



Version 29.0: Winter '14

# Force.com Canvas Developer's Guide



Last updated: January 3, 2014



# Table of Contents

<b>GETTING STARTED.....</b>	<b>1</b>
<b>Chapter 1: Introducing Force.com Canvas.....</b>	<b>1</b>
Force.com Canvas Scenarios.....	2
Where Canvas Apps Appear.....	2
Supported Browsers.....	3
Supported Salesforce Editions.....	3
User Permissions Required.....	4
User Interface Considerations.....	4
Canvas App Process.....	5
<b>Chapter 2: Quick Start.....</b>	<b>7</b>
Prerequisites.....	8
Create the App.....	8
Set the App Location.....	9
<b>Chapter 3: Quick Start—Advanced.....</b>	<b>11</b>
Prerequisites.....	12
Clone the Project from GitHub.....	12
Run the Web App Locally.....	13
Create the Canvas App.....	14
Configure Who Can Access the Canvas App.....	16
Deploy the Web App to Heroku.....	16
Update the Canvas App.....	18
Package the Canvas App.....	19
Upload the Canvas App Package.....	19
Install the Canvas App.....	20
Configure Who Can Access the Installed Canvas App.....	20
<b>USING FORCE.COM CANVAS.....</b>	<b>22</b>
<b>Chapter 4: Force.com Canvas SDK.....</b>	<b>22</b>
Referencing the Force.com Canvas SDK.....	23
Authentication.....	23
Signed Request Authentication.....	23
OAuth Authentication.....	29
Getting Context in Your Canvas App.....	32
Cross-Domain XHR.....	32
Getting a List of Chatter Users.....	33
Posting to a Chatter Feed.....	33
Resizing a Canvas App.....	34
Automatically Resizing a Canvas App.....	34
Explicitly Resizing a Canvas App.....	34
Getting the Size of a Canvas App.....	35

Subscribing to Parent Events.....	35
Implementing Canvas App Events.....	36
Canvas App Events Considerations.....	36
Creating a Canvas App Event.....	37
Subscribing to a Canvas App Event.....	38
Unsubscribing from a Canvas App Event.....	38
Using Streaming API in a Canvas App.....	39
Using the Streaming API Event.....	39
Subscribing to a Streaming API Event.....	40
Unsubscribing from a Streaming API Event.....	41
<b>Chapter 5: Canvas Apps and Visualforce Pages.....</b>	<b>42</b>
Visualforce Page Code Examples.....	43
Visualforce Considerations.....	44
apex:canvasApp Component.....	44
<b>Chapter 6: Canvas Apps in the Publisher—Pilot.....</b>	<b>46</b>
Set Canvas App Location and Create the Action.....	47
Create the Action.....	47
Force.com Canvas SDK Publisher Events.....	48
Publisher Context Considerations.....	49
Publisher Canvas App Access Considerations.....	50
<b>Chapter 7: Canvas Apps in the Chatter Feed—Pilot.....</b>	<b>51</b>
Chatter Feed Context Considerations.....	52
Chatter Feed Canvas App Access Considerations.....	52
<b>Chapter 8: Canvas in the Salesforce1 App—Pilot.....</b>	<b>54</b>
Salesforce1 Context Considerations.....	55
Salesforce1 Access Considerations.....	55
<b>REFERENCE.....</b>	<b>57</b>
<b>Chapter 9: Objects.....</b>	<b>57</b>
CanvasRequest.....	57
Client.....	59
Context.....	60
SignedRequest.....	68
<b>Chapter 10: Force.com Canvas Limits.....</b>	<b>70</b>
<b>Index.....</b>	<b>71</b>

# GETTING STARTED

## Chapter 1

### Introducing Force.com Canvas

---

#### In this chapter ...

- [Force.com Canvas Scenarios](#)
- [Where Canvas Apps Appear](#)
- [Supported Browsers](#)
- [Supported Salesforce Editions](#)
- [User Permissions Required](#)
- [User Interface Considerations](#)
- [Canvas App Process](#)

Force.com Canvas enables you to easily integrate a third-party application in Salesforce. Force.com Canvas is a set of tools and JavaScript APIs that you can use to expose an application as a canvas app. This means you can take your new or existing applications and make them available to your users as part of their Salesforce experience.

Instead of redesigning and reintegrating your external applications, you can now use these tools to integrate your technology within Force.com Canvas. Force.com Canvas includes tools that handle:

- [Authentication](#)—If your application requires authorization, you can implement it by using a signed request or OAuth 2.0.
- [Context](#)—API support that enables you to retrieve context information about the environment in which the canvas app is running.
- [Cross-domain XHR](#)—JavaScript support for cross-domain XML HTTP requests back to the Salesforce domain.
- [Resizing](#)—Methods that support the ability to resize your canvas app.
- [Events](#)—Events provide a JavaScript-based way to send and receive events between canvas apps. Use events to enable communication between multiple canvas apps on a single page.
- [Canvas Apps in Visualforce](#)—A Visualforce component that lets you expose your canvas app on a Visualforce page.
- [Canvas Apps in the Publisher—Pilot](#)—Lets you add a canvas app as a custom action and expand the publisher to include a canvas app.
- [Canvas Apps in the Chatter Feed—Pilot](#)—Lets you expose your canvas apps as feed items.
- [Canvas in the Salesforce1 App—Pilot](#)—Makes your canvas apps available in Salesforce1.

The third-party app that you want to expose as a canvas app can be written in any language. The only requirement is that the app has a secure URL (HTTPS).

#### See Also:

[Force.com Canvas Scenarios](#)  
[Canvas App Process](#)  
[Quick Start](#)

## Force.com Canvas Scenarios

From a high-level view, there are two common scenarios where Force.com Canvas is implemented.

- Application integration—You're a partner, systems integrator, or customer that builds cloud apps, and you'd like to integrate these applications with Salesforce.
- Application rationalization/enterprise desktop—You're a large organization that has many existing apps that your users access in addition to Salesforce. You'd like to integrate these apps into Salesforce so that users can accomplish all of their tasks in one place.

### See Also:

[Introducing Force.com Canvas](#)

[Where Canvas Apps Appear](#)

## Where Canvas Apps Appear

Canvas apps can appear in a few places:



**Note:** Support for Force.com Canvas apps in the publisher, the Chatter feed, and Salesforce1 is currently available through a pilot program and is available in all new Development Edition organizations. For information on enabling it for your organization, contact [salesforce.com](mailto:salesforce.com).

- After you create a canvas app, it appears in the Canvas App Previewer in the organization in which it was created.
- In the Chatter feed, if you code the canvas app to appear there and the current user has access to the canvas app.
- On the Chatter tab, in the Chatter apps list, for any user who has been allowed access to it.
- In an Open CTI call control tool after you add it to the call center's definition file.
- In the publisher, if you configure it to appear as a publisher action.
- In a Salesforce Console after you add it as a custom console component.
- In a Visualforce page after you add it to a page and make that page available to users.
- In a Profile page in the Chatter apps list, for any user who has been allowed access to it.

Where an installed canvas app appears depends on the values you select in the `Locations` field when creating the connected app in Salesforce.

- **Chatter Feed**—The canvas app appears in the Chatter feed. If this option is selected, you must create a CanvasPost feed item and ensure that the current user has access to the canvas app.
- **Chatter Tab**—The canvas app appears in the app navigation list on the Chatter tab. If this option is selected, the canvas app appears there automatically.
- **Open CTI**—The canvas app appears in the call control tool. If this option is selected, you must specify the canvas app in your call center's definition file for it to appear.
- **Publisher**—The canvas app appears in the publisher. If this option is selected, you must also create a canvas custom action and add it to the global layout or to an object layout.
- **Salesforce Console**—The canvas app appears in the footer or sidebars of a Salesforce console. If this option is selected, you must choose where the canvas app appears in a console by adding it as a custom console component.

- **Visualforce Page**—The canvas app can appear on a Visualforce page. If you add an `<apex:canvasApp>` component to expose a canvas app on a Visualforce page, be sure to select this location for the canvas app; otherwise, you'll receive an error.

### See Also:

[Introducing Force.com Canvas](#)

[Set the App Location](#)

[Supported Browsers](#)

## Supported Browsers

Force.com Canvas supports the following browsers:

- Mozilla® Firefox® (preferred)
- Google Chrome™
- Microsoft® Internet Explorer® version 8, 9, and 10 (be sure Compatibility Mode is disabled)
- Apple® Safari® (be sure to set the Block Cookies setting to Never)

If your app uses session cookies, you might need to set your P3P header to allow for third-party cookies or change the browser settings to allow all session cookies.

### See Also:

[Introducing Force.com Canvas](#)

[Supported Salesforce Editions](#)

## Supported Salesforce Editions

Force.com Canvas supports these Salesforce Editions:

Edition	Create a canvas app	Publish a canvas app	Install a canvas app
Performance Edition	Yes	No	Yes
Unlimited Edition	Yes	No	Yes
Enterprise Edition	Yes	No	Yes
Professional Edition	No	No	Yes*
Professional Edition with Force.com Canvas enabled	Yes	No	Yes
Developer Edition	Yes	Yes	Yes

\*Professional Edition organizations must have Force.com Canvas enabled in order for a canvas app to appear in the specified location.

### See Also:

[Introducing Force.com Canvas](#)

[User Permissions Required](#)

## User Permissions Required

The following user permissions are required to create canvas apps and view them in the Canvas App Previewer:

- Customize Application
- Modify All Data

### See Also:

[Introducing Force.com Canvas](#)

[Configure Who Can Access the Canvas App](#)

[Configure Who Can Access the Installed Canvas App](#)

## User Interface Considerations

### Canvas Size

The frame size for canvas apps varies depending on the location where the app appears. When using the SDK, these values are returned in the [Dimensions](#) object.



**Note:** If you plan to use your canvas app in Salesforce1, take into account mobile device screen sizes. For more information, see [Canvas in the Salesforce1 App—Pilot](#).

Location	Description
Chatter tab	The default dimensions are 800 pixels (wide) by 900 pixels (high). The maximum dimensions are 1,000 pixels (wide) by 2,000 pixels (high).
Chatter feed	The default dimensions are 420 pixels (wide) by 100 pixels (high). The maximum dimensions are 420 pixels (wide) by 400 pixels (high).
Open CTI	The default and maximum dimensions are determined by the way you set up the custom console component.
Publisher	The way you set up the canvas publisher action determines the default height. The default width is 522 pixels. The maximum dimensions are 522 pixels (wide) by 500 pixels (high).
Salesforce Console	The default and maximum dimensions are determined by the way you set up the custom console component.
Visualforce page	The default dimensions are 800 pixels (wide) by 900 pixels (high). A developer can change these dimensions by modifying the attributes on the <a href="#">apex:canvasApp Component</a> .

### Logo Image

The logo image associated with a canvas app is displayed when someone installs your canvas app or during OAuth authentication when the user is prompted to allow the app to run. We recommend that you use an image of size 256 pixels (high) by 256 pixels (wide).



## Icon Image

The icon image associated with a canvas app is displayed in these locations:

- To the left of the link to your canvas app on the Chatter tab, in the Chatter apps list.
- To the left of the link to your canvas app in the Canvas App Previewer.

We recommend that you use an image of size 16 pixels (wide) by 16 pixels (high).

## Thumbnail Image

The thumbnail image associated with a canvas app feed item is displayed when someone accesses your canvas app in the feed. If specified, this image appears next to the feed item title and description.

We recommend that you use an image of size 120 pixels (wide) by 12 pixels (high) or smaller.

### See Also:

[Introducing Force.com Canvas](#)

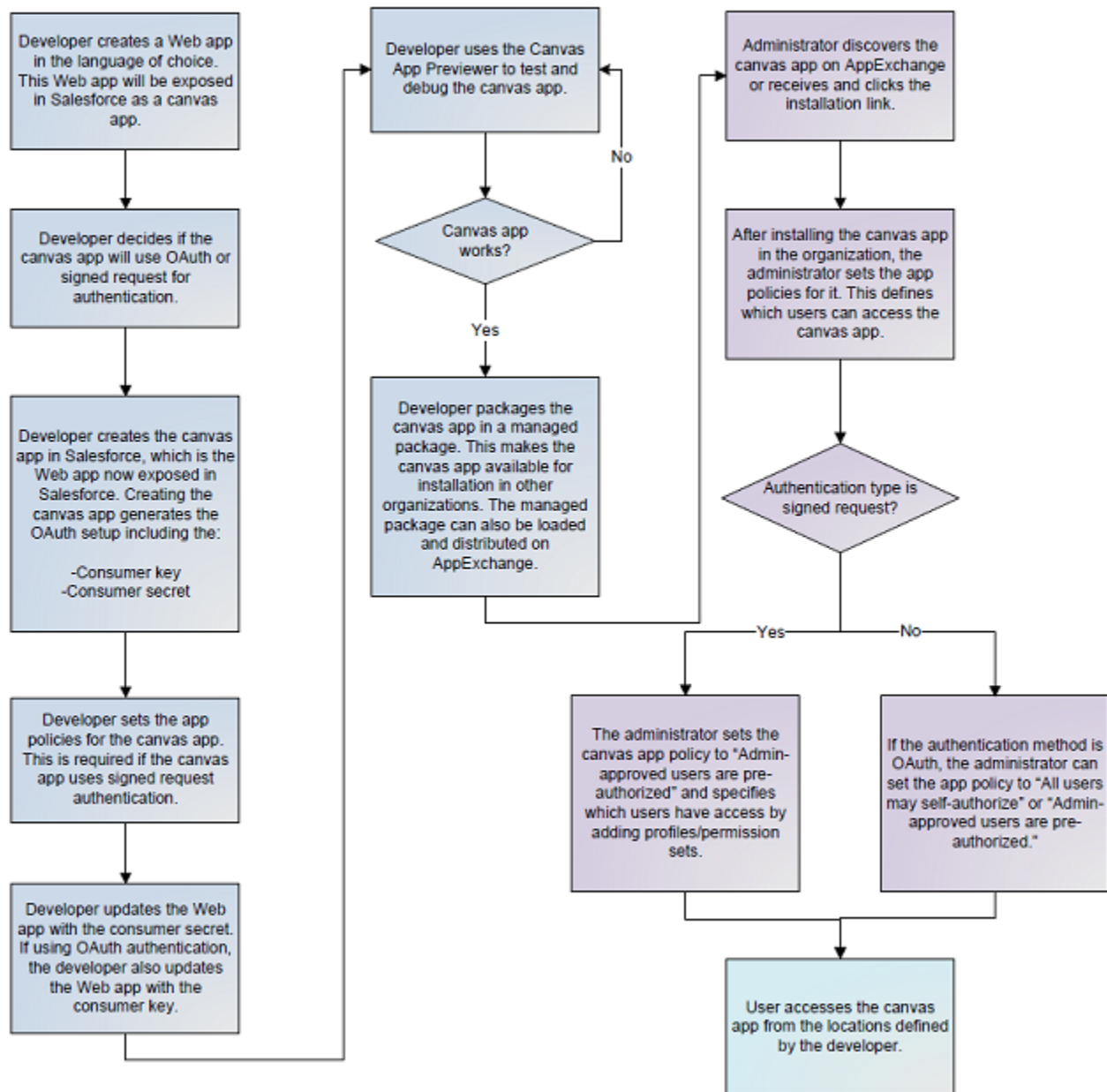
[Resizing a Canvas App](#)

[Canvas App Process](#)

## Canvas App Process

In the process of creating, publishing, installing, and running a canvas app, there are actions required by the developer, the administrator, and the user, as shown in the following diagram.

## Canvas App Process



### See Also:

[Introducing Force.com Canvas Quick Start](#)  
[Quick Start—Advanced](#)

# Chapter 2

## Quick Start

---

### In this chapter ...

- [Prerequisites](#)
- [Create the App](#)
- [Set the App Location](#)

This simple quick start shows you how to get started with Force.com Canvas by using the Heroku Quick Start. The Heroku Quick Start creates a “hello world” app on Heroku in either Java or Ruby, depending on the template you select. At the same time, it creates a corresponding canvas app in Salesforce.

The Heroku app is a “hello world” Web page that calls the Force.com Canvas SDK to display information about the current user and lets you post to the current user’s Chatter feed.

### See Also:

[Prerequisites](#)

[Create the App](#)

[Set the App Location](#)

## Prerequisites

You need the appropriate access and tools to complete the quick start steps.

- Access to a Developer Edition organization.

If you are not already a member of the Force.com developer community, go to <http://developer.force.com/join> and follow the instructions for signing up for a Developer Edition organization. Even if you already have Enterprise Edition, Unlimited Edition, or Performance Edition, use Developer Edition for developing, staging, and testing your solutions against sample data to protect your organization's live data. This is especially true for applications that insert, update, or delete data (as opposed to simply reading data).

If you have an existing Developer Edition organization, and, from Setup, you don't see the menu item **Canvas App Previewer**, contact salesforce.com.

- “Customize Application” and “Modify All Data” user permissions. If you're an administrator, you most likely already have these permissions. Otherwise, you need to add them so that you can see the Canvas App Previewer and create canvas apps.
- A Heroku account. Go here to create a Heroku account: <https://api.heroku.com/signup>.

### See Also:

[Quick Start](#)

[Supported Salesforce Editions](#)

[Create the App](#)

## Create the App

In this step, you'll create both the Heroku “hello world” app and the associated canvas app in your Salesforce organization.

1. In Salesforce, from Setup, click **Canvas App Previewer**.
2. Click **Heroku Quick Start**.
3. In the **Template** field, select **Java – Quick Start Template**.
4. In the **Canvas App Name** field, enter a unique name of up to 30 characters.
5. In the **Heroku App Name** field, enter a unique name of up to 30 characters that begins with a letter and contains only lowercase letters, numbers, and dashes. The `newappName` must be unique across all Heroku apps. This name becomes part of the URL for your app, for example, `newappName.herokuapp.com`.
6. In the **Heroku Username** field, enter the username for the account used to log in to Heroku. This is typically an email address. The Heroku app is created under this user's credentials.



**Note:** This field has a maximum length of 30 characters. If your Heroku username is longer than 30 characters, you'll need to use the API key associated with your account. You can find this value on the Heroku **My Account** page.

7. In the **Heroku Password** field, enter the password for the account used to log in to Heroku.



**Tip:** Instead of using the username and password for the Heroku account, you can use the account's associated API key. You can find this value on the Heroku **Account** page.

8. Click **Create**. The app displays in the left navigation pane.

If you see an error like “Error [Read timed out] executing POST to Heroku clone REST service,” this means the operation has timed out trying to contact Heroku. You can check the status of Heroku at <http://status.heroku.com>.

- Click the link to your new app on the left.

The app appears and you'll see the message `Hello User.FullName`, as well as other information about the current user.

You just created a canvas app—congratulations! You'll only be able to see your canvas app in the Canvas App Previewer until you set the locations where it can appear by following the steps in [Set the App Location](#). This defines where a user sees your app after it's installed in their organization.

Behind the scenes, the Heroku Quick Start sets the canvas app's `Permitted Users`, which includes admin-approved users and your profile. For example, if your user profile is System Administrator, that profile is added to the canvas app you just created, and any users with that profile can access the canvas app.

### See Also:

[Quick Start](#)

[Prerequisites](#)

[Force.com Canvas Limits](#)

[Set the App Location](#)

## Set the App Location

In this step, you'll specify where your canvas app can display to a user in Salesforce.



**Note:** Support for Force.com Canvas apps in the publisher, the Chatter feed, and Salesforce1 is currently available through a pilot program and is available in all new Development Edition organizations. For information on enabling it for your organization, contact [salesforce.com](mailto:salesforce.com).

- In Salesforce, from Setup, click **Create > Apps**.
- In the Connected Apps related list, click the app you just created and then click **Edit**.
- In the Canvas Apps Settings section, in the `Locations` field, select where the canvas app can appear to the user. For this walkthrough, select **Chatter Tab**.
  - Chatter Feed**—The canvas app appears in the Chatter feed. If this option is selected, you must create a CanvasPost feed item and ensure that the current user has access to the canvas app.
  - Chatter Tab**—The canvas app appears in the app navigation list on the Chatter tab. If this option is selected, the canvas app appears there automatically.
  - Open CTI**—The canvas app appears in the call control tool. If this option is selected, you must specify the canvas app in your call center's definition file for it to appear.
  - Publisher**—The canvas app appears in the publisher. If this option is selected, you must also create a canvas custom action and add it to the global layout or to an object layout.
  - Salesforce Console**—The canvas app appears in the footer or sidebars of a Salesforce console. If this option is selected, you must choose where the canvas app appears in a console by adding it as a custom console component.
  - Visualforce Page**—The canvas app can appear on a Visualforce page. If you add an `<apex:canvasApp>` component to expose a canvas app on a Visualforce page, be sure to select this location for the canvas app; otherwise, you'll receive an error.
- Click **Save**.

Because you selected **Chatter Tab**, your canvas app now appears in the left navigation pane on the Chatter tab.

**See Also:**[Quick Start](#)[Create the App](#)[Where Canvas Apps Appear](#)[Quick Start—Advanced](#)

# Chapter 3

## Quick Start—Advanced

---

### In this chapter ...

- [Prerequisites](#)
- [Clone the Project from GitHub](#)
- [Run the Web App Locally](#)
- [Create the Canvas App](#)
- [Configure Who Can Access the Canvas App](#)
- [Deploy the Web App to Heroku](#)
- [Update the Canvas App](#)
- [Package the Canvas App](#)
- [Upload the Canvas App Package](#)
- [Install the Canvas App](#)
- [Configure Who Can Access the Installed Canvas App](#)

This advanced quick start shows you how to get started with more of the Force.com Canvas features. It takes you step-by-step through the process of creating, packaging, uploading, installing, and running a canvas app. The sample canvas app is a “hello world” Web page that calls the Force.com Canvas SDK to display the current user’s name.

In this example, you’ll:

- Clone the “hello world” app from GitHub
- Run the app on a local Web server
- Expose the Web app as a canvas app in your Salesforce development organization and test it in the Canvas App Previewer
- Deploy the Web app to Heroku
- Package and upload the canvas app
- Install the canvas app in another Salesforce organization and run it as a user would

The steps in this quick start assume you’re using Windows. You can use another OS, but there might be some minor differences in the steps.

### See Also:

[Introducing Force.com Canvas Prerequisites](#)

## Prerequisites

You need the appropriate access and tools to complete the quick start steps.

- Access to a Developer Edition organization for developing your canvas app. To install your canvas app, you'll need a Developer Edition organization other than the one you use to create the canvas app.

If you are not already a member of the Force.com developer community, go to <http://developer.force.com/join> and follow the instructions for signing up for a Developer Edition organization. Even if you already have Enterprise Edition, Unlimited Edition, or Performance Edition, use Developer Edition for developing, staging, and testing your solutions against sample data to protect your organization's live data. This is especially true for applications that insert, update, or delete data (as opposed to simply reading data).

If you have an existing Developer Edition organization, and, from Setup, you don't see the menu item **Canvas App Previewer**, contact salesforce.com.

- “Customize Application” and “Modify All Data” user permissions. If you're an administrator, you most likely already have these permissions. Otherwise, you need to add them so that you can see the Canvas App Previewer and create canvas apps.
- Git installed. Go here to install and configure Git: <https://help.github.com/articles/set-up-git>.
- A GitHub account to clone the code example. Go here to set up a GitHub account: <https://github.com/plans>.
- Maven 3.0 or greater installed to package the sample app. Go here to download and install Maven: <http://maven.apache.org/download.html>.
- A Heroku account if you want to run the app from Heroku. Go here to create a Heroku account: <https://api.heroku.com/signup>.
- Heroku Toolbelt if you want to manage the Heroku app from the command line. Go here to download and install Heroku Toolbelt: <https://toolbelt.heroku.com>.

### See Also:

[Quick Start—Advanced](#)

[Supported Salesforce Editions](#)

[Clone the Project from GitHub](#)

## Clone the Project from GitHub

The “hello world” sample project is part of the Force.com Canvas SDK which is located on GitHub. In this step, we'll clone the project to make a copy of it on your local machine.

1. Log into <https://github.com>.
2. Locate the project that contains the Force.com Canvas SDK and code examples by going to: <https://github.com/forcedotcom/SalesforceCanvasFrameworkSDK>.
3. Clone the project by clicking Zip and downloading the .zip file.
4. Extract all the files from the zip file locally into a directory called `c:\SalesforceCanvasFrameworkSDK`. Alternatively, you can download the project by using the Git command line and issuing this command: `git clone git@github.com:forcedotcom/SalesforceCanvasFrameworkSDK.git`.
5. Open a command window and navigate to `c:\SalesforceCanvasFrameworkSDK`.
6. Enter the command `git submodule init`.
7. Enter the command `git submodule update`. This adds other projects into the current project. The file `c:\SalesforceCanvasFrameworkSDK\.gitmodules` shows you which projects are included as submodules.



After you clone the project, the `c:\SalesforceCanvasFrameworkSDK` directory should contain a subdirectory named `src`, as well as files like `pom.xml` and `README.md`.

### See Also:

[Quick Start—Advanced](#)

[Prerequisites](#)

[Run the Web App Locally](#)

## Run the Web App Locally

In this step, you'll package the Web app using Maven and then run it locally using Jetty. When you package the Web app, the process downloads all the components needed to run the Web app, including Jetty.

1. Open a command window and navigate to `c:\SalesforceCanvasFrameworkSDK`.
2. Enter the command `mvn package`. You'll see output in the command window as Maven packages the app and its dependent components. If the process completes successfully, you'll see something like this:

```
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 28.016s
[INFO] Finished at: Tue Jul 03 08:00:42 PDT 2012
[INFO] Final Memory: 8M/59M
[INFO] -----
```

3. To use Jetty to run the app, you'll need to enable local SSL support. This step only needs to be done once per app, so if you've already done this, skip this step. Ensure that the command window is open and you're in the directory `c:\SalesforceCanvasFrameworkSDK`.
4. Run the following command: `keytool -keystore keystore -alias jetty -genkey -keyalg RSA`.

After you run this command, you'll be prompted for the following information. Enter 123456 for the keystore password and yes to confirm at the end. When prompted, "Enter key password for <jetty>," press Enter to use the keystore password. For the other information, you can enter values or leave them blank.

```
Enter keystore password: <Choose Your Password>
Re-enter new password: <Choose Your Password>
What is your first and last name?
[Unknown]: <Enter First and Last Name>
What is the name of your organizational unit?
[Unknown]: <Enter an Org Unit>
What is the name of your organization?
[Unknown]: <Enter an Org>
What is the name of your City or Locality?
[Unknown]: <Enter a City>
What is the name of your State or Province?
[Unknown]: <Enter a State>
What is the two-letter country code for this unit?
[Unknown]: <Enter a Country>
Is CN=XXXX, OU=XXXX, O=XXXX, L=XXXX, ST=XX, C=XX correct?
[no]: yes

Enter key password for <jetty>
(RETURN if same as keystore password):
```

This creates a file named `keystore` in the directory `c:\SalesforceCanvasFrameworkSDK`. The keystore is used by Jetty for SSL support.

5. Run the Web server by entering this command: `target\bin\webapp.bat` (Windows) or `sh target/bin/webapp` (Unix/OS X).

If you're using Unix/OS X, you may need to add execute permissions to `webapp` before you can run it. Use this command to do so: `chmod +x target/bin/webapp`.

6. Verify that the app is running by opening a browser and navigating to the following URL:

`https://localhost:8443/examples/hello-world/index.jsp`.

Depending on your browser and security settings, you might need to add a security exception because you're running a site with an unsigned SSL certificate.

You should see a message that says, "This App must be invoked via a signed request!" This is an indication that the Web app is running locally. This message appears because the app is designed to receive a signed request from Salesforce, therefore, the app won't run outside of the Salesforce canvas environment.

### See Also:

[Quick Start—Advanced](#)

[Clone the Project from GitHub](#)

[Create the Canvas App](#)

## Create the Canvas App

In this step, you'll create the canvas app in your Salesforce organization. You'll need user permissions "Customize Application" and "Modify All Data" to create a canvas app.

1. In Salesforce, from Setup, click **Create > Apps**.
2. In the Connected Apps related list, click **New**.
3. In the `Connected App Name` field, enter `Hello World`.
4. Accept the default `API Name` of `Hello_World`. This is the internal name of the canvas app and can't be changed after you save it.
5. In the `Contact Email` field, enter your email address.
6. In the `Logo Image URL` field, enter `https://localhost:8443/images/salesforce.png`. This is the default salesforce.com "No Software" image. This image appears on the installation screen and on the detail screen for the app.
7. In the `Icon URL` field, enter `https://localhost:8443/examples/hello-world/logo.png`. This is the default salesforce.com "No Software" image.

This is the image that appears next to the app name in the user interface. If you leave this field blank, a default cloud image appears next to the app name.

8. In the `API (Enable OAuth Settings)` section, select the `Enable OAuth Settings` field.
9. In the `Callback URL` field, enter `https://localhost:8443/sdk/callback.html`.
10. In the `Selected OAuth Scopes` field, select `Access your basic information`.
11. In the `Canvas App Settings` section, select `Force.com Canvas`.
12. In the `Canvas App URL` field, enter `https://localhost:8443/examples/hello-world/index.jsp`.
13. In the `Access Method` field, select `Signed Request (Post)`.
14. In the `Locations` field, select `Chatter Tab`.
15. Click **Save**. After the canvas app is saved, the detail page appears.
16. On the detail page for the canvas app, next to the `Consumer Secret` field, click the link `Click to reveal`. The consumer secret is used in the app to authenticate.
17. Select the consumer secret value and enter `CTRL+C` to copy it.

18. Go to the command window and stop the Jetty Web server by entering CTRL+C. At the prompt, enter Y.
19. Create an environment variable named `CANVAS_CONSUMER_SECRET` and set that value to the consumer secret you just copied. To do this in Windows, in the command window, enter `set CANVAS_CONSUMER_SECRET=value_you_just_copied`.

If you're using Unix/OS X, set the environment variable with the command `export CANVAS_CONSUMER_SECRET=value_you_just_copied`.

The "hello world" page

(`c:\SalesforceCanvasFrameworkSDK\src\main\webapp\examples\hello-world\index.jsp`) uses the consumer secret, as shown in the following code:

```
<%@ page import="canvas.SignedRequest" %>
<%@ page import="java.util.Map" %>
<%
    // Pull the signed request out of the request body and verify and decode it.
    Map<String, String[]> parameters = request.getParameterMap();
    String[] signedRequest = parameters.get("signed_request");
    if (signedRequest == null) {
        This app must be invoked via a signed request!
        return;
    }
    String yourConsumerSecret=System.getenv("CANVAS_CONSUMER_SECRET");
    String signedRequestJson = SignedRequest.verifyAndDecodeAsJson(signedRequest[0],
yourConsumerSecret);
%>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en">
<head>

    <title>Hello World Canvas Example</title>

    <link rel="stylesheet" type="text/css" href="/sdk/css/connect.css" />

    <script type="text/javascript" src="/sdk/js/canvas-all.js"></script>

    <!-- Third part libraries, substitute with your own -->
    <script type="text/javascript" src="/scripts/json2.js"></script>

    <script>
        if (self === top) {
            // Not in an iFrame.
            alert("This canvas app must be included within an iFrame");
        }

        Sfdc.canvas(function() {
            var sr = JSON.parse('<%=signedRequestJson%>');
            Sfdc.canvas.byId('username').innerHTML = sr.context.user.fullName;
        });

    </script>
</head>
<body>
    <br/>
    <h1>Hello <span id='username'></span></h1>
</body>
</html>
```

20. Restart the Web server by entering this command: `target\bin\webapp.bat` (Windows) or `sh target/bin/webapp` (Unix/OS X).

### See Also:

[Quick Start—Advanced](#)  
[Run the Web App Locally](#)  
[Where Canvas Apps Appear](#)  
[User Permissions Required](#)  
[Force.com Canvas Limits](#)  
[Configure Who Can Access the Canvas App](#)

## Configure Who Can Access the Canvas App

You’ve created the canvas app in Salesforce, but you won’t be able to see the app in your development organization until you configure user access.

1. In Salesforce, from Setup, click **Manage Apps > Connected Apps**.
2. Click the Hello World app, and then **Edit**.
3. In the **Permitted Users** field, select **Admin approved users are pre-authorized**. Click **OK** on the pop-up message that appears.
4. Click **Save**.

This is where you define who can see your canvas app. This can be done using profiles and permission sets. In this example, we assume that your profile is **System Administrator**.

5. In the Profiles related list, click **Manage Profiles**.
6. Select the **System Administrator** profile and click **Save**.
7. In Salesforce, from Setup, click **Canvas App Previewer**. You can use the Canvas App Previewer to test out your canvas app before publishing it.
8. Click the Hello World link on the left.

The app appears and you’ll see the message **Hello *User.FullName***. The canvas app works in this context because when you click the app name in the previewer, the signed request is sent to the endpoint `https://localhost:8443/examples/hello-world/index.jsp`.

After configuring access, you can see the canvas app in the Canvas App Previewer and on the Chatter tab in your development organization.

### See Also:

[Quick Start—Advanced](#)  
[Create the Canvas App](#)  
[Deploy the Web App to Heroku](#)

## Deploy the Web App to Heroku

In this walkthrough, you previously ran the “hello world” Web app locally before adding it as a canvas app and testing it. Now that your canvas app works locally, you’re ready to deploy your “hello world” Web app to Heroku and run it from there. Here’s how you do it.

1. If you haven’t already, log into Heroku and install Heroku Toolbelt following the links in the [Prerequisites](#).

2. Open a command window, navigate to `c:\SalesforceCanvasFrameworkSDK`, and enter the following command:  
`git init`. This re-initializes the directory as a Git repository.
3. In the command window, enter the following command: `heroku create`. This creates a new “shell” app on Heroku.  
Confirmation that the app was created looks like this:

```
Creating deep-samurai-7923... done, stack is cedar
http://deep-samurai-7923.herokuapp.com/ | git@heroku.com:deep-samurai-7923.git
Git remote heroku added
```

4. To rename this Heroku app, enter the following command in the command window: `heroku rename newAppName --app oldAppName`.

In this example, `oldAppName` is `deep-samurai-7923`. The `newAppName` you create must begin with a letter and can only contain lowercase letters, numbers, and dashes. You'll see a confirmation of the renaming that looks similar to this:

```
http://newappName.herokuapp.com/ | git@heroku.com:newappName.git
```

The `newappName` must be unique across all Heroku apps. This name becomes part of the URL for your app, for example, `newappName.herokuapp.com`.

5. Run the following command in the command window: `git add -A`. This adds the entire `SalesforceCanvasFrameworkSDK` project to the Git repository. If you're working in the Windows environment, you might see some messages about LF (line feeds) being replaced by CRLF (carriage return line feeds).
6. Enter the following command in the command window to commit the changes along with a comment: `git commit -m "MyChangeComments"`.
7. Enter the following command in the command window to deploy the changes to Heroku: `git push heroku master`.  
If the process completes successfully, you'll see something like this:

```
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 13.188s
[INFO] Finished at: Sat Feb 09 21:14:23 UTC 2013
[INFO] Final Memory: 11M/490M
[INFO] -----
```

If you receive a “permission denied” error message, you may need to set up your SSH key and add it to Heroku. See <https://devcenter.heroku.com/articles/keys>.

8. Open a command window, and set the Heroku environment variable that contains the consumer secret by entering this command and replacing `consumer_secret_value` with the value you just copied: `heroku config:add CANVAS_CONSUMER_SECRET=consumer_secret_value`.  
To get the consumer secret for the canvas app, from Setup, navigate to **Create > Apps** and click the Hello World app. You'll see the Consumer Secret field in the OAuth Settings section.
9. Verify that the app is running in Heroku by opening a browser and navigating to the following URL:  
`https://newappName.herokuapp.com/examples/hello-world/index.jsp`.

You should see a message that says, “This App must be invoked via a signed request!” This is an indication that the app is running in Heroku. This message appears because the app is designed to receive a signed request from Salesforce, therefore, the app won’t run outside of the Salesforce canvas environment.

### See Also:

[Quick Start—Advanced](#)

[Configure Who Can Access the Canvas App](#)

[Update the Canvas App](#)

## Update the Canvas App

In this step, you’ll update the canvas app to run the “hello world” app that’s now running on Heroku.

1. In Salesforce, from Setup, click **Create > Apps**.
2. In the Connected Apps related list, click **Hello World**.
3. Click **Edit**.
4. In the Logo Image URL field, enter `https://appName.herokuapp.com/images/salesforce.png`. This is the default salesforce.com “No Software” image. This image appears on the installation screen and on the detail screen for the app. The `appName` is the name of Heroku app that you just created.
5. In the Icon URL field, enter `https://appName.herokuapp.com/examples/hello-world/logo.png`. This is the default salesforce.com “No Software” image.

This is the image that appears next to the app name on the Chatter tab or in the Canvas App Previewer. If you leave this field blank, a default cloud image appears next to the app name. The `appName` is the name of Heroku app that you just created.

6. In the Callback URL field, enter `https://appName.herokuapp.com/sdk/callback.html`. The `appName` is the name of Heroku app that you just created.
7. In the Canvas App URL field, enter `https://appName.herokuapp.com/examples/hello-world/index.jsp`. The `appName` is the name of Heroku app that you just created.
8. Click **Save**. After the canvas app is saved, the detail page appears.
9. In Salesforce, from Setup, click **Canvas App Previewer**. You can use the Canvas App Previewer to test out your canvas app before repackaging it.
10. Click the Hello World link on the left.

The app should appear and you’ll see the message **Hello *User.FullName***. The canvas app works in this context because when you click the app name in the previewer, the signed request is sent to the endpoint `https://appName.herokuapp.com/examples/hello-world/index.jsp`.

In this example, we’re using the same canvas app that we just created, but updating it to point to the “hello world” Web app running on Heroku. Therefore, the consumer secret that we previously added to our “hello world” app doesn’t need to be updated.

If you want to create a new canvas app in Salesforce that displays the “hello world” app running on Heroku, then go to [Create the Canvas App](#) to create the new app, update the consumer secret in the app, and then deploy the changes to Heroku.

### See Also:

[Quick Start—Advanced](#)

[Deploy the Web App to Heroku](#)

[Package the Canvas App](#)

## Package the Canvas App

Now that you've created the canvas app, tested it in the Canvas App Previewer, and deployed it to Heroku, you're ready to package and distribute it. Packaging is the first step in making your canvas app available for installation in another organization. For more information about packaging, see the [ISVforce Guide](#).

1. In Salesforce, from Setup, click **Create > Packages** and click **New**.



**Tip:** To package your canvas app for installation in other organizations, you must create a namespace prefix. A namespace prefix can only be defined in a Developer Edition organization. For more information, see the topic “Registering a Namespace Prefix” in the online help.

2. Set the `Package Name` field to Hello World Package and accept the defaults for the other fields.
3. Select the `Managed` field to make the package the managed package for the organization.
4. A prompt appears stating you're only allowed one managed package. Click **OK**.
5. Click **Save**.
6. The package is empty, so click **Add**.
7. In the `Component Type` field, select Connected App.
8. Select the checkbox next to the Hello World app and click **Add To Package**. The Package Detail screen appears. From Setup, click **Create > Packages** to see the new managed package.

Now you're ready to upload the package you created and get the installation link. Use the installation link to install your canvas app in another organization.

### See Also:

[Quick Start—Advanced](#)

[Update the Canvas App](#)

[Understanding Managed and Unmanaged Packages](#)

[Upload the Canvas App Package](#)

## Upload the Canvas App Package

Now that you've packaged the canvas app, you're ready to upload the package. This creates an installation link that you can provide to anyone who needs to install your canvas app in their organization.

1. In Salesforce, from Setup, click **Create > Packages**.
2. In the Packages list, click the Hello World Package link.
3. On the Package Detail page, click **Upload** to publish the managed package.
4. In the `Version Name` field, enter Winter 2014 Beta. Keep the default Version Number of 1.0 and Release Type of Managed—Beta.
5. Click **Upload**.



**Note:** A canvas app can only be installed in an organization other than the one you're developing in.

After the upload completes, the Version Detail page appears. The Installation URL field contains the URL that you can use to install the canvas app in another organization. You'll also receive a notification email that contains the installation URL.

### See Also:

[Quick Start—Advanced](#)

[Package the Canvas App](#)

[Publishing Your App](#)

[Install the Canvas App](#)

## Install the Canvas App

Uploading the packaged canvas app creates the installation link. You can then use this link to install the canvas app in another organization. Beta packages can only be installed in sandbox or Developer Edition organizations, otherwise, you'll receive an error.

1. Open a browser and enter the canvas app installation URL in the browser address bar. If you're in the process of developing your canvas app, be sure to log out of your development organization before you install the app.

The installation URL looks something like:

`https://login.instance.salesforce.com/services/packaging/installPackage.apexp?p0=04eU0000000AWNA.`

2. When prompted, enter your login credentials for the organization in which you're installing the package.
3. The Package Installation Details page appears. In the Package Components list, you should see the Hello World canvas app. Click **Continue**.
4. Click **Next** and then click **Install**.

After the package installation completes successfully, you'll receive an email notification.

### See Also:

[Quick Start—Advanced](#)

[Upload the Canvas App Package](#)

[Supported Salesforce Editions](#)

[Configure Who Can Access the Installed Canvas App](#)

## Configure Who Can Access the Installed Canvas App

You've installed the canvas app in your Salesforce organization, but no one can see it until you configure user access.

1. In Salesforce, from Setup, click **Manage Apps > Connected Apps**.
2. Click the Hello World app, and then click **Edit**.
3. In the `Permitted Users` field, select Admin approved users are pre-authorized. Click **OK** on the pop-up message that appears.
4. Click **Save**.

Now you'll define who can see your canvas app. This can be done using profiles and permission sets. In this example, we'll allow anyone with the System Administrator to access the app.

5. On the Connected App Detail page, in the Profiles related list, click **Manage Profiles**.
6. Select the `System Administrator` profile and click **Save**.
7. Click the Chatter tab.



8. Click the Hello World link on the left.

The app appears and you'll see the message `Hello User.FullName`.

**See Also:**

[Quick Start—Advanced](#)

[Install the Canvas App](#)

# USING FORCE.COM CANVAS

## Chapter 4

### Force.com Canvas SDK

---

#### In this chapter ...

- [Referencing the Force.com Canvas SDK](#)
- [Authentication](#)
- [Getting Context in Your Canvas App](#)
- [Cross-Domain XHR](#)
- [Resizing a Canvas App](#)
- [Implementing Canvas App Events](#)
- [Using Streaming API in a Canvas App](#)

Force.com Canvas is a set of tools that enable you to integrate your apps within Salesforce. This framework includes an SDK that you can use to authenticate your app and retrieve data from Salesforce. The Force.com Canvas SDK and code examples are available on GitHub at

<https://github.com/forcedotcom/salesforcecanvasframeworksdk>.

The Force.com Canvas SDK is versioned and matches the API version in each release. The current version is 29.0. You can find out the version of the SDK that you have by calling the `version` method. Previous versions of this developer's guide can be found at

[http://wiki.developerforce.com/page/Earlier\\_Reference\\_Documentation](http://wiki.developerforce.com/page/Earlier_Reference_Documentation).

#### See Also:

[Referencing the Force.com Canvas SDK](#)

[Authentication](#)

[Getting Context in Your Canvas App](#)

[Cross-Domain XHR](#)

[Resizing a Canvas App](#)

[Implementing Canvas App Events](#)

## Referencing the Force.com Canvas SDK

The Force.com Canvas SDK is available on [GitHub](#), and you have two options for referencing it from your canvas app.

- Host the SDK on your own Web server and access it there
- Access the SDK hosted on the Salesforce server

For example, here's what the include statement looks like if you host the SDK on your own Web server:

```
<script type="text/javascript" src="/sdk/js/canvas-all.js"></script>
```

Here's what the include statement looks like if you reference the hosted SDK:

```
<script type="text/javascript"
src="https://<instance>.salesforce.com/canvas/sdk/js/29.0/canvas-all.js"></script>
```

The ability to reference the SDK on the Salesforce server is useful when you want to include one of the SDK files in a Web app or from a Visualforce page.

### See Also:

[Force.com Canvas SDK](#)  
[Authentication](#)

## Authentication

When you create a canvas app, you can use one of the following authentication methods:

- [Signed request](#)—The default method of authentication for canvas apps in which the signed request containing the consumer key, access token, and other contextual information is provided to the canvas app.
- [OAuth 2.0](#)—Canvas apps can use the OAuth 2.0 protocol to authenticate and acquire access tokens. For more information about OAuth and the Force.com platform, see [http://wiki.developerforce.com/page/Digging\\_Deeper\\_into\\_OAuth\\_2.0\\_on\\_Force.com](http://wiki.developerforce.com/page/Digging_Deeper_into_OAuth_2.0_on_Force.com).

### See Also:

[Force.com Canvas SDK](#)  
[Understanding Authentication](#)  
[Canvas App User Flow—Signed Request](#)  
[Canvas App User Flow—OAuth](#)

## Signed Request Authentication

This is the default authorization method for canvas apps. In the first and subsequent requests to the canvas app, Salesforce performs a POST with all the authorization information contained in the body of the signed request. This information can be verified with the client secret and used to customize the app, and make subsequent calls to Salesforce.

When you use signed request to authenticate a canvas app, the app is accessible to users as soon as the administrator installs it in the organization and configures which users can see it. Therefore, the user won't see a popup window to allow the app through OAuth.

The signed request is a string of the following elements concatenated:

- The canvas app consumer secret encrypted with HMAC SHA-256 algorithm
- A period (“.”)
- The context and authorization token JSON encoded in Base64

The signed request looks similar to this, although it will be much longer:

9Rp16rE7R2bSNjoSfYdERk8nffmgtKQNhr5U/5eSJPI=.eyJjb250ZXh0Ijp7InVzZXIiOiOnsibGFuZ3V....

Signed request considerations:

- Salesforce performs an HTTP POST when invoking the canvas app URL.
- The access token and context information are included in the signed request, so there’s no need for multiple requests.
- Server-side code is needed to verify and decode the request.

### See Also:

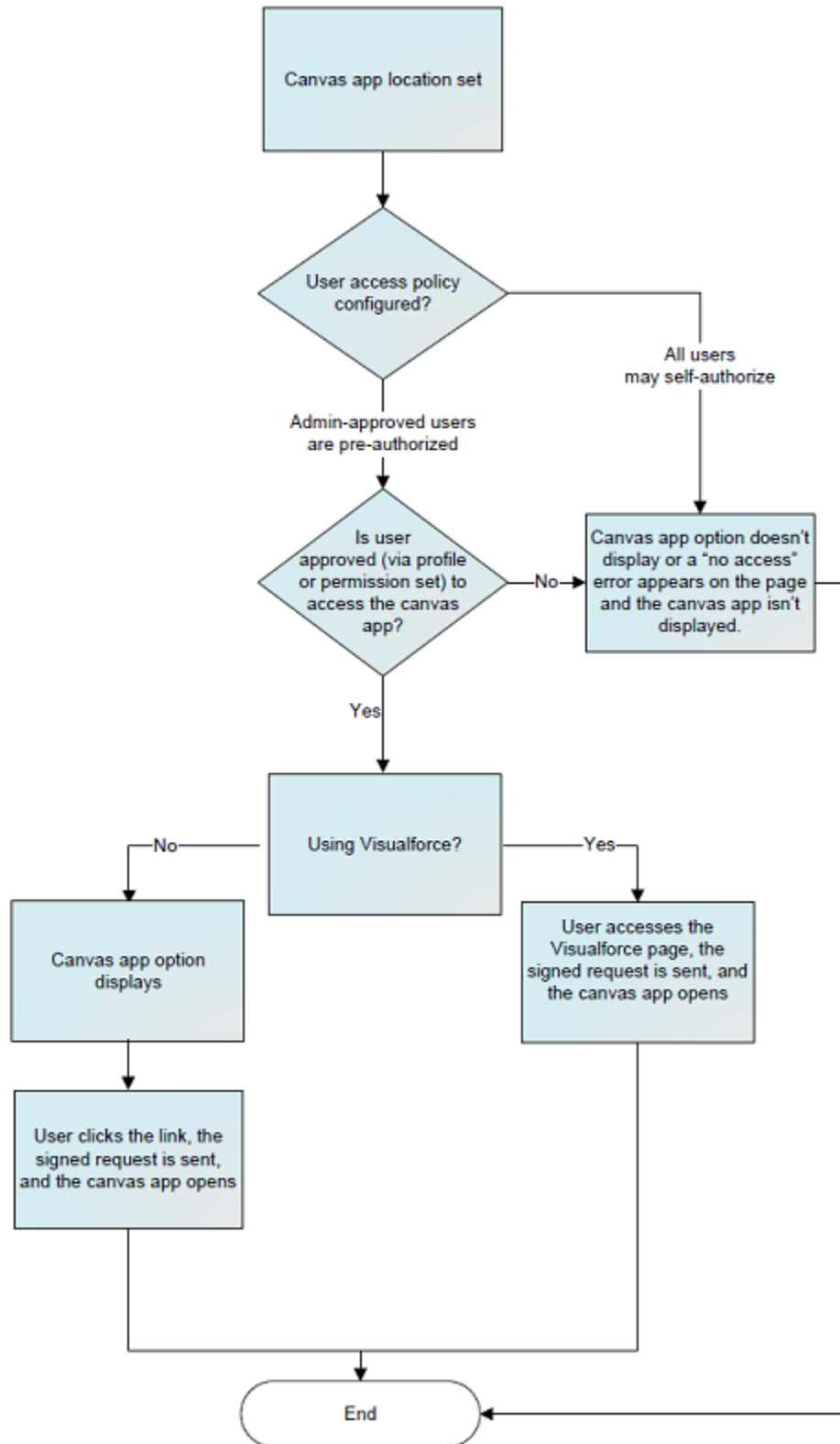
[Authentication](#)

[Verifying and Decoding a Signed Request](#)

[Canvas App User Flow—Signed Request](#)

## Canvas App User Flow—Signed Request

If your canvas app uses signed request authentication, the user experience varies depending on where the canvas app is located in the user interface and how the user access is set. This diagram shows the user flow for a canvas app that uses signed request authentication.



**See Also:**[Signed Request Authentication](#)[SignedRequest](#)[Verifying and Decoding a Signed Request](#)**Verifying and Decoding a Signed Request**

When using a signed request, Salesforce delivers the user context and authentication information to your canvas app URL. To ensure that the signed request is valid, you must verify that the signed request was signed using your specific canvas app consumer secret. If the correct consumer secret was used, then you can trust the context; otherwise, you can assume that the request was not initiated by Salesforce. To verify and decode the signed request, your application should:

1. Receive the POST message that contains the initial signed request from Salesforce.
2. Split the signed request on the first period. The result is two strings: the hashed Base64 context signed with the consumer secret and the Base64 encoded context itself.
3. Use the HMAC SHA-256 algorithm to hash the Base64 encoded context and sign it using your consumer secret.
4. Base64 encode the string created in the previous step.
5. Compare the Base64 encoded string with the hashed Base64 context signed with the consumer secret you received in step 2.

If the two values are the same, then you know that the signed request was signed using your consumer secret and can be trusted. From there, you can Base64 decode the encoded context and parse out any values you need. For more information on those values, see [CanvasRequest](#). If the two strings are different, then the request was not hashed and signed using your consumer secret, and you should return the appropriate message to the user.

**Functions for Verifying and Decoding**

To verify the signed request, you can call the one the following functions found in the Force.com Canvas SDK (in `SalesforceCanvasFrameworkSDK\src\main\java\canvas\SignedRequest.java`):

- `verifyAndDecode`—Returns a verified and decoded version of the signed request as a Java object.
- `verifyAndDecodeAsJson`—Returns a verified and decoded version of the signed request as a JSON-formatted string.

The following code example shows you how to verify and decode a signed request using the functions in the SDK. This code splits the signed request string at the period to parse out the signed secret and the Base64 JSON string. It then encrypts the canvas app consumer secret signed with the HMAC SHA-256 algorithm and compares the encrypted value with the encrypted value sent to you by Salesforce.

If the two values are the same, you know that the context is valid and came from Salesforce. If the two values are different, then the request didn't come from Salesforce.

```
/**
 *
 * The utility method can be used to validate/verify the signed request.
 * In this case, the signed request is verified that it's from Salesforce and that
 * it has not been tampered with.
 *
 * This utility class has two methods. One verifies and decodes the request
 * as a Java object, the other as a JSON String.
 */
public class SignedRequest {
    public static CanvasRequest verifyAndDecode(String input, String secret)
        throws SecurityException {
        String[] split = getParts(input);
        String encodedSig = split[0];
        String encodedEnvelope = split[1];

        // Deserialize the JSON body.
```

```

String json_envelope = new String(new Base64(true).decode(encodedEnvelope));
ObjectMapper mapper = new ObjectMapper();
ObjectReader reader = mapper.reader(CanvasRequest.class);
CanvasRequest canvasRequest;
String algorithm;
try {
    canvasRequest = reader.readValue(json_envelope);
    algorithm = canvasRequest.getAlgorithm() == null ?
        "HMACSHA256" : canvasRequest.getAlgorithm();
} catch (IOException e) {
    throw new SecurityException(String.format("Error [%s] deserializing JSON to
        Object [%s]", e.getMessage(), CanvasRequest.class.getName()), e);
}
verify(secret, algorithm, encodedEnvelope, encodedSig);
// If we got this far, then the request was not tampered with.
// Return the request as a Java object.
return canvasRequest;
}

public static String verifyAndDecodeAsJson(String input, String secret)
    throws SecurityException {
    String[] split = getParts(input);
    String encodedSig = split[0];
    String encodedEnvelope = split[1];
    String json_envelope = new String(new Base64(true).decode(encodedEnvelope));
    ObjectMapper mapper = new ObjectMapper();
    String algorithm;
    StringWriter writer;
    TypeReference<HashMap<String, Object>> typeRef
        = new TypeReference<HashMap<String, Object>>() { };
    try {
        HashMap<String, Object> o = mapper.readValue(json_envelope, typeRef);
        writer = new StringWriter();
        mapper.writeValue(writer, o);
        algorithm = (String)o.get("algorithm");
    } catch (IOException e) {
        throw new SecurityException(String.format("Error [%s] deserializing
            JSON to Object [%s]", e.getMessage(),
                typeRef.getClass().getName()), e);
    }
    verify(secret, algorithm, encodedEnvelope, encodedSig);
    // If we got this far, then the request was not tampered with.
    // Return the request as a JSON string.
    return writer.toString();
}

private static String[] getParts(String input) {
    if (input == null || input.indexOf(".") <= 0) {
        throw new SecurityException(String.format("Input [%s] doesn't
            look like a signed request", input));
    }
    String[] split = input.split("[.]", 2);
    return split;
}

private static void verify(String secret, String algorithm,
    String encodedEnvelope, String encodedSig )
    throws SecurityException
{
    if (secret == null || secret.trim().length() == 0) {
        throw new IllegalArgumentException("secret is null, did you
            set your environment variable CANVAS_CONSUMER_SECRET?");
    }
    SecretKey hmacKey = null;
    try {
        byte[] key = secret.getBytes();
        hmacKey = new SecretKeySpec(key, algorithm);
        Mac mac = Mac.getInstance(algorithm);
        mac.init(hmacKey);
        // Check to see if the body was tampered with.
        byte[] digest = mac.doFinal(encodedEnvelope.getBytes());
    }

```

```

        byte[] decode_sig = new Base64(true).decode(encodedSig);
        if (!Arrays.equals(digest, decode_sig)) {
            String label = "Warning: Request was tampered with";
            throw new SecurityException(label);
        }
    } catch (NoSuchAlgorithmException e) {
        throw new SecurityException(String.format("Problem with algorithm [%s]
            Error [%s]", algorithm, e.getMessage()), e);
    } catch (InvalidKeyException e) {
        throw new SecurityException(String.format("Problem with key [%s]
            Error [%s]", hmacKey, e.getMessage()), e);
    }
    // If we got here and didn't throw a SecurityException then all is good.
}
}

```

### Calling the verifyAndDecode Function

The following code shows an example of getting the signed request, and then verifying and decoding the request by using the `verifyAndDecode` function.

```

// From a JSP or servlet.
<%@ page import="canvas.SignedRequest" %>
<%@ page import="java.util.Map" %>
<%
    // Pull the signed request out of the request body and verify/decode it.
    Map<String, String[]> parameters = request.getParameterMap();
    String[] signedRequest = parameters.get("signed_request");
    if (signedRequest == null) {%>
        This app must be invoked via a signed request!<%
        return;
    }
    String yourConsumerSecret=System.getenv("CANVAS_CONSUMER_SECRET");
    String signedRequest = SignedRequest.verifyAndDecode(signedRequest[0],
yourConsumerSecret);
%>
...
// From JavaScript, you can handle the signed request as needed.
var signedRequest = '<%=signedRequestJson%>';

```

### Calling the verifyAndDecodeAsJson Function

The following code shows an example of getting the signed request, verifying and decoding the request by using the `verifyAndDecodeAsJson` function, and parsing the returned JSON result.

```

// From a JSP or servlet.
<%@ page import="canvas.SignedRequest" %>
<%@ page import="java.util.Map" %>
<%
    // Pull the signed request out of the request body and verify/decode it.
    Map<String, String[]> parameters = request.getParameterMap();
    String[] signedRequest = parameters.get("signed_request");
    if (signedRequest == null) {%>
        This App must be invoked via a signed request!<%
        return;
    }
    String yourConsumerSecret=System.getenv("CANVAS_CONSUMER_SECRET");
    String signedRequestJson = SignedRequest.verifyAndDecodeAsJson(signedRequest[0],
yourConsumerSecret);
%>
...

```



```
// From JavaScript, you can parse with your favorite JSON library.  
var signedRequest = JSON.parse('<%=signedRequestJson%>');
```

### See Also:

[Signed Request Authentication](#)  
[SignedRequest](#)

## OAuth Authentication

Force.com Canvas supports OAuth 2.0 for authorization. When using OAuth, you have two options:

- **Web Server OAuth Authentication Flow**—When users run your canvas app, they can authorize the app to access their data. This requires each user to allow the canvas app to access their information. For more information, see [Understanding the Web Server OAuth Authentication Flow](#) in the [Force.com REST API Developer's Guide](#).
- **User-Agent OAuth Authentication Flow**—When users run your canvas app, they can authorize the app to access their data by using just the browser for authentication. Like Web Server OAuth authentication, this option requires each user to allow the canvas app to access their information.

It's recommended that you use this authorization method during development and not in production because no refresh token is supplied. With this type of authorization, no server-side code is needed and there's no need to expose development machines to the public internet. For more information, see [Understanding the User-Agent OAuth Authentication Flow](#) in the [Force.com REST API Developer's Guide](#).

Regardless of which OAuth flow you implement, the canvas app must provide code for initiating the standards-based OAuth flow. OAuth considerations include:

- Salesforce performs an HTTP GET when invoking the canvas app URL.
- With user agent OAuth, all authorization can be performed in the browser (no server-side code is needed).

For more information about OAuth and the Force.com platform, see

[http://wiki.developerforce.com/page/Digging\\_Deeper\\_into\\_OAuth\\_2.0\\_on\\_Force.com](http://wiki.developerforce.com/page/Digging_Deeper_into_OAuth_2.0_on_Force.com).

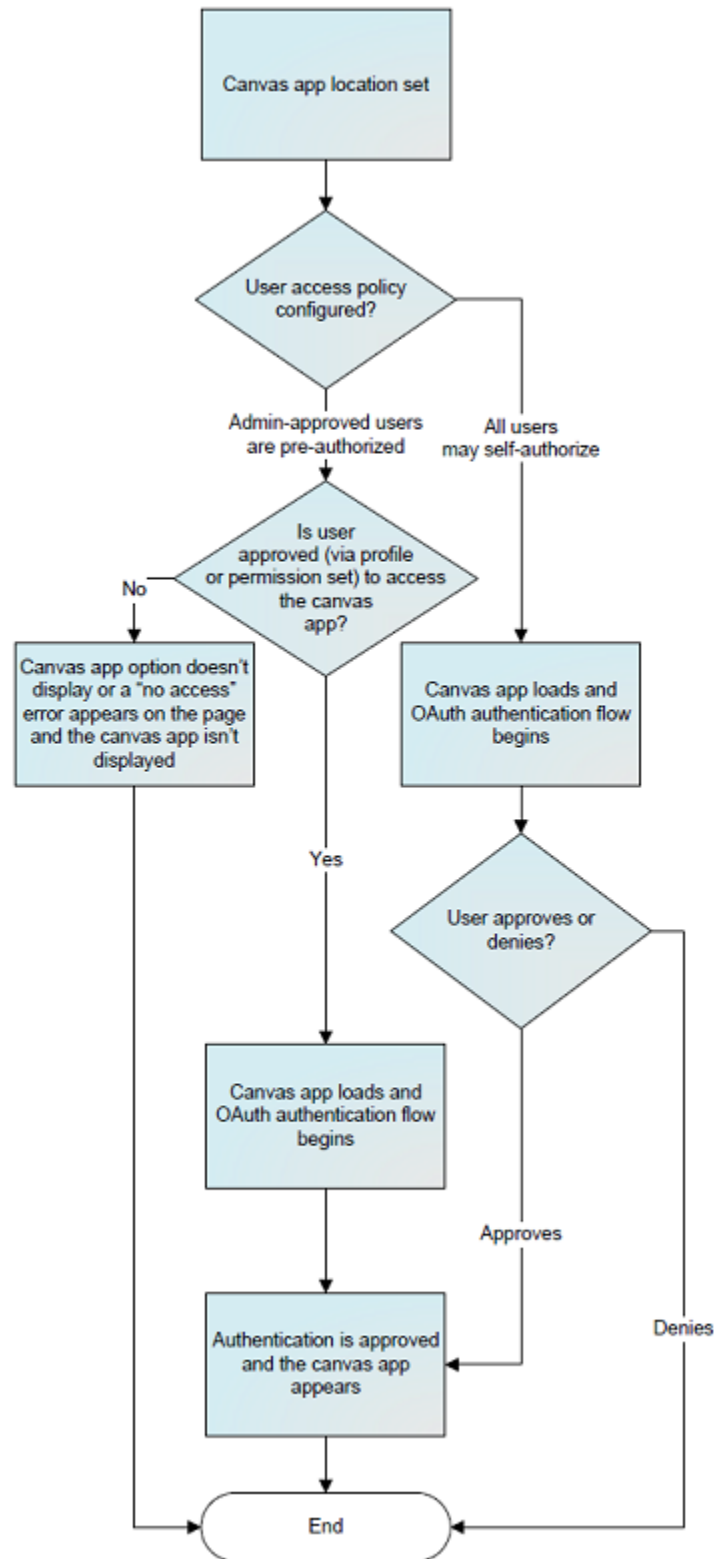
If you have an existing connected app that uses OAuth authentication and you want to expose that app as a canvas app, you have two options. First, you can edit the existing app (create a new version) and add the canvas app information to it. This means your app can continue to use the same client ID and consumer secret. The second option is to create a new canvas app. If you do this, you'll get a new client ID and consumer secret, and you'll need to update your app with that information.

### See Also:

[Authentication](#)  
[Canvas App User Flow—OAuth](#)

## Canvas App User Flow—OAuth

If your canvas app uses OAuth authentication, the user experience varies depending on where the canvas app is located in the user interface and how the user access is set. This diagram shows the user flow for a canvas app that uses OAuth authentication.

**See Also:**

[OAuth Authentication](#)  
[Initiating OAuth Flow](#)

## Initiating OAuth Flow

The following code examples show you how to start the authorization process in your canvas app using OAuth.

```
<html>
<head>
  <script type="text/javascript" src="/sdk/js/canvas-all.js"></script>
</head>
<body>
  <script>

    function loginHandler(e) {
      var uri;
      if (! Sfdc.canvas.oauth.loggedin()) {
        uri = Sfdc.canvas.oauth.loginUrl();
        Sfdc.canvas.oauth.login(
          {uri : uri,
           params: {
             response_type : "token",
             client_id : "3MVG9lKcPoNINVBliGmW.8dAn4L5HwY VBzxbW5FFdzvU0re2
             f7o9aHJNUpY9ACdh.3SUGw5rF2nSsC9_cRqzD",
             redirect_uri : encodeURIComponent(
               "https://demoapp1234.herokuapp.com/sdk/callback.html")
           }});
      }
      else {
        Sfdc.canvas.oauth.logout();
        login.innerHTML = "Login";
        Sfdc.canvas.byId("oauth").innerHTML = "";
      }
      return false;
    }

    // Bootstrap the page once the DOM is ready.
    Sfdc.canvas(function() {
      // On Ready...
      var login = Sfdc.canvas.byId("login"),
          loggedIn = Sfdc.canvas.oauth.loggedin(),
          token = Sfdc.canvas.oauth.token()
      login.innerHTML = (loggedIn) ? "Logout" : "Login";
      if (loggedIn) {
        // Only displaying part of the OAuth token for better formatting.
        Sfdc.canvas.byId("oauth").innerHTML = Sfdc.canvas.oauth.token()
          .substring(1,40) + "...";
      }
      login.onclick=loginHandler;
    });
  </script>
  <h1 id="header">Force.com Canvas OAuth App</h1>
  <div>
    access_token = <span id="oauth"></span>
  </div>
  <div>
    <a id="login" href="#">Login</a><br/>
  </div>
</body>
</html>
```

### See Also:

[OAuth Authentication](#)

## Getting Context in Your Canvas App

When you authenticate your canvas app using signed request, you get the [CanvasRequest](#) object (which contains the Context object) as part of the POST to the canvas app URL. If you're authenticating using OAuth, or you want to make a call to get context information, you can do so by making a JavaScript call.

The following code sample is an example of a JavaScript call to get context. This code creates a link with the text "Get Context" which then calls the `Sfdc.canvas.client.ctx` function.

```
<script>
  function callback(msg) {
    if (msg.status !== 200) {
      alert("Error: " + msg.status);
      return;
    }
    alert("Payload: ", msg.payload);
  }

  var ctxlink = Sfdc.canvas.byId("ctxlink");
  var client = Sfdc.canvas.oauth.client();
  ctxlink.onclick=function() {
    Sfdc.canvas.client.ctx(callback, client);
  }
</script>

<a id="ctxlink" href="#">Get Context</a>
```

### See Also:

[Force.com Canvas SDK](#)

[Context](#)

## Cross-Domain XHR

Canvas apps are loaded on a Salesforce page in an iFrame. Therefore, the canvas app (in its own domain) can't make XHR (XML HTTP request) calls back to the \*.salesforce.com domain. You can develop and deploy your own proxies as part of the SDK, however, Force.com Canvas provides a client-side proxy written in JavaScript. This proxy enables client-side XHR calls back to Salesforce.

If you use this proxy from the client to make an XHR request, the API forwards the request to the outer iFrame and the request is submitted on your behalf. When the request is complete, the SDK calls the client's callback function with the results. Here are some examples of how you can make XHR calls:

- [Getting a List of Chatter Users](#)
- [Posting to a Chatter Feed](#)



**Note:** The SDK supports cross-domain XHR calls, however, it shouldn't be used to make same-domain calls.

### See Also:

[Force.com Canvas SDK](#)

## Getting a List of Chatter Users

The following code example shows a call to return a list of Chatter users.

```
// Paste the signed request string into a JavaScript object for easy access.
var sr = JSON.parse('<%=signedRequestJson%>');
// Reference the Chatter user's URL from Context.Links object.
var chatterUsersUrl = sr.context.links.chatterUsersUrl;

// Make an XHR call back to salesforce through the supplied browser proxy.
Sfdc.canvas.client.ajax(chatterUsersUrl,
  {client : sr.client,
   success : function(data){
     // Make sure the status code is OK.
     if (data.status === 200) {
       // Alert with how many Chatter users were returned.
       alert("Got back " + data.payload.users.length +
         " users"); // Returned 2 users
     }
   }
  });
```

### See Also:

[Cross-Domain XHR](#)

[Context](#)

[Links](#)

## Posting to a Chatter Feed

The following code example shows a call to post an item to the context user's Chatter feed.

```
var sr = JSON.parse('<%=signedRequestJson%>');
// Reference the Chatter user's URL from Context.Links object.
var url = sr.context.links.chatterFeedsUrl+"/news/"+sr.context.user.userId+"/feed-items";
var body = {body : {messageSegments : [{type: "Text", text: "Some Chatter Post"}]}};

Sfdc.canvas.client.ajax(url,
  {client : sr.client,
   method: 'POST',
   contentType: "application/json",
   data: JSON.stringify(body),
   success : function(data) {
     if (201 === data.status) {
       alert("Success");
     }
   }
  });
```

### See Also:

[Cross-Domain XHR](#)

[Context](#)

[Links](#)

## Resizing a Canvas App

Force.com Canvas provides methods for resizing a canvas app. Full reference documentation for these methods can be found in the [SDK](#) and [here](#).

- `autogrow`—Starts or stops a timer that checks the content size of the canvas iFrame and adjusts the frame. See [Automatically Resizing a Canvas App](#).
- `resize`—Informs the parent window to resize the canvas iFrame. See [Explicitly Resizing a Canvas App](#).
- `size`—Returns the current size of the canvas iFrame. See [Getting the Size of a Canvas App](#).
- `subscribe`—Subscribes to parent events. Currently, `canvas.scroll` (of the parent) is the only supported parent event in the `canvas` namespace. See [Subscribing to Parent Events](#).

### See Also:

[Force.com Canvas SDK](#)

## Automatically Resizing a Canvas App

The following code example shows how to call the `autogrow` method to resize a canvas app. Use this method when your content will change size, but you're not sure when.

```
// Turn on auto grow with default settings.
Sfdc.canvas(function() {
    sr = JSON.parse('<%=signedRequestJson%>');
    Sfdc.canvas.client.autogrow(sr.client);
});

// Turn on auto grow with polling interval of 100ms (milliseconds).
Sfdc.canvas(function() {
    sr = JSON.parse('<%=signedRequestJson%>');
    Sfdc.canvas.client.autogrow(sr.client, true, 100);
});

// Turn off auto grow.
Sfdc.canvas(function() {
    sr = JSON.parse('<%=signedRequestJson%>');
    Sfdc.canvas.client.autogrow(sr.client, false);
});
```

### See Also:

[Resizing a Canvas App](#)

[Explicitly Resizing a Canvas App](#)

## Explicitly Resizing a Canvas App

The following code example shows how to call the `resize` method to resize a canvas app. If you don't specify the height and width parameters, the parent window attempts to determine the height of the canvas app based on its content and then set the iFrame width and height accordingly.

```
// Automatically determine the size.
Sfdc.canvas(function() {
    sr = JSON.parse('<%=signedRequestJson%>');
    Sfdc.canvas.client.resize(sr.client);
});
```

```
});

// Set the height and width explicitly.
Sfdc.canvas(function() {
    sr = JSON.parse('<%=signedRequestJson%>');
    Sfdc.canvas.client.resize(sr.client, {height : "1000px", width : "900px"});
});

// Set only the height.
Sfdc.canvas(function() {
    sr = JSON.parse('<%=signedRequestJson%>');
    Sfdc.canvas.client.resize(sr.client, {height : "1000px"});
});
```

**See Also:**[Resizing a Canvas App](#)[Automatically Resizing a Canvas App](#)[Getting the Size of a Canvas App](#)

## Getting the Size of a Canvas App

The following code example shows how to call the `size` method to get the size of the canvas app. The `console.log` function outputs the frame sizes so you can see the sizes change as you resize the canvas app.

```
// Get the canvas app sizes in the sizes object.
var sizes = Sfdc.canvas.client.size();

console.log("contentHeight; " + sizes.heights.contentHeight);
console.log("pageHeight; " + sizes.heights.pageHeight);
console.log("scrollTop; " + sizes.heights.scrollTop);
console.log("contentWidth; " + sizes.widths.contentWidth);
console.log("pageWidth; " + sizes.widths.pageWidth);
console.log("scrollLeft; " + sizes.widths.scrollLeft);

// Resize the canvas app.
Sfdc.canvas(function() {
    sr = JSON.parse('<%=signedRequestJson%>');
    Sfdc.canvas.client.autogrow(sr.client);
});
```

**See Also:**[Resizing a Canvas App](#)[Subscribing to Parent Events](#)

## Subscribing to Parent Events

The following code example shows how to call the `subscribe` method so that a canvas app can subscribe to parent events. This example handles the `onscroll` event that fires when the user scrolls in the parent window.

```
//Subscribe to the parent window onscroll event.
Sfdc.canvas(function() {
    sr = JSON.parse('<%=signedRequestJson%>');
    // Capture the onScrolling event of the parent window
    Sfdc.canvas.client.subscribe(sr.client,
        {name : 'canvas.scroll', onData : function (event) {
            console.log("Parent's contentHeight; " + event.heights.contentHeight);
            console.log("Parent's pageHeight; " + event.heights.pageHeight);
        }
    });
});
```

```

        console.log("Parent's scrollTop; " + event.heights.scrollTop);
        console.log("Parent's contentWidth; " + event.widths.contentWidth);
        console.log("Parent's pageWidth; " + event.widths.pageWidth);
        console.log("Parent's scrollLeft; " + event.widths.scrollLeft);
    }
}
);
});

```

**See Also:**[Resizing a Canvas App](#)

## Implementing Canvas App Events

Events provide a JavaScript-based way to send and receive events between canvas apps. Use events to enable communication between multiple canvas apps on a single page.

One scenario might be a page on which you expose two custom apps as canvas apps: a travel and expense app, and an approvals app. You can create an event so that when the status of an expense report changes, that event gets raised and contains data (in JSON format) about that expense report. The approvals canvas app subscribes to that event and specifies a function that's called when the event is raised. When the status is changed, the approvals app receives the event and the specified function runs.

Force.com Canvas provides methods for implementing custom events in a canvas app. Full reference documentation for these methods can be found in the [SDK](#) and [here](#).

- `publish`—Creates a custom event to which other canvas apps can subscribe. See [Creating a Canvas App Event](#).
- `subscribe`—Subscribes to a parent event or custom event. This method can be used to subscribe to multiple events. See [Subscribing to a Canvas App Event](#).
- `unsubscribe`—Unsubscribe from a parent event or custom event. This method can be used to unsubscribe from multiple events. See [Unsubscribing from a Canvas App Event](#).



**Note:** The `subscribe` and `unsubscribe` methods can also be used to subscribe to a single Streaming API event.

**See Also:**[Force.com Canvas SDK](#)[Canvas App Events Considerations](#)[Using Streaming API in a Canvas App](#)

## Canvas App Events Considerations

Keep these considerations in mind when implementing canvas app events:

- We recommend that you use a namespace when naming events, but it's not required.
- The event namespace is different than the organization namespace in Salesforce. However, if you use namespaces, we recommend that you make the event namespace the same as the organization namespace.
- The namespace must be a string with no periods in it. For example, `my.name.space.statusChanged` is invalid. An example of a valid event name with a namespace is `mynamespace.statusChanged`.
- These names are reserved and can't be used as a namespace:

◇ `canvas`



- ◇ chatter
- ◇ force
- ◇ publisher
- ◇ salesforce
- ◇ sfdc

- Events work only between canvas apps on the same page. If you have a canvas app on the Chatter tab, that app can't subscribe to events published by a canvas app on a Visualforce page.
- You can subscribe to more than one custom event in a `subscribe` call.
- You can subscribe to only one Streaming API event in a `subscribe` call.
- You can't subscribe to a custom event and a Streaming API event with the same `subscribe` call.
- If you define multiple events with the same name in an array, only the last event defined is available. In this example, the last event where the Status is Negotiating is the one that's used.

```
Sfdc.canvas.client.subscribe(sr.client, [
{
  name : "mynamespace.statusChanged",
  payload : {status : 'Closed'}
},
{
  name: "mynamespace.statusChanged",
  payload : {status : 'Negotiating'}
}]);
```

This is also true for Streaming API events.

### See Also:

[Implementing Canvas App Events](#)

## Creating a Canvas App Event

The following code example shows how to call the `publish` method to create a canvas app event. If you're using a namespace, the event name must be prefaced by the namespace. For example, `namespace.eventName`.

```
Sfdc.canvas(function() {
  sr = JSON.parse('<%=signedRequestJson%>');
  Sfdc.canvas.client.publish(sr.client,
    {name : "mynamespace.statusChanged", payload : {status : 'Completed'}});
});
```

### See Also:

[Implementing Canvas App Events](#)

[Creating a Canvas App Event](#)

## Subscribing to a Canvas App Event

### Subscribing to a Custom Event

The following code example shows how to call the `subscribe` method to subscribe to a canvas app event.

```
// Subscribe to a custom event.
Sfdc.canvas(function() {
  sr = JSON.parse('<%=signedRequestJson%>');
  Sfdc.canvas.client.subscribe(sr.client,
    {name : 'mynamespace.statusChanged', onData : function (event) {
      console.log("Subscribed to custom event ", event);
    }}
  );
});
```

### Subscribing to Multiple Custom Events

The following code example shows how to call the `subscribe` method to subscribe to multiple canvas app events. The events you subscribe to can be in different namespaces or might not have a namespace. When a canvas app subscribes to an event, it creates an association between an event (in the other canvas app) and a function (in the subscribing canvas app).

```
// Subscribe to multiple events.
Sfdc.canvas(function() {
  sr = JSON.parse('<%=signedRequestJson%>');
  Sfdc.canvas.client.subscribe(sr.client, [
    {name : 'mynamespace.statusChanged', onData : handler1},
    {name : 'anothernamespace.tripCancelled', onData : handler2},
  ]);
});
```

Using the travel and expense and approval canvas app examples, your approvals canvas app has two functions: `handler1` and `handler2`. That canvas app then subscribes to two events in the travel and expense canvas app:

`mynamespace.statusChanged` and `mynamespace.tripCancelled`. When the `mynamespace.statusChanged` event is received by the approvals app, function `handler1` is called. When the `anothernamespace.tripCancelled` event is received by the approvals app, function `handler2` is called.

### See Also:

- [Implementing Canvas App Events](#)
- [Subscribing to Parent Events](#)
- [Unsubscribing from a Canvas App Event](#)

## Unsubscribing from a Canvas App Event

### Unsubscribing from a Custom Event

The following code example shows how to call the `unsubscribe` method to unsubscribe from a canvas app event.

```
// Unsubscribe from a custom event.
Sfdc.canvas(function() {
  sr = JSON.parse('<%=signedRequestJson%>');
  Sfdc.canvas.client.unsubscribe(sr.client, {name : "mynamespace.statusChanged"});
});
```

## Unsubscribing from Multiple Custom Events

The following code example shows how to call the `unsubscribe` method to unsubscribe from multiple canvas app events. The events you subscribe to can be in different namespaces or might not have a namespace.

```
// Unsubscribe from multiple events.
Sfdc.canvas(function() {
    sr = JSON.parse('<%=signedRequestJson%>');
    Sfdc.canvas.client.unsubscribe(sr.client, ['mynamespace.statusChanged',
    "anothernamespace.tripCancelled"]);
});
```

### See Also:

- [Implementing Canvas App Events](#)
- [Subscribing to a Canvas App Event](#)
- [Subscribing to Parent Events](#)

## Using Streaming API in a Canvas App

Force.com Canvas provides an event and methods that enable canvas apps to listen for Streaming API notifications.

- `sfdc.streamingapi`—JavaScript event that you create and associate with a Streaming API channel defined by a PushTopic. See [Using the Streaming API Event](#).
- `subscribe`—Subscribes to the `sfdc.streamingapi` event that you define. See [Subscribing to a Streaming API Event](#).
- `unsubscribe`—Unsubscribes from the `sfdc.streamingapi` event. See [Unsubscribing from a Streaming API Event](#).

### See Also:

- [Using the Streaming API Event](#)
- [Subscribing to a Streaming API Event](#)
- [Unsubscribing from a Streaming API Event](#)
- [Force.com Streaming API Developer's Guide](#)

## Using the Streaming API Event

The Force.com Canvas SDK contains an event called `sfdc.streamingapi` that lets you define an event in your canvas app and associate that event with a Streaming API channel. You then use the `subscribe` method to subscribe to the event and receive Streaming API notifications.

For example, in Salesforce, you can create a Streaming API channel that receives notifications when an `InvoiceStatement` is updated and the Status changes to Closed. In your canvas app, you can then create an event associated with that channel and subscribe to it. In Salesforce, whenever an invoice statement is closed, the activated canvas app receives the notification and can perform an action such as displaying a message to the user.

Here are some considerations when defining the Streaming API event:

- The event takes a single parameter that contains the PushTopic name.

- The PushTopic name must be prefaced by “/topic/.”

```
{name:"sfdc.streamingapi", params:{topic:"/topic/myPushTopicName"}}
```

### See Also:

[Using Streaming API in a Canvas App](#)

[Subscribing to a Streaming API Event](#)

## Subscribing to a Streaming API Event

This code example shows how to call the `subscribe` method so that a canvas app can subscribe to a Streaming API event. When you subscribe to an event, you call the standard `sfdc.canvas.client.subscribe` method that you use to subscribe to a canvas app event. When you call the `subscribe` method, you must pass in the client and the Streaming API event. Only canvas apps that are open and subscribed to the event can receive Streaming API notifications.

In this example, the `onComplete` method specifies the function that runs after the code successfully subscribes to the event. The `onData` method specifies the function that runs when a Streaming API notification is received by the event.

```
// Subscribe to Streaming API events.
// The PushTopic to subscribe to must be passed in.
// The 'onComplete' method may be defined,
// and will fire when the subscription is complete.
Sfdc.canvas(function() {
    sr = JSON.parse('<%=signedRequestJson%>');
    var handler1 = function(){ console.log("onData done");},
    handler2 = function(){ console.log("onComplete done");};
    Sfdc.canvas.client.subscribe(sr.client,
        {name : 'sfdc.streamingapi', params:{topic:"/topic/InvoiceStatements"}},
        onData : handler1, onComplete : handler2}
    );
});
```

When you call the `subscribe` method, a REST call is made to ensure that the canvas app has the OAuth scope needed to connect to the Streaming API. Therefore, each time a canvas app subscribes to a Streaming API event, one API call is used and is counted against the organization’s total API requests limits. The canvas app needs at least the “Access and Manage Your Data (API)” OAuth scope to connect to the Streaming API.

If the call to the `subscribe` method is successful, the `onComplete` method is called with a payload of `{success:true,handle:handle}`. The `handle` is an array that contains the name of the Streaming API channel being subscribed to and the `subscriptionId` is an integer that contains a unique ID. For example, `["/topics/InvoiceStatements", subscriptionId]`. If the call to the `subscribe` method fails, the `onComplete` method is called with a payload of `{success:false,errorMessage:msg}`. The `msg` is a string that contains the cause of the error.

To receive Streaming API notifications, you must create a channel defined by a PushTopic. For more information, see “Step 2: Create a PushTopic” in the [Force.com Streaming API Developer’s Guide](#).

### See Also:

[Using the Streaming API Event](#)

[Unsubscribing from a Streaming API Event](#)

## Unsubscribing from a Streaming API Event

This code example shows how to call the `unsubscribe` method so that a canvas app can unsubscribe from a Streaming API event.

```
//Unsubscribe from Streaming API events.  
//The PushTopic to unsubscribe from must be passed in.  
Sfdc.canvas(function() {  
    sr = JSON.parse('<%=signedRequestJson%>');  
    Sfdc.canvas.client.unsubscribe(sr.client, {name : 'sfdc.streamingapi',  
        params:{topic:"/topic/InvoiceStatements"}}});  
});
```

### See Also:

[Using the Streaming API Event](#)

[Subscribing to a Streaming API Event](#)

## Chapter 5

# Canvas Apps and Visualforce Pages

---

### In this chapter ...

- [Visualforce Page Code Examples](#)
- [Visualforce Considerations](#)
- [apex:canvasApp Component](#)

In addition to standard canvas apps, Force.com Canvas also lets you expose a canvas app on a Visualforce page. This means you can display a canvas app anywhere you can display a Visualforce page.

Developers can use Visualforce pages to:

- Override standard buttons, such as the New button for accounts, or the Save button for contacts
- Override tab overview pages, such as the Accounts tab home page
- Define custom tabs
- Embed components in detail page layouts
- Create dashboard components or custom help pages
- Customize, extend, or integrate the sidebars in the Salesforce Console (custom console components)

To host a canvas app on a Visualforce page, use the `<apex:canvasApp>` component.

### See Also:

[Visualforce Page Code Examples](#)

[Visualforce Considerations](#)

[apex:canvasApp Component](#)

## Visualforce Page Code Examples

You can display a canvas app on a Visualforce page in a number of ways. These examples show the different ways to reference the canvas app using `applicationName`, `developerName`, and `namespacePrefix`.

### Object Detail Page

The following code snippet is an example of how to display a canvas app on an Account page. The code specifies the size of the canvas app to be 400 pixels high and 750 pixels wide. This examples specifies the canvas app using the `applicationName` and `namespacePrefix`.

```
<apex:page standardController="Account">

    <apex:canvasApp applicationName="Test Inline Visualforce"
        namespacePrefix="testorg"
        height="400px" width="750px"/>

</apex:page>
```

### Standard Page

The following code snippet is an example of how to display a canvas app on a Visualforce page. The code specifies the size of the canvas app to be 1,000 pixels high and 800 pixels wide. In addition, the code passes three custom parameters to the canvas app. This examples specifies the canvas app using the `developerName` and `namespacePrefix`.

```
<apex:page>

    <apex:canvasApp developerName="Test_Standard_Visualforce"
        namespacePrefix="testorg" height="1000px" width="800px"
        parameters="{p1:'value1',p2:'value2',p3:'value3' }"/>

</apex:page>
```

### Standard Page with a Border and Scrolling

The following code snippet is an example of how to display a canvas app with some additional UI enhancements on a Visualforce page. The code specifies the size of the canvas app to be 100 pixels high and 500 pixels wide. In addition, the code specifies that there should be a border of 2 pixels around the canvas app and that scrolling should be enabled. This examples specifies the canvas app using only the `applicationName` (this is only valid in the organization in which the canvas app was created, and if that organization doesn't have a `namespacePrefix`).

```
<apex:page>

    <apex:canvasApp applicationName="Test Scrolling Visualforce"
        height="100px" width="500px"
        border="2" scrolling="yes"/>

</apex:page>
```

### See Also:

[Canvas Apps and Visualforce Pages](#)  
[Visualforce Considerations](#)  
[Visualforce Developer's Guide](#)  
[apex:canvasApp Component](#)

## Visualforce Considerations

Keep the following considerations in mind when using the `<apex:canvasApp>` component:

- The `<apex:canvasApp>` component is available only in organizations that have Force.com Canvas enabled and in Visualforce pages at version 27.0 or higher.
- If you include a canvas app on an object detail layout, you must provide the height of the canvas app in the page layout as well as in the `<apex:canvasApp>` component.
- Location—If the canvas app is in a Visualforce page, then the `Environment.displayLocation` field contains the value `Visualforce`.

### See Also:

[Canvas Apps and Visualforce Pages](#)  
[apex:canvasApp Component](#)

## apex:canvasApp Component

Use this component to display a canvas app on a Visualforce page. This table lists the component attributes.

Attribute	Type	Description
<code>applicationName</code>	String	Name of the canvas app. Either <code>applicationName</code> or <code>developerName</code> is required.
<code>border</code>	String	Width of the canvas app border, in pixels. If not specified, defaults to 0 pixels.
<code>canvasId</code>	String	Unique ID of the canvas app window. Use this attribute when targeting events to the canvas app.
<code>containerId</code>	String	<p>ID of the HTML element in which the canvas app is rendered. If not specified, defaults to null. The container specified by this attribute can't appear after the <code>&lt;apex:canvasApp&gt;</code> component. These code examples show valid usage of the <code>&lt;div&gt;</code> container and the <code>containerId</code> attribute:</p> <pre>&lt;apex:page&gt;   &lt;div id="container1"&gt;&lt;/div&gt;   &lt;apex:canvasApp applicationName="myApp"     containerId="container1"/&gt; &lt;/apex:page&gt;  &lt;apex:page&gt;   &lt;div id="container1"&gt;     &lt;apex:canvasApp       applicationName="myApp"       containerId="container1"/&gt;   &lt;/div&gt; &lt;/apex:page&gt;</pre> <p>This code example shows invalid usage of the <code>&lt;div&gt;</code> container and the <code>containerId</code> attribute:</p> <pre>&lt;apex:page&gt;   &lt;apex:canvasApp applicationName="myApp"</pre>



Attribute	Type	Description
		<pre> containerId="container1"/&gt;   &lt;div id="container1"&gt;     &lt;/div&gt; &lt;/apex:page&gt; </pre>
developerName	String	Internal name of the canvas app. You specify this value in the API Name field when you expose the canvas app by creating a connected app. Either developerName or applicationName is required.
height	String	Canvas app window height, in pixels. If not specified, defaults to 900 pixels.
id	String	Unique identifier that allows the <apex:canvasApp> component to be referenced by other components on the page.
namespacePrefix	String	Namespace value of the Developer Edition organization in which the canvas app was created. You can set a namespace only in a Developer Edition organization, so this is optional if the canvas app was created in a different type of organization. If not specified, defaults to null.
onCanvasAppError	String	Name of the JavaScript function to be called if the canvas app fails to render.
onCanvasAppLoad	String	Name of the JavaScript function to be called after the canvas app loads.
parameters	String	Object representation of parameters passed to the canvas app. Supply in JSON format or as a JavaScript object literal. Here's an example of parameters in a JavaScript object literal: {param1: 'value1', param2: 'value2'}. If not specified, defaults to null.
rendered	Boolean	Specifies whether the component is rendered on the page. If not specified, defaults to true.
scrolling	String	Specifies whether the canvas app window uses scroll bars. If not specified, defaults to no.
width	String	Canvas app window width, in pixels. If not specified, defaults to 800 pixels.

**See Also:**

[Canvas Apps and Visualforce Pages](#)

[Visualforce Developer's Guide](#)

[Visualforce Page Code Examples](#)

## Chapter 6

### Canvas Apps in the Publisher—Pilot

#### In this chapter ...

- [Set Canvas App Location and Create the Action](#)
- [Create the Action](#)
- [Force.com Canvas SDK Publisher Events](#)
- [Publisher Context Considerations](#)
- [Publisher Canvas App Access Considerations](#)



**Note:** Support for Force.com Canvas apps in the publisher, the Chatter feed, and Salesforce1 is currently available through a pilot program and is available in all new Development Edition organizations. For information on enabling it for your organization, contact [salesforce.com](mailto:salesforce.com).

Force.com Canvas enables you to expose your canvas apps as publisher actions. The publisher allows users access to the most common actions in your organization. You can expand the publisher to include a canvas app so that users can leverage the common actions of a canvas app. These actions can then integrate with the Chatter feed and create a feed post specific to the action that was performed.

Developers can use canvas apps in the publisher to:

- Add content from a Web application into the Chatter publisher.
- Create a custom action that exposes a canvas app.
- Integrate your canvas app directly into the publisher life cycle: post from your canvas app into the Chatter feed, use the Share button functionality, and designate where to post your message.

For example, you might have a canvas app that your users use to log their hours worked. You can create a publisher action that allows a user to open that canvas app in the publisher so they can quickly submit a time record, all right from within the publisher.

Users can still access the canvas app in the standard way for full functionality; but the canvas app in the publisher provides quick access to the most common functions of your app. A user can select the publisher action and create a Chatter feed item that can be a text post, a link post, or even a canvas post.

#### See Also:

[Set Canvas App Location and Create the Action](#)  
[Canvas Apps in the Chatter Feed—Pilot](#)

## Set Canvas App Location and Create the Action

To add a canvas app to the publisher, you must set the location and create the action when you create the canvas app.



**Note:** Support for Force.com Canvas apps in the publisher, the Chatter feed, and Salesforce1 is currently available through a pilot program and is available in all new Development Edition organizations. For information on enabling it for your organization, contact salesforce.com.

1. In Salesforce, from Setup, click **Create > Apps**.
2. In the Connected Apps related list, click **New**. Fill out the fields for your canvas app. See [Create the Canvas App](#).
3. In the `Locations` field, select Publisher.

You must select this location for your canvas app to appear in the publisher.

4. In the Canvas App Settings section, select the `Create Actions Automatically` field.

This creates a quick action for the canvas app.

For the canvas app to appear as a publisher action, you must add the action to the global layout. See “Customizing Global Publisher Layouts” in the Salesforce Help.

### See Also:

[Where Canvas Apps Appear](#)

[Create the Action](#)

[Canvas Apps in the Publisher—Pilot](#)

## Create the Action

If you didn't select the `Create Actions Automatically` field when you created the canvas app, then you'll need to create the action manually.



**Note:** Support for Force.com Canvas apps in the publisher, the Chatter feed, and Salesforce1 is currently available through a pilot program and is available in all new Development Edition organizations. For information on enabling it for your organization, contact salesforce.com.

1. From Setup, click **Create > Global Actions**
2. Click **New Action**.
3. In the `Action Type` field, select Custom Canvas.
4. In the `Canvas App` field, select the canvas app that you want to appear as an action.

Only canvas apps that have a location of Publisher will appear in this field.

5. In the `Height` field, enter the height of the canvas app in pixels.

This is the initial height of the canvas app when it appears in the publisher. You can use the Force.com Canvas SDK `resize()` method to change the height up to a maximum of 500 pixels.

6. In the `Label` field, enter a value.

This value appears as the publisher action title in the user interface.

7. In the `Name` field, enter a unique value with no spaces.

8. Optionally, in the `Icon` field, you can upload an icon by clicking **Change Icon**. You must upload the icon as a static resource before you can change it here.

9. Click **Save**.

For the canvas app to appear as a publisher action, you must add the action to the global layout. See “Customizing Global Publisher Layouts” in the Salesforce Help.

### See Also:

[Set Canvas App Location and Create the Action](#)  
[Publisher Context Considerations](#)

## Force.com Canvas SDK Publisher Events

When you expose a canvas app in the publisher, you can use well-defined events to enable communication between the canvas app and the publisher.



**Note:** Support for Force.com Canvas apps in the publisher, the Chatter feed, and Salesforce1 is currently available through a pilot program and is available in all new Development Edition organizations. For information on enabling it for your organization, contact [salesforce.com](https://salesforce.com).

Your canvas app can subscribe and publish these events to more tightly integrate with the publisher framework. For example, you can activate the standard Chatter Share button to post a Chatter feed item. You can also access the post text that the user enters in the *What are you working on?* field in the publisher and combine that with content from your app.

Field	Description
<code>publisher.clearPanelState</code>	Fired by the publisher when the canvas app is deactivated or hidden. This can happen when the user selects a different application in the publisher or after the Share button has been clicked. A Visualforce page can also listen for this event.
<code>publisher.failure</code>	Fired by the publisher when an error condition is encountered such as when invalid data has been submitted. For example: <ul style="list-style-type: none"> <li>The text in the feed is too long</li> <li>The canvas app you’re attempting to publish to the feed doesn’t exist</li> <li>The canvas app URL is invalid</li> </ul> The canvas app should listen for this event and alert the user that an error occurred and the post didn’t get created.
<code>publisher.getPayload</code>	Fired by the publisher when the Share button is clicked. The payload contains information such as the text entered into the <i>What are you working on?</i> field and who the feed item is being shared with.
<code>publisher.setupPanel</code>	Fired by the publisher when the Chatter feed page is initially loaded.
<code>publisher.setPayload</code>	Fired by the canvas app to indicate to the publisher that the content being sent to the publisher should be shared in the feed item. This event is in response to <code>publisher.getPayload</code> and contains information about the feed item you’re trying to create. You can create three feed item types: <ul style="list-style-type: none"> <li><code>TextPost</code></li> <li><code>LinkPost</code></li> <li><code>CanvasPost</code></li> </ul>

Field	Description
<code>publisher.setValidForSubmit</code>	<p>Fired by the canvas app to indicate to the publisher that the canvas app is ready to submit a payload. After this event fires, the Share button becomes active.</p> <p>This code snippet enables the Share button:</p> <pre>         \$\$client.publish(sr.client,             {name : 'publisher.setValidForSubmit',               payload : true});       </pre>
<code>publisher.showPanel</code>	<p>Fired by the publisher when the user selects a canvas app in the publisher. This event indicates that the canvas app is being displayed. A Visualforce page can also listen for this event.</p>
<code>publisher.success</code>	<p>Fired by the publisher after the Share button is clicked and data is successfully submitted.</p>

## Sequence of Publisher Events

Here's the order of publisher events from the perspective of the canvas app:

1. The canvas app listens for `publisher.setupPanel`.
2. The canvas app listens for `publisher.showPanel`.
3. The user interacts with the canvas app, for example, clicks a button or enters some text. The canvas app does any validation required and then fires `publisher.setValidForSubmit`. As a result, the publisher then enables the Share button.
4. The canvas app listens for `publisher.getPayload`.
5. The canvas app fires `publisher.setPayload`.
6. The canvas app listens for `publisher.success`.
7. The canvas app listens for `publisher.failure`.
8. The canvas app listens for `publisher.clearPanelState`.

## See Also:

[Create the Action](#)

[Publisher Context Considerations](#)

## Publisher Context Considerations

When you display a canvas app inside the publisher, the context information you receive from the signed request or from a `getContext()` call contains information specific to the publisher:



**Note:** Support for Force.com Canvas apps in the publisher, the Chatter feed, and Salesforce1 is currently available through a pilot program and is available in all new Development Edition organizations. For information on enabling it for your organization, contact [salesforce.com](https://salesforce.com).

- Location—If the canvas app is in the publisher, then the `Environment.displayLocation` field contains the value `Publisher`.
- Size—The `Environment.Dimensions` object contains information about the size of the canvas app.
  - ◇ The canvas app height will be the height you specify in the quick action that you created.
  - ◇ If you selected `Create Actions Automatically` when you created the canvas app, the canvas app height defaults to 200 pixels.

- ◇ The canvas app width defaults to 521 pixels, which is the same as the maximum width of a canvas app in the publisher.
- ◇ The maximum height of a canvas app in the publisher is 500 pixels.
- ◇ The maximum width of a canvas app in the publisher is 521 pixels.
- ◇ This code snippet shows the default size values of a canvas app in the publisher:

```
"dimensions":  
{  
  "width": "521px",  
  "height": "200px",  
  "maxHeight": "500px",  
  "maxWidth": "521px"  
}
```

- ◇ The publisher is a fixed width of 521 pixels. For example, if you resize your canvas app to be 400 pixels, the publisher width remains 521 pixels.
- ◇ You can use the `resize()` method in the Force.com Canvas SDK to change the values of your canvas app up to the `maxHeight` and `maxWidth`.

### See Also:

[Create the Action](#)

[Environment](#)

[Dimensions](#)

[Resizing a Canvas App](#)

[Publisher Canvas App Access Considerations](#)

[Canvas Apps in the Chatter Feed—Pilot](#)

## Publisher Canvas App Access Considerations

When modifying a canvas app that appears in the publisher, keep these considerations in mind:



**Note:** Support for Force.com Canvas apps in the publisher, the Chatter feed, and Salesforce1 is currently available through a pilot program and is available in all new Development Edition organizations. For information on enabling it for your organization, contact [salesforce.com](https://salesforce.com).

- If the canvas app has a quick action associated with it, then you can't delete the canvas app or remove the Publisher location. You must first delete the quick action.
- If the user doesn't have access to the canvas app through profiles or permission sets and they select the app in the publisher, then they'll receive an error.
- If the canvas app attempts to perform an action for which the user doesn't have permissions, then that action will fail and the canvas app will receive an error. For example, if the app tries to create a Merchandise record but the user doesn't have create permission on Merchandise, then the app will receive an error. The canvas app should then relay the error to the user.

## Chapter 7

### Canvas Apps in the Chatter Feed—Pilot

#### In this chapter ...

- [Chatter Feed Context Considerations](#)
- [Chatter Feed Canvas App Access Considerations](#)



**Note:** Support for Force.com Canvas apps in the publisher, the Chatter feed, and Salesforce1 is currently available through a pilot program and is available in all new Development Edition organizations. For information on enabling it for your organization, contact [salesforce.com](mailto:salesforce.com).

Force.com Canvas enables you to expose your canvas apps as feed items. The feed gives users information about what's happening inside of Salesforce and information about records and groups they're following.

Developers can use canvas apps in the feed to:

- Post to the Chatter feed from a canvas app in the publisher or through the Chatter API.
- Display a canvas app inside a Chatter feed item.

When you create a canvas app Chatter feed item, it contains a thumbnail image, a link title, and a description. When the user clicks on the link or the description, the canvas app opens up in the feed. If the user clicks the link again, the content is collapsed, giving users a seamless experience for working in their feed.

For example, you might have a canvas app that allows a user to log their hours worked. You can now programmatically create a feed item that displays a canvas app which shows the user their currently logged hours.

In addition, the feed item could display actions depending on the current user. So the canvas app could then post a feed item to the user's manager, and the manager could approve or deny the hours logged. Since the content is served from the canvas app, the developer has full control over the behavior.

#### See Also:

[Publisher Context Considerations](#)  
[Canvas Apps in the Publisher—Pilot](#)  
[Chatter Feed Context Considerations](#)

## Chatter Feed Context Considerations

When you display a canvas app inside of a feed item, the context information you receive from the signed request or from a `getContext()` call contains information specific to the feed:



**Note:** Support for Force.com Canvas apps in the publisher, the Chatter feed, and Salesforce1 is currently available through a pilot program and is available in all new Development Edition organizations. For information on enabling it for your organization, contact [salesforce.com](https://salesforce.com).

- **Location**—If the canvas app is in the feed, then the `Environment.displayLocation` field contains the value `ChatterFeed`.
- **Parameters**—When you create a feed item that contains a canvas app, you can specify a JSON string as the parameters value. When the canvas app receives the context, the parameters in the feed item will be contained in the `Environment.Parameters` object.
- **Size**—The `Environment.Dimensions` object contains information about the size of the canvas app.
  - ◇ The canvas app height defaults to 100 pixels.
  - ◇ The canvas app width defaults to 420 pixels, which is the same as the maximum width of a canvas app in the feed.
  - ◇ The maximum height of a canvas app in the feed is 400 pixels.
  - ◇ The maximum width of a canvas app in the feed is 420 pixels.
  - ◇ This code snippet shows the default size values of a canvas app in the feed:

```
"dimensions":
{
  "width": "420px",
  "height": "100px",
  "maxHeight": "400px",
  "maxWidth": "420px"
}
```

- ◇ The feed is a fixed width of 420 pixels. For example, if you resize your canvas app to be 200 pixels, the feed width remains 420 pixels.
- ◇ You can use the `resize()` method in the Force.com Canvas SDK to change the values of your canvas app up to the `maxHeight` and `maxWidth`.

### See Also:

[Canvas Apps in the Chatter Feed—Pilot Environment Dimensions](#)  
[Resizing a Canvas App](#)  
[Chatter Feed Canvas App Access Considerations](#)

## Chatter Feed Canvas App Access Considerations

When modifying a canvas app that appears in the feed, keep these considerations in mind:



**Note:** Support for Force.com Canvas apps in the publisher, the Chatter feed, and Salesforce1 is currently available through a pilot program and is available in all new Development Edition organizations. For information on enabling it for your organization, contact [salesforce.com](https://salesforce.com).



- If the canvas app is deleted and that app is in feed items, those feed items will remain. If a user accesses one of those feed items, they'll receive an error that the canvas app doesn't exist.
- If you remove a user's access to a canvas app and that app is in feed items, those feed items will remain. If a user accesses one of those feed items, they'll receive an error that they don't have permissions to access the canvas app.
- When creating a canvas app feed item either through a publisher action or through the Chatter API, Salesforce checks to see if the canvas app exists and if the user has permissions to it.
  - ◊ If the canvas app doesn't exist, the feed item can't be created and an error is returned.
  - ◊ If the canvas app exists, but the user attempting to create the feed item doesn't have access to the canvas app, the feed item is created. However, the user won't be able to view the feed item and an error is returned.
- If the canvas app attempts to perform an action for which the user doesn't have permissions, then that action will fail and the canvas app will receive an error. For example, if the app tries to create a Merchandise record but the user doesn't have create permission on Merchandise, then the app will receive an error. The canvas app should then relay the error to the user.

**See Also:**[\*Chatter Feed Context Considerations\*](#)[\*CanvasRequest\*](#)

## Chapter 8

### Canvas in the Salesforce1 App—Pilot

#### In this chapter ...

- [Salesforce1 Context Considerations](#)
- [Salesforce1 Access Considerations](#)



**Note:** Support for Force.com Canvas apps in the publisher, the Chatter feed, and Salesforce1 is currently available through a pilot program and is available in all new Development Edition organizations. For information on enabling it for your organization, contact [salesforce.com](mailto:salesforce.com).

Force.com Canvas enables you to expose your canvas apps in Salesforce1. The Salesforce1 app is Salesforce on the go. This enterprise-class mobile app gives you real-time access to the same information that you see in the office, but it's organized for getting work done when you're away from your desk. Just like in the full Salesforce site, users can access publisher and Chatter feed items, including Force.com Canvas apps.

Developers can use canvas apps in Salesforce1 to:

- Expose a canvas app in Salesforce1 publisher actions. An icon indicates a canvas app. You can use either the default puzzle icon or upload a custom icon for the related publisher action.
- Post to the Chatter feed from a canvas app in Salesforce1 or through the Chatter API.
- Display a canvas app inside a Chatter feed item from within Salesforce1. An icon indicates a canvas app. You can use either the default puzzle icon or provide a thumbnail URL in the feed item to display a custom icon.

For example, you might have a canvas app that warehouse employees use to process orders on a mobile device. You can create a publisher action that accesses the app from the publisher icon of the device, allowing employees to pull up a list of customer orders. After an order is processed, the app sets the order status in Salesforce and posts a feed item to the associated customer account.

Users can still access your canvas app from within Salesforce on a desktop machine. The additional functionality for mobile devices that Salesforce1 offers doesn't impact or limit existing functionality.

#### See Also:

[Canvas Apps in the Publisher—Pilot](#)  
[Canvas Apps in the Chatter Feed—Pilot](#)  
[Salesforce1 Access Considerations](#)  
[Salesforce1 Context Considerations](#)

## Salesforce1 Context Considerations

When you display a canvas app inside of Salesforce1, keep these considerations in mind:



**Note:** Support for Force.com Canvas apps in the publisher, the Chatter feed, and Salesforce1 is currently available through a pilot program and is available in all new Development Edition organizations. For information on enabling it for your organization, contact salesforce.com.

When you display a canvas app inside of the feed or publisher, the canvas context you receive (either from the Signed Request or from the `getContext` call) contains information specific to the Salesforce1 publisher.

- You can verify you're on either the feed or publisher by looking at the `displayLocation` value in the environment section. For publisher, `displayLocation` is set to **Publisher**. For the feed, `displayLocation` is set to **ChatterFeed**.
- When creating a Canvas feed item, you can specify a JSON string as the parameter's value. When the context is sent, any value in the parameter's field on the feed item is sent in the parameters of the environment section of the context.
- As with any canvas app, the context contains information about the app's dimensions. Since Salesforce1 is designed for mobile, the sizes we provide are different than for the full Salesforce site.
- To utilize a single finger touch scrolling experience:
  - ◇ Ensure that the outermost `div` elements are contain the following properties:
    - `min-height: 250px;`
    - `overflow: scroll;`
    - `width: 100%;`
    - `-webkit-overflow-scrolling: touch;`
    - `-webkit-transform: translated(0%,0px,0px);`
  - ◇ Set the `height` attribute to the `clientHeight` value delivered in the signed request. For example:

```
var h = parseInt(sr.context.environment.dimensions.clientHeight, 10); // Where sr is
a parsed signed request object.
Sfdc.canvas.byId('divElementId').style.height = h;
```

- ◇ The `clientHeight` value can be very small, particularly in a phone's landscape mode, and users may not be able to see any content. Use `min-height` set to the desired height in pixels to ensure a good user experience.

### See Also:

[Canvas in the Salesforce1 App—Pilot](#)

[Salesforce1 Access Considerations](#)

[Resizing a Canvas App](#)


## Salesforce1 Access Considerations

When modifying a canvas app that appears in Salesforce1, keep these considerations in mind:



**Note:** Support for Force.com Canvas apps in the publisher, the Chatter feed, and Salesforce1 is currently available through a pilot program and is available in all new Development Edition organizations. For information on enabling it for your organization, contact salesforce.com.

By necessity, the Salesforce1 layout is different than the full Salesforce site layout you might be used to. Remember to keep the following in mind when creating a canvas app for use in Salesforce1.

- Since canvas apps are designed to display your third-party application inside of Salesforce, the device must have access to your canvas app URL. If your app is only accessible behind a firewall, the mobile device must be behind the firewall, too. If users don't have access to the canvas URL, they receive an error—possibly a 404 or 500 error.
- When in the Salesforce1 publisher, your canvas app scrolls up and behind the “What are you working on?” text box.
- The canvas app link and description in the feed might display fewer characters than what is displayed in the full Salesforce site.
- Depending on the device you use, the feed screen can change if the device is rotated. Your canvas app should support rotation if possible.
- The heights used in Salesforce1 and the full Salesforce site are different. Use the dimensions object in the signed request to render your publisher actions correctly.
- In the publisher, long publisher action labels might be truncated.
- The feed layout is different than that of the full Salesforce site. Instead of an app opening in the feed, a page opens that displays the canvas app on the entire screen. To return to Salesforce1, tap .
- When you view the publisher or feed in Salesforce1, the default Force.com Canvas puzzle icon displays for canvas apps. You can override this default icon with an image you provide.

**See Also:**

[Canvas in the Salesforce1 App—Pilot](#)  
[Salesforce1 Context Considerations](#)

Force.com Canvas provides an SDK that contains objects that enable communication between your canvas app and Salesforce.

#### See Also:

[CanvasRequest](#)

[SignedRequest](#)

## CanvasRequest

When you use a signed request for authentication in your canvas app, you get a `CanvasRequest` object back in the response. This object contains fields related to the request, and also contains the [Context](#) and [Client](#) objects. The `CanvasRequest` object is returned in JSON format and contains the following fields.

Field	Description
<code>algorithm</code>	The algorithm used to sign the request.
<code>issuedAt</code>	The timestamp when the <code>oAuthToken</code> was issued.
<code>userId</code>	The context user's ID.

The following code snippet shows an example of the `CanvasRequest` object.

```
{
  "context":
  {
    "application":
    {
      "applicationId": "06Px000000003ed",
      "authType": "SIGNED_REQUEST",
      "canvasUrl": "http://instance.salesforce.com:8080
        /canvas_app_path/canvas_app.jsp",
      "developerName": "my_java_app",
      "name": "My Java App",
      "namespace": "org_namespace",
      "referenceId": "09HD00000000AUM",
      "version": "1.0.0"
    },
    "user":
    {
      "accessibilityModeEnabled": false,
      "currencyISOCode": "USD",
      "email": "admin@6457617734813492.com",

```

```

    "firstName": "Sean",
    "fullName": "Sean Forbes",
    "isDefaultNetwork": false,
    "language": "en_US",
    "lastName": "Forbes",
    "locale": "en_US",
    "networkId": "0DBxx0000000001r",
    "profileId": "00ex0000000jzpt",
    "profilePhotoUrl": "/profilephoto/005/F",
    "profileThumbnailUrl": "/profilephoto/005/T",
    "roleId": null,
    "siteUrl": "https://mydomain.force.com/",
    "siteUrlPrefix": "/mycommunity",
    "timeZone": "America/Los_Angeles",
    "userId": "005x0000001SyEAAS",
    "userName": "admin@6457617734813492.com",
    "userType": "STANDARD"
  },
  "environment":
  {
    "parameters":
    {
      "complex":
      {
        "key1": "value1",
        "key2": "value2"
      },
      "integer": 10,
      "simple": "This is a simple string.",
      "boolean": true
    },
    "dimensions":
    {
      "clientHeight": "50px",
      "clientWidth": "70px",
      "height": "900px",
      "width": "800px",
      "maxWidth": "1000px",
      "maxHeight": "2000px"
    },
    "displayLocation": "Chatter",
    "locationUrl": "http://www.salesforce.com/some/path/index.html",
    "uiTheme": "Theme3",
    "version":
    {
      "api": "29.0",
      "season": "WINTER"
    },
  },
  "organization":
  {
    "currencyIsoCode": "USD",
    "multicurrencyEnabled": true,
    "name": "Edge Communications",
    "namespacePrefix": "org_namespace",
    "organizationId": "00Dx00000001hxyEAA"
  },
  "links":
  {
    "chatterFeedItemsUrl": "/services/data/v29.0/chatter/feed-items",
    "chatterFeedsUrl": "/services/data/v29.0/chatter/feeds",
    "chatterGroupsUrl": "/services/data/v29.0/chatter/groups",
    "chatterUsersUrl": "/services/data/v29.0/chatter/users",
    "enterpriseUrl": "/services/Soap/c/29.0/00Dx00000001hxy",
    "loginUrl": "http://login.salesforce.com",
    "metadataUrl": "/services/Soap/m/29.0/00Dx00000001hxy",
    "partnerUrl": "/services/Soap/u/29.0/00Dx00000001hxy",
    "queryUrl": "/services/data/v29.0/query/",
    "recentItemsUrl": "/services/data/v29.0/recent/",
    "restUrl": "/services/data/v29.0/",
  }
}

```

```
        "searchUrl":"/services/data/v29.0/search/",
        "sobjectUrl":"/services/data/v29.0/sobjects/",
        "userUrl":"/005x0000001SyyEAAS"
      }
    },
    "client":
    {
      "instanceId":"06Px000000002JZ",
      "instanceUrl":"http://instance.salesforce.com:8080",
      "oauthToken":"00Dx0000X00Or4J!ARQAQJ34YL8gKowP65p8FDHkvk.Uq5...",
      "targetOrigin":"http://instance.salesforce.com:8080"
    },
    "algorithm":"HMACSHA256",
    "userId":"005x0000001SyyEAAS",
    "issuedAt":null
  }
}
```

See Also:

- [Objects](#)
- [Client](#)
- [Context](#)

Client

The Client object is a JSON-formatted object returned by the signed request in the [CanvasRequest](#) object. It contains context information about the client app.

Field	Description
instanceId	The ID of a canvas app exposed on a Visualforce page. Used internally for canvas app instance management.
instanceUrl	The URL of the Salesforce instance. For example, <code>http://instance.salesforce.com</code> . This is used to preface the URLs returned by the <a href="#">Links</a> object.
oauthToken	The OAuth access token that's returned to the caller.
targetOrigin	The URL of the canvas app. Used internally for cross-domain communication between the page and the iFrame that contains the canvas app.

The following code snippet shows an example of the Client object.

```
"client":
{
  "instanceId":"06Px000000002JZ",
  "instanceUrl":"http://instance.salesforce.com:8080",
  "oauthToken":"00Dx0000X00Or4J!ARQAQJ34YLUq54RMTrQbxVoWUgorjFi3",
  "targetOrigin":"http://instance.salesforce.com:8080"
}
```

See Also:

- [CanvasRequest](#)

## Context

When you add your canvas app as a connected app in Salesforce, you can retrieve information about the current environment by using the Context object. The Context object provides information to your app about how and by whom it's being consumed. You can use this information to make subsequent calls for information and code your app so that it appears completely integrated with the Salesforce user interface. This object is returned in JSON format and contains the following objects:

- **Application**—Information about the canvas app, such as version, access method, URL, and so on.
- **Environment**—Information about the environment, such as location, UI theme, and so on.
- **Links**—Links, such as the metadata URL, user URL, Chatter groups URL, and so on. You can use these links to make calls into Salesforce from your app.
- **Organization**—Information about the organization, such as name, ID, currency code, and so on.
- **User**—Information about the currently logged-in user, such as locale, name, user ID, email, and so on.

The following code is an example of the Context object. The Context object is returned in the CanvasRequest object when you authenticate using signed request. You can also explicitly make a call to get this object.

```
"context":{
  "application":
  {
    "applicationId": "06Px000000003ed",
    "authType":"SIGNED_REQUEST",
    "canvasUrl":"http://instance.salesforce.com:8080
      /canvas_app_path/canvas_app.jsp",
    "developerName":"my_java_app",
    "name":"My Java App",
    "namespace":"org_namespace",
    "referenceId":"09HD00000000AUM",
    "version":"1.0.0"
  },
  "user":
  {
    "accessibilityModeEnabled":false,
    "currencyISOCode":"USD",
    "email":"admin@6457617734813492.com",
    "firstName":"Sean",
    "fullName":"Sean Forbes",
    "isDefaultNetwork":false,
    "language":"en_US",
    "lastName":"Forbes",
    "locale":"en_US",
    "networkId":"0DBxx0000000001r",
    "profileId":"00ex0000000jzpt",
    "profilePhotoUrl":"/profilephoto/005/F",
    "profileThumbnailUrl":"/profilephoto/005/T",
    "roleId":null,
    "siteUrl":"https://mydomain.force.com/",
    "siteUrlPrefix":"/mycommunity",
    "timeZone":"America/Los_Angeles",
    "userId":"005x0000001SyxEAAS",
    "userName":"admin@6457617734813492.com",
    "userType":"STANDARD"
  },
  "environment":
  {
    "parameters":{},
    "dimensions":
    {
      "clientHeight": "50px",
      "clientWidth": "70px",
      "width":"800px",
      "height":"900px"
      "maxWidth":"1000px",
      "maxHeight":"2000px"
    }
  }
}
```



```

    },
    "displayLocation": "Chatter",
    "locationUrl": "http://instance.salesforce.com:8080/canvas_app_path",
    "uiTheme": "Theme3",
    "version":
    {
        "api": "29.0",
        "season": "SUMMER"
    }
  },
  "organization":
  {
    "currencyIsoCode": "USD",
    "multicurrencyEnabled": true,
    "name": "Edge Communications",
    "namespacePrefix": "org_namespace",
    "organizationId": "00Dx00000001hxyEAA"
  },
  "links":
  {
    "chatterFeedItemsUrl": "/services/data/v29.0/chatter/feed-items",
    "chatterFeedsUrl": "/services/data/v29.0/chatter/feeds",
    "chatterGroupsUrl": "/services/data/v29.0/chatter/groups",
    "chatterUsersUrl": "/services/data/v29.0/chatter/users",
    "enterpriseUrl": "/services/Soap/c/29.0/00Dx00000001hxy",
    "loginUrl": "http://login.salesforce.com",
    "metadataUrl": "/services/Soap/m/29.0/00Dx00000001hxy",
    "partnerUrl": "/services/Soap/u/29.0/00Dx00000001hxy",
    "queryUrl": "/services/data/v29.0/query/",
    "restUrl": "/services/data/v29.0/",
    "recentItemsUrl": "/services/data/v29.0/recent/",
    "searchUrl": "/services/data/v29.0/search/",
    "subjectUrl": "/services/data/v29.0/subjects/",
    "userUrl": "/005x0000001SyyEAAS"
  }
}

```

## See Also:

[Client](#)

[Getting Context in Your Canvas App](#)

## Application

The Application object is a JSON-formatted object returned by the signed request in the [Context](#) object. The Application object contains specific canvas app information obtained after you've added your app as a connected app in Salesforce from Setup by clicking **Create > Apps > Connected Apps**.

Field	Description
applicationId	The ID of the canvas app.
authType	The access method of the canvas app. You specify this value when you expose the canvas app by creating a connected app.
canvasUrl	The URL of the canvas app, for example: <code>http://instance.salesforce.com:8080/canvas_app_path/canvas_app.jsp</code> .
developerName	The internal name of the canvas app. You specify this value in the API Name field when you expose the canvas app by creating a connected app.
name	The name of the canvas app.
namespace	The Salesforce namespace prefix associated with the canvas app.

Field	Description
referenceId	The unique ID of the canvas app definition. Used internally.
version	The version of the canvas app. This value changes after you update and re-publish a canvas app in an organization.

The following code snippet shows an example of the Application object.

```
"application":
{
  "applicationId": "06Px000000003ed",
  "authType": "SIGNED_REQUEST",
  "canvasUrl": "http://instance.salesforce.com:8080/canvas_app_path/canvas_app.jsp",
  "developerName": "my_java_app",
  "name": "My Java App",
  "namespace": "org_namespace",
  "referenceId": "09HD00000000AUM",
  "version": "1.0.0"}
```

## See Also:

[Context](#)

## Environment

The Environment object is a JSON-formatted object containing context information about the canvas app environment. This object contains the [Dimensions](#) object, the [Version](#) object, and parameters (if there are any) in the [Parameters](#) on page 64 object.

Field	Description
displayLocation	The location in the application where the canvas app is currently being called from. Valid values are: <ul style="list-style-type: none"> <li>Chatter—The canvas app was called from the Chatter tab.</li> <li>ChatterFeed—The canvas app was called from a Chatter canvas feed item.</li> <li>OpenCTI—The canvas app was called from an Open CTI component.</li> <li>Publisher—The canvas app was called from a canvas custom publisher action.</li> <li>ServiceDesk—The canvas app was called from a Salesforce Console component.</li> <li>Visualforce—The canvas app was called from a Visualforce page.</li> <li>None—The canvas app was called from the Canvas App Previewer.</li> </ul>
locationUrl	The URL of the page in Salesforce where the user accesses the canvas app. For example, if users access your canvas app by clicking a link on the Chatter tab, this field contains the URL of the Chatter tab.
uiTheme	The default theme for the context organization.

The following code snippet shows an example of the Environment object.

```
"environment":
{
```

```

    "parameters": {},
    "dimensions": {
      "height": "900px",
      "width": "800px",
      "maxHeight": "2000px",
      "maxWidth": "1000px"
    },
    "displayLocation": "Chatter",
    "locationUrl": "http://www.salesforce.com/some/path/index.html",
    "uiTheme": "Theme3",
    "version": {
      "api": "29.0",
      "season": "WINTER"
    }
  }
}

```

### See Also:

[Context](#)

[Dimensions](#)

[Parameters](#)

[Version](#)

### Dimensions

The Dimensions object is a JSON-formatted object containing context information about dimensions of the iFrame on which the canvas app appears.

Field	Description
<code>clientHeight</code>	The values of <code>clientHeight</code> are sent to the canvas app within SignedRequest and can be used to set the height of the outermost <code>div</code> element. This field is useful for setting the height of an app that's accessed from a mobile device. The field can be referenced as <code>sr.context.environment.dimensions.clientHeight</code> , where <code>sr</code> is the parsed signed request JavaScript object.
<code>clientWidth</code>	The values of <code>clientWidth</code> are sent to the canvas app within SignedRequest and can be used to set the width of the outermost <code>div</code> element. This field is useful for setting the width of an app that's accessed from a mobile device. The field can be referenced as <code>sr.context.environment.dimensions.clientWidth</code> , where <code>sr</code> is the parsed signed request JavaScript object.
<code>height</code>	The height of the iFrame in pixels.
<code>width</code>	The width of the iFrame in pixels.
<code>maxHeight</code>	The maximum height of the iFrame in pixels. Defaults to 2,000; "infinite" is also a valid value.
<code>maxWidth</code>	The maximum width of the iFrame in pixels. Defaults to 1,000; "infinite" is also a valid value.

The following code snippet shows an example of the Dimensions object.

```
"dimensions":
{
    "clientHeight": "50px",
    "clientWidth": "70px",
    "height": "900px",
    "width": "800px",
    "maxHeight": "2000px",
    "maxWidth": "1000px"
},
```

### See Also:

[Environment](#)

[User Interface Considerations](#)

### Parameters

The Parameters object is a JSON-formatted object containing context information specified by the developer. The Parameters object is specified in a Visualforce page by using the "parameters" tag in the `apex:canvasApp` component, or in a Chatter canvas feed item by using the "parameters" variable when creating the feed item.

The following code snippet shows an example of the Parameters object.

```
"parameters":
{
    "inventoryStatus": "Picked",
    "stockPrice": "12.99",
    "count": 2,
    "address": { "street": "1 Market", "city": "San Francisco", "state": "CA" }
}
```

### See Also:

[Environment](#)

[apex:canvasApp Component](#)

### Version

The Version object is a JSON-formatted object containing context information about the version of Salesforce running on the server.

Field	Description
api	The API version of the server.
season	The season of the release running on the server.

The following code snippet shows an example of the Version object.

```
"version":
{
    "api": "29.0",
```

```
    "season": "WINTER"
  }
```

## See Also:

[Environment](#)

## Links

The Links object is a JSON-formatted object containing URLs that can be helpful when integrating your canvas app. For example, use the `enterpriseUrl` to make calls into Salesforce using the enterprise WSDL.

Field	Description
<code>chatterFeedItemsUrl</code>	URL to return Chatter feed items for the context organization.
<code>chatterFeedsUrl</code>	URL to return Chatter feeds for the context organization.
<code>chatterGroupsUrl</code>	URL to return Chatter groups for the context organization.
<code>chatterUsersUrl</code>	URL that returns Chatter users for the context organization.
<code>enterpriseUrl</code>	URL to make API calls back to Salesforce using the enterprise WSDL.
<code>loginUrl</code>	URL of the login server for the instance the context user is logged into. Used internally.
<code>metadataUrl</code>	URL to make calls to the Metadata API.
<code>partnerUrl</code>	URL to make API calls back to Salesforce using the partner WSDL.
<code>queryUrl</code>	URL to issue a SOQL query.
<code>recentItemsUrl</code>	URL to access recent items.
<code>restUrl</code>	URL to return a list of REST API resources.
<code>searchUrl</code>	URL to issue a SOSL search.
<code>sobjectUrl</code>	URL that retrieves metadata or data from Salesforce objects.
<code>userUrl</code>	URL for the context user.

The following code snippet shows an example of the Links object.

```
"links":
{
  "chatterFeedItemsUrl": "/services/data/v29.0/chatter/feed-items",
  "chatterFeedsUrl": "/services/data/v29.0/chatter/feeds",
  "chatterGroupsUrl": "/services/data/v29.0/chatter/groups",
  "chatterUsersUrl": "/services/data/v29.0/chatter/users",
  "enterpriseUrl": "/services/Soap/c/29.0/00Dx00000001hxy",
  "loginUrl": "http://login.salesforce.com",
  "metadataUrl": "/services/Soap/m/29.0/00Dx00000001hxy",
  "partnerUrl": "/services/Soap/u/29.0/00Dx00000001hxy",
  "queryUrl": "/services/data/v29.0/query/",
  "recentItemsUrl": "/services/data/v29.0/recent/",
  "restUrl": "/services/data/v29.0/",
  "searchUrl": "/services/data/v29.0/search/",
  "sobjectUrl": "/services/data/v29.0/sobjects/",
  "userUrl": "/005x00000001SyyEAAS"
}
```

## Available Links and OAuth Scopes

Links that the canvas app can use depend on what OAuth scopes are selected when you create the connected app.

Scope	Description
Access your basic information	<ul style="list-style-type: none"> <li>• <code>userUrl</code></li> <li>• <code>loginUrl</code></li> </ul>
Access and manage your data	<ul style="list-style-type: none"> <li>• <code>enterpriseUrl</code></li> <li>• <code>metadataUrl</code></li> <li>• <code>partnerUrl</code></li> <li>• <code>queryUrl</code></li> <li>• <code>recentItemsUrl</code></li> <li>• <code>restUrl</code></li> <li>• <code>searchUrl</code></li> <li>• <code>sobjectUrl</code></li> </ul>
Access and manage your chatter feed	<ul style="list-style-type: none"> <li>• <code>chatterFeedItemsUrl</code></li> <li>• <code>chatterFeedsUrl</code></li> <li>• <code>chatterGroupsUrl</code></li> <li>• <code>chatterUsersUrl</code></li> </ul>
Full access	All links

## See Also:

[Context](#)

## Organization

The Organization object is a JSON-formatted object containing context information about the organization in which the canvas app is running.

Field	Description
<code>currencyIsoCode</code>	If multi-currency is enabled for the context organization, then the context user's currency is returned. Defaults to the corporate currency if not set. If multi-currency is disabled for the context organization, the currency for the corporate currency locale is returned.
<code>multicurrencyEnabled</code>	Indicates whether the context organization uses multiple currencies ( <code>true</code> ) or does not ( <code>false</code> ).
<code>namespacePrefix</code>	The Salesforce namespace prefix associated with the canvas app.
<code>name</code>	The name of the context organization.
<code>organizationId</code>	The ID of the context organization.

The following code snippet shows an example of the Organization object.

```
"organization":
{
```

```

    "currencyIsoCode": "USD",
    "multicurrencyEnabled": true,
    "name": "Edge Communications",
    "namespacePrefix": "org_namespace",
    "organizationId": "00Dx00000001hxyEAA"
  }

```

## See Also:

[Context](#)

## User

The User object is a JSON-formatted object containing context information about the current user. This information can be used within the canvas app (for example, to display the user's name) or to make subsequent calls to retrieve additional information (for example, to get role information for the current user).

Field	Description
accessibilityModeEnabled	For the context user, indicates whether user interface modifications for the visually impaired are on ( <code>true</code> ) or off ( <code>false</code> ).
currencyISOCode	Returns the context user's default currency code for multiple currency organizations or the organization's currency code for single currency organizations.
email	The context user's email address.
firstName	The context user's first name.
fullName	The context user's full name. The format is: Firstname Lastname.
isDefaultNetwork	For the context user, indicates whether the user is in a community ( <code>true</code> ) or not ( <code>false</code> ).
language	The context user's language. String is 2-5 characters long. The first two characters are always an ISO language code, for example "fr" or "en." If the value is further qualified by country, then the string also has an underscore ( <code>_</code> ) and another ISO country code, for example "US" or "UK." For example, the string for the United States is "en_US," and the string for French Canadian is "fr_CA."
lastName	The context user's last name.
locale	The context user's locale.
networkId	The ID of the context user's currently logged-in community.
profileId	The ID of the profile associated with the role currently assigned to the context user.
profilePhotoUrl	The URL for the photo of the context user's profile photo if Chatter is enabled. The URL is updated every time a photo is uploaded and reflects the most recent photo. URL returned for an older photo is not guaranteed to return a photo if a newer photo has been uploaded. Always query this field for the URL of the most recent photo.
profileThumbnailUrl	The URL for a thumbnail of the context user's profile photo if Chatter is enabled. The URL is updated every time a photo is uploaded and reflects the most recent photo. URL returned for an older photo is not

Field	Description
	guaranteed to return a photo if a newer photo has been uploaded. Always query this field for the URL of the most recent photo.
roleId	The ID of the context user's currently assigned role.
siteUrl	The URL of the context user's currently logged-in community.
siteUrlPrefix	The URL prefix of the context user's currently logged-in community.
timeZone	The context user's time zone.
userId	The context user's ID.
userName	The context user's login name.
userType	The type of user license assigned to the profile associated with the context user.

The following code snippet shows an example of the User object.

```
"user":
{
  "accessibilityModeEnabled":false,
  "currencyISOCODE":"USD",
  "email":"admin@6457617734813492.com",
  "firstName":"Sean",
  "fullName":"Sean Forbes",
  "isDefaultNetwork":false,
  "language":"en_US",
  "lastName":"Forbes",
  "locale":"en_US",
  "networkId":"0DBxx000000001r",
  "profileId":"00ex0000000jzpt",
  "profilePhotoUrl":"/profilephoto/005/F",
  "profileThumbnailUrl":"/profilephoto/005/T",
  "roleId":null,
  "siteUrl":"https://mydomain.force.com/",
  "siteUrlPrefix":"/mycommunity",
  "timeZone":"America/Los_Angeles",
  "userId":"005x0000001SyxEAAS",
  "userName":"admin@6457617734813492.com",
  "userType":"STANDARD"
}
```

## See Also:

[Context](#)

## SignedRequest

Force.com Canvas provides a SignedRequest object that has methods you can use to work with signed requests.

Method	Signature	Description
verifyAndDecode	public static CanvasRequest verifyAndDecode(String input, String secret) throws SecurityException	Used to verify and decode a signed request; returns a <a href="#">CanvasRequest</a> Java object.



Method	Signature	Description
verifyAndDecodeAsJson	public static String verifyAndDecodeAsJson(String input, String secret) throws SecurityException	Used to verify and decode a signed request; returns the <a href="#">CanvasRequest</a> data in a JSON string.

**See Also:**[CanvasRequest](#)[Verifying and Decoding a Signed Request](#)

# Chapter 10

## Force.com Canvas Limits

---

Because Force.com Canvas runs in a multitenant environment, limits are enforced to ensure protection of shared resources.

Description	Limit
Number of canvas apps per user that can be displayed on the Chatter tab. Only the first 50 canvas apps are displayed (sorted alphabetically).	50
Number of Force.com Canvas calls per day per user (24-hour period)	5,000  This includes SDK calls to get context and signed request calls. Note that when you call a SignedRequest method, there are actually two calls that are made—one call for the method and one call for internal logging.
Heroku Quick Start calls per day per user	100  Heroku accounts have their own limits on the number of calls you can make.

### See Also:

[Introducing Force.com Canvas](#)

[Quick Start](#)

[Quick Start—Advanced](#)

# Index

## A

- apex
  - canvasApp component [44](#)
- Application object [61](#)
- Authentication
  - OAuth [29](#)
  - signed request [23](#)

## B

- Browsers supported [3](#)

## C

- Canvas app process
  - overview [5](#)
- Canvas app user flow
  - OAuth [29](#)
  - signed request [24](#)
- Canvas apps
  - events [36](#)
  - in Chatter feed [51](#)
  - in the publisher [46](#)
  - listening for Streaming API events [39](#)
  - resizing [34](#)
  - Visualforce pages [42](#)
  - where they appear in the user interface [2](#)
- Canvas apps in the publisher
  - create the action manually [47](#)
  - set the location and create the action [47](#)
- CanvasRequest
  - object [57](#)
- CanvasRequest object [57](#)
- Chatter feed
  - canvas apps in [51](#)
- Chatter Feed
  - canvas app access considerations when exposing a canvas app in the feed [52](#)
  - context considerations when exposing a canvas app in the feed [52](#)
- Client object [59](#)
- Code examples
  - events, creating an event [37](#)
  - events, subscribing to an event [38](#)
  - events, unsubscribing from an event [38](#)
  - exposing canvas app on a Visualforce page [43](#)
  - getting context [32](#)
  - OAuth flow [31](#)
  - resizing, automatically resizing a canvas app by calling autogrow [34](#)
  - resizing, explicitly resizing a canvas app by calling resize [34](#)
  - resizing, getting the size of a canvas app by calling size [35](#)
  - resizing, subscribing to parent events by calling subscribe [35](#)
  - Streaming API, creating a Streaming API event [39](#)

## Code examples (*continued*)

- Streaming API, subscribing to a Streaming API event by calling subscribe [40](#)
- Streaming API, unsubscribing from a Streaming API event by calling unsubscribe [41](#)
- verifying and decoding a signed request [26](#)
- XHR, getting a list of Chatter users [33](#)
- XHR, posting to a Chatter feed [33](#)

## Context

- Application object [61](#)
- Client object [59](#)
- Dimensions object [63](#)
- Environment object [62](#)
- getting from canvas app [32](#)
- Links object [65](#)
- object [60](#)
- Organization object [66](#)
- Parameters object [64](#)
- User object [67](#)
- Version object [64](#)

## Context object [60](#)

## Cross-domain XHR [32](#)

## D

- Dimensions object [63](#)

## E

- Editions supported [3](#)
- Environment object [62](#)
- Events
  - considerations [36](#)
  - creating, code example [37](#)
  - subscribing to, code examples [38](#)
  - unsubscribing from, code examples [38](#)
- Events code examples
  - creating an event [37](#)
  - subscribing to an event [38](#)
  - unsubscribing from an event [38](#)

## F

- Force.com Canvas scenarios [2](#)
- Force.com Canvas SDK
  - hosted on Salesforce server [23](#)
  - publisher events [48](#)

## I

- Introduction to Force.com Canvas [1](#)

## L

Limits [70](#)  
 Links object [65](#)

## O

OAuth  
   initiating [31](#)  
 OAuth authentication [29](#)  
 Organization object [66](#)

## P

Parameters object [64](#)  
 Process for creating a canvas app [5](#)  
 Publisher  
   canvas app access considerations when exposing a canvas app in the publisher [50](#)  
   canvas apps in [46](#)  
   context considerations when exposing a canvas app in the publisher [49](#)  
 Publisher events [48](#)

## Q

Quick start  
   create the app [8](#)  
   introduction [7](#)  
   prerequisites [8](#)  
   set the app location [9](#)  
 Quick start—advanced  
   clone the “hello world” project [12](#)  
   configure who can access the canvas app [16](#)  
   configure who can access the installed canvas app [20](#)  
   create the canvas app [14](#)  
   deploy the “hello world” Web app to Heroku [16](#)  
   install the canvas app [20](#)  
   introduction [11](#)  
   package the canvas app [19](#)  
   prerequisites [12](#)  
   run the Web app locally [13](#)  
   update the canvas app [18](#)  
   upload the canvas app package [19](#)

## R

Resizing [34](#)  
 Resizing code examples  
   automatically resizing a canvas app by calling autogrow [34](#)  
   explicitly resizing a canvas app by calling resize [34](#)  
   getting the size of a canvas app by calling size [35](#)  
   subscribing to parent events by calling subscribe [35](#)

## S

Salesforce1  
   access considerations when exposing a canvas app in Salesforce1 [55](#)  
   context considerations when exposing a canvas app in Salesforce1 [55](#)  
 Salesforce1 feed  
   canvas apps in [54](#)  
 SDK Objects [57](#)  
 sfdc.streamingapi event [39](#)  
 Signed request  
   verifying and decoding [26](#)  
 Signed request authentication [23](#)  
 SignedRequest  
   object [68](#)  
 SignedRequest object [68](#)  
 Streaming API [39](#)  
 Streaming API code examples  
   subscribing to a Streaming API event by calling subscribe [40](#)  
   unsubscribing from a Streaming API event by calling unsubscribe [41](#)  
 Streaming API event [39](#)  
 Supported browsers [3](#)  
 Supported Salesforce Editions [3](#)

## U

User interface considerations [4](#)  
 User object [67](#)  
 User permissions required [4](#)

## V

Version object [64](#)  
 Visualforce  
   apex:canvasApp component [44](#)  
   canvas app code examples [43](#)  
   canvas apps, exposing [42](#)  
   considerations when exposing a canvas app [44](#)

## W

Where canvas apps appear [2](#)

## X

XHR [32](#)  
 XHR code example  
   getting a list of Chatter users [33](#)  
   posting to a Chatter feed [33](#)