects. | Re-import your Salesforce enterprise WSDL. |
| Flex desktop application not showing data or showing incorrect data | Delete the local database from `%appdata%\<Flex project>\Local Store` (in Windows) or `$HOME/Library/Preferences/<Flex project>/Local Store` (in Mac OS X), then synchronize the Flex desktop application again. |

# APPENDIX B Known Issues

**Relationship Fields**

Flex desktop applications support relationship fields; however, you can't relate records created offline until you synchronize them with Salesforce. This is because Salesforce bases relationships on IDs that records receive only after you upload them to Salesforce.

**Record Types**

Flex doesn't support record types.

**Localization**

Flex doesn't support the localization of dates, currency, and numbers, nor does it support right-to-left languages. However, the `FieldElement` and `LabelAndField` components support translated text.

**Field-Level Security**

The `FieldElement` and `LabelAndField` components do not support field-level security. The `EntityContainer` component is the only Flex component that supports field-level security.

# INDEX