Canvas Developer Guide

Version 66.0, Spring ’26

Last updated: March 20, 2026

© Copyright 2000–2026 Salesforce, Inc. All rights reserved. Salesforce is a registered trademark of Salesforce, Inc., as are other names and marks. Other marks appearing herein may be trademarks of their respective owners.

CONTENTS

GETTING STARTED 1

Chapter 1: Introducing Canvas 1

Canvas Scenarios 3
Where Canvas Apps Appear 3
Supported Browsers 4
Supported Salesforce Editions 4
User Permissions Required 5
User Interface Considerations 5
Canvas App Process 7
Canvas Personal Apps 8
- Canvas Personal App Process 9
- Enabling Canvas Personal Apps within an Organization 9
- Making an App a Canvas Personal App 10
- Uninstalling a Canvas Personal App 11

Chapter 2: Quick Start 12

Prerequisites 13
Create the App 13
Set the Canvas App Location for External Client Apps 14
Set the Canvas App Location for Connected Apps 15

Chapter 3: Quick Start—Advanced 17

Prerequisites 18
Clone the Project from GitHub 18
Run the Web App Locally 19
Create the Canvas App 20
- Use an External Client App 21
- Use a Connected App 23

Configure Who Can Access the Canvas App 25
- Configure External Client App Access 25
- Configure Connected App Access 27

Deploy the Web App to Heroku 27
Deploy the Canvas External Client App to an Org 29
Update the Canvas App 30
Package the Canvas App 31
- Package the External Client App for Canvas 31
- Package the Connected App for Canvas 32
- Upload the Canvas Connected App Package 32
- Install the Canvas App 33

Configure Who Can Access the Installed Canvas App . . . 34

USING CANVAS . . . 35

Chapter 4: Canvas SDK . . . 35

Referencing the Canvas SDK . . . 36
Authentication . . . 36
Signed Request Authentication . . . 37
OAuth 2.0 Authorization . . . 44
SAML Single Sign-On for Canvas Apps . . . 48
Getting Context in Your Canvas App . . . 48
Cross-Domain XHR . . . 49
Getting a List of Chatter Users . . . 50
Posting to a Chatter Feed . . . 50
Alternatives to Cookies for User Tracking . . . 51
Resizing a Canvas App . . . 52
Automatically Resizing a Canvas App . . . 53
Explicitly Resizing a Canvas App . . . 53
Getting the Size of a Canvas App . . . 54
Subscribing to Parent Events . . . 54
Handling Orientation Changes in Your Canvas App . . . 55
Implementing Canvas App Events . . . 56
Canvas App Events Considerations . . . 56
Creating a Canvas App Event . . . 57
Subscribing to a Canvas App Event . . . 58
Unsubscribing from a Canvas App Event . . . 59
Using Streaming API in a Canvas App . . . 59
Using the Streaming API Event . . . 60
Subscribing to a Streaming API Event . . . 60
Unsubscribing from a Streaming API Event . . . 61
Debugging in a Canvas App . . . 61
Using the <select> Tag in a Canvas App Tag in a Canvas App . . . 62

Chapter 5: Canvas Apps and Visualforce Pages . . . 63

Visualforce Page Code Examples . . . 64
Visualforce Considerations . . . 65
apex:canvasApp Component . . . 65
Returning Fields in the Record Object . . . 68
Using Events between a Visualforce Page and a Canvas App . . . 69
Publishing a Canvas Event from a Visualforce Page . . . 70
Resizing a Canvas App in a Visualforce Page . . . 71
Subscribing to Events . . . 71
Unsubscribing from Events in a Visualforce Page . . . 72

Chapter 6: Lightning Component Code Examples . . . 73

Chapter 7: Canvas Apps in a Page Layout or a Mobile Card 75
Where Canvas Apps Appear in a Page Layout 76
Add a Canvas App to a Page Layout 76

Chapter 8: Canvas Apps in the Publisher 78
Set Canvas App Location and Create the Action 79
Create the Action Manually 79
Canvas SDK Publisher Events 80
Publisher Context Considerations 81
Publisher Canvas App Access Considerations 82

Chapter 9: Canvas Apps in the Chatter Feed 83
Chatter Feed Context Considerations 84
Chatter Feed Canvas App Access Considerations 84

Chapter 10: Canvas in the Salesforce Mobile App 86
Set Canvas App Location and Add it to the Navigation Menu 87
Salesforce Mobile App Context Considerations 87
Salesforce Mobile App Access Considerations 88
Salesforce Mobile App Custom Icons 89
Salesforce Mobile App Navigation Methods for Use with Canvas Apps 89

Chapter 11: Customizing Your App Lifecycle 93
Creating a CanvasLifecycleHandler 94
Associating Your CanvasLifecycleHandler with Your App 94
Filtering CanvasRequest Context Data 95
Controlling Canvas App Behavior 95
Presenting User Error Messages 96
Testing Your CanvasLifecycleHandler Implementation 97
Distributing Your CanvasLifecycleHandler Class 98

REFERENCE 100

Chapter 12: Objects 100
CanvasRequest 100
Client 103
Context 104
Controller 118
SignedRequest 119

Chapter 13: Canvas Limits 120

INDEX 121

GETTING STARTED

CHAPTER 1 Introducing Canvas

In this chapter ...

- Canvas Scenarios
- Where Canvas Apps Appear
- Supported Browsers
- Supported Salesforce Editions
- User Permissions Required
- User Interface Considerations
- Canvas App Process
- Canvas Personal Apps

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

Important: To prepare your Canvas apps for the new web browser restrictions on third-party cookies, see How Third-Party Cookie Restrictions Impact Salesforce Canvas Apps in Salesforce Help.

Instead of redesigning and reintegrating your external applications, you can now use these tools to integrate your technology within Canvas. 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 Aura on page 73—An Aura component that lets you expose your canvas app in a custom Aura component.
- Canvas Apps in Visualforce—A Visualforce component that lets you expose your canvas app on a Visualforce page. Canvas apps embedded in Visualforce using `<apex:canvasApp>` serialize parameters as part of a redirect URL during load with known limitations.
- Canvas Apps in the Publisher—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—Lets you expose your canvas apps as feed items.
- Canvas in the Salesforce Mobile App—Makes your canvas apps available in the Salesforce mobile app.

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).

Other Integration Options

Before diving into Canvas, consider these other options for integrating a third-party application in Salesforce.

Web tabs

Canvas apps present third-party applications as part of a page. Web tabs can present a full application in a large screen space.

HTML iframes in a custom component

Canvas apps provide greater functionality than developing with iframes. Iframes are sometimes easier to integrate with your application.

SEE ALSO:

Canvas Scenarios

Canvas App Process

Quick Start

Canvas Scenarios

From a high-level view, there are two common scenarios where 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 Canvas

Where Canvas Apps Appear

Where Canvas Apps Appear

Canvas apps can appear in various places, depending on your Canvas app settings.

Canvas apps can appear in a few places, either in both Salesforce Classic and Lightning Experience, or only in Salesforce Classic.

These Canvas location options are available for Salesforce Classic and Lightning Experience.

- In the Canvas App Previewer in the org 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
- As an option in the navigation menu in the Salesforce mobile app, for any user who has been allowed access to it
- In the Chatter publisher and Salesforce mobile app action bar, if you configure it to appear as a quick action
- In a Visualforce page after you add it to a page and make that page available to users
- In a page layout for a standard or custom object after you add the canvas app to the page layout. Depending on where you place the canvas app on the page layout, the canvas app can appear in the record detail page or in a mobile card.

These Canvas location options are available for Salesforce Classic only. External client apps with Canvas integrations don't support these location options.

- 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 a Salesforce Console after you add it as a custom console component
- 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 of the Canvas app settings.

- **Chatter Feed**—The canvas app appears in the 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.
- **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.
- **Layouts and Mobile Cards**—The canvas app can appear on a page layout or a mobile card. If this option is selected, you choose where the canvas app appears by adding it to the page layout.
- **Lightning Component**—The canvas app appears on your custom Lightning component using the `force:canvasApp` component.

- **Mobile Nav**—The canvas app is accessible from the mobile app navigation menu.
- **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 Chatter publisher and action bar. If this option is selected, you must also create a canvas custom action and add it to the global publisher layout or to an object's page layout.
- **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 Canvas

Set the Canvas App Location for Connected Apps

Supported Browsers

Supported Browsers

Learn where Canvas is supported.

Important: To prepare your Canvas apps for the new web browser restrictions on third-party cookies, see How Third-Party Cookie Restrictions Impact Salesforce Canvas Apps in Salesforce Help.

Canvas supports the most recent versions of the following browsers:

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

See the supported browsers topics in Salesforce Help for specific versions and some important considerations. Because Canvas is not built on Lightning, it still supports Microsoft® Internet Explorer® 11.

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:

Supported Browsers

Introducing Canvas

Supported Salesforce Editions

Alternatives to Cookies for User Tracking

Supported Salesforce Editions

Canvas supports these Salesforce Editions:

Edition	Create a canvas app	Publish a canvas app	Install a canvas app
Group	Yes*	No	Yes*

Edition	Create a canvas app	Publish a canvas app	Install a canvas app
Professional	Yes*	No	Yes*
Enterprise	Yes	No	Yes
Unlimited	Yes	No	Yes
Performance	Yes	No	Yes
Developer	Yes	Yes	Yes

*Group and Professional Editions can only use or create canvas apps that have a permitted users value of "Admin approved users are pre-authorized" if profiles and page layouts are enabled in the org.

SEE ALSO:

Introducing Canvas

User Permissions Required

User Permissions Required

You need certain user permissions to interact with canvas apps.

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

- "Customize Application"
- "Modify All Data"

Note: The "Customize Application" user permission is required to see Canvas apps in the App Launcher on mobile.

SEE ALSO:

Introducing Canvas

Configure Connected App Access

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 the Salesforce mobile app, take into account mobile device screen sizes. For more information, see Canvas in the Salesforce Mobile App.

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 apex:canvasApp Component.

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
- In the Salesforce navigation menu

We recommend that you use an image of size 60 pixels (wide) by 60 pixels (high) so that the icon appears correctly in both mobile and the Salesforce site. If you have no plans to display your icon in the Salesforce mobile app, you can use a smaller icon size of 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.

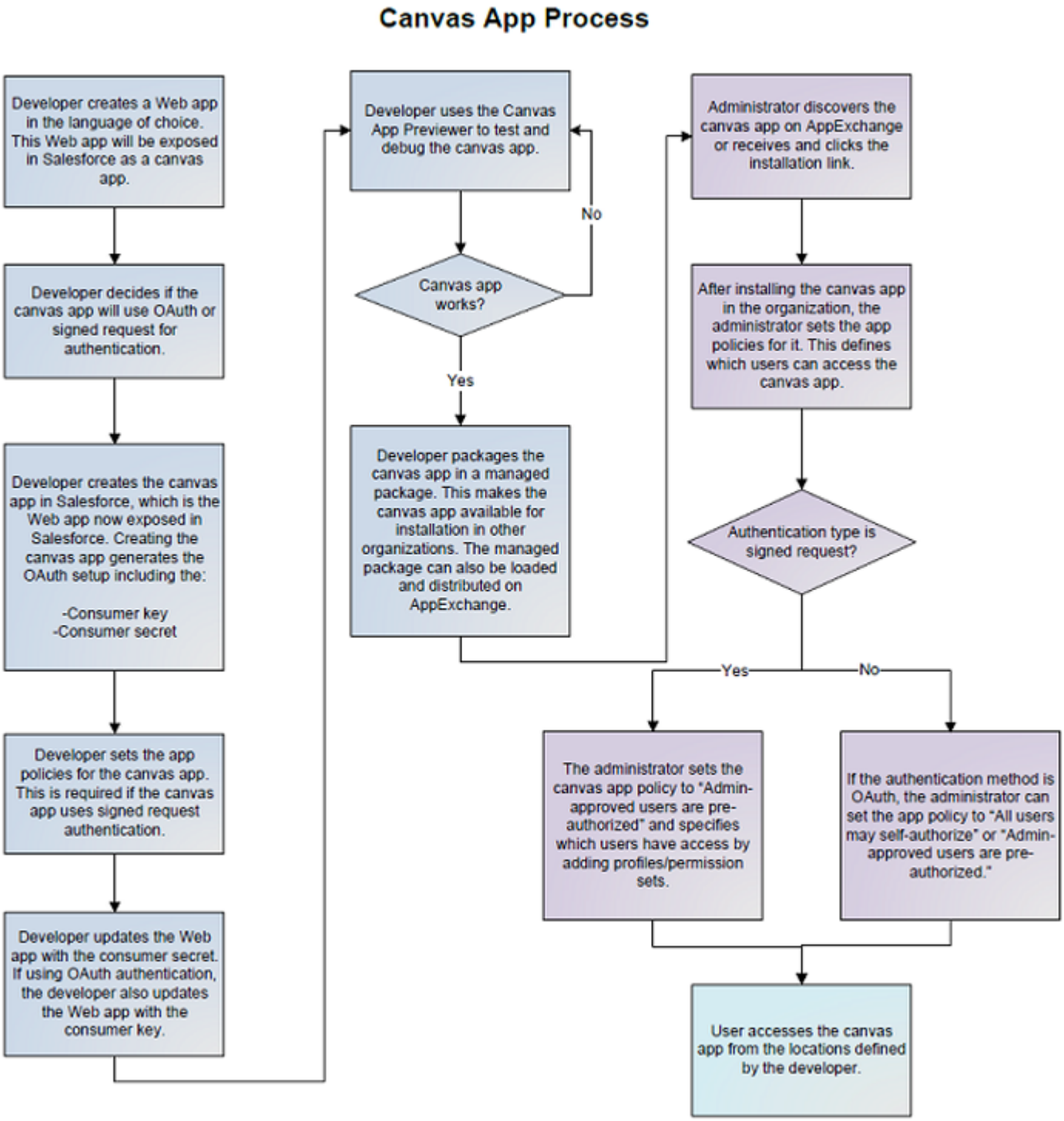
We recommend that you use an image of size 120 pixels (wide) by 120 pixels (high) or smaller.

SEE ALSO:

- Introducing 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.



Once the user is authenticated inside the Canvas app, the browser may restrict the app from receiving third-party cookies or using local storage. See Alternatives to Cookies for User Tracking on page 51.

SEE ALSO:

Introducing Canvas
Quick Start
Quick Start—Advanced
Canvas Personal App Process
Enabling Canvas Personal Apps within an Organization
Package the Connected App for Canvas

Canvas Personal Apps

Canvas personal apps let you create connected apps that are designed specifically for end users across orgs. With a canvas personal app, you make your own app available for installation without relying on org admins for distribution.

Note: In Spring '26, the ability to create new connected apps is disabled by default. We recommend using external client apps for all new Canvas integrations. To enable creation of new connected apps, contact Salesforce Customer Support.

Admins can install canvas apps, and developers can create canvas personal apps that end users can install themselves.

End users install and use canvas personal apps right from the Chatter tab (provided that they have appropriate permissions in Salesforce). When end users install the app, they're prompted to allow the app to use their Salesforce data.

Creating a canvas personal app is similar to creating any canvas app, except that:

- Users are prompted to approve or deny a canvas personal app the first time they access the app, regardless of whether you used OAuth or signed request as the access method. This is because initial access triggers the OAuth approval flow. After initial access, users are identified transparently and aren't prompted for approval or credentials (as long as they are logged in to Salesforce).
- With a canvas personal app, only the app's metadata is installed, without the other package components (such as Apex classes and Visualforce components). This facilitates widespread installation of the app, and lets you limit the app's functionality to only what the end user needs.

For information about managed packages, see Use Managed Packages to Develop Your AppExchange Solution.

Canvas Personal App Process

Creating a canvas personal app is similar to creating any canvas app. However, you distribute a canvas personal app directly to end users, who install it via a link that you provide, integrating the app with their Salesforce data. After the app is installed, end users run it right from the Chatter tab.

Enabling Canvas Personal Apps within an Organization

The administrator of an organization controls whether users can install canvas personal apps within that organization. Before attempting to create a canvas personal app, verify that the organization administrator has enabled canvas personal apps.

Making an App a Canvas Personal App

You can enable an app as a canvas personal app when you first create it, or make your existing canvas app a canvas personal app. This process involves ensuring that an org preference is active, and activating a setting on the app's Detail page.

Uninstalling a Canvas Personal App

When a user installs and allows a canvas personal app, a refresh token is created, which in turn generates an OAuth token. To stop end users from accessing the app, this refresh token must be revoked. This action effectively uninstalls the canvas personal app.

Canvas Personal App Process

Creating a canvas personal app is similar to creating any canvas app. However, you distribute a canvas personal app directly to end users, who install it via a link that you provide, integrating the app with their Salesforce data. After the app is installed, end users run it right from the Chatter tab.

Here's how to make canvas personal apps available to end users.

1. You verify that the organization administrator has enabled personal apps within the destination organization. For details, see Enabling Canvas Personal Apps within an Organization on page 9.
2. You create the app, and specify that it's a canvas personal app. For details, see Making an App a Canvas Personal App on page 10.
3. The end user clicks the app link that you've provided. For example, the link could be a button on your website, or delivered in an email. Clicking the link invokes the OAuth approval process for the app. Here's an example of what a link to the app looks like:

   ```
   https://MyDomainName.my.salesforce.com/services/oauth2/authorize?response_type=code&client_id=<your_client_id>&redirect_uri=<your_redirect_uri>
   ```

4. When prompted, the end user approves the app and allows access to Salesforce data.
5. The OAuth approval flow triggers the canvas installer in asynchronous mode. The canvas installer gets context information for the app from the API, and installs the app's metadata. The app is installed in the background, and the end user receives an email when installation is complete.

   Note: Only the canvas personal app's metadata is installed. Any additional components that are typically packaged with a canvas app are not installed with the app. This means that users get only the functionality they need, and installation is quick and lightweight. However, if the administrator chooses to install the canvas personal app for the entire organization, the additional package components are installed, as with any canvas app.

6. The user starts using the canvas personal app by clicking the app's icon from the Chatter tab.

SEE ALSO:

Canvas App Process

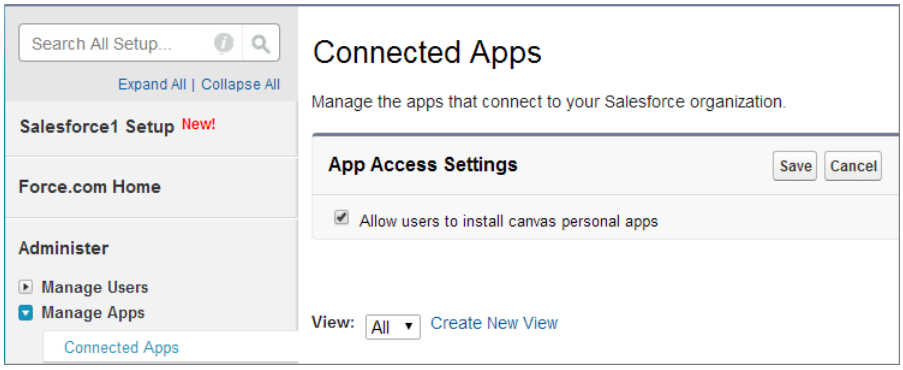
Making an App a Canvas Personal App

Enabling Canvas Personal Apps within an Organization

The administrator of an organization controls whether users can install canvas personal apps within that organization. Before attempting to create a canvas personal app, verify that the organization administrator has enabled canvas personal apps.

Here's how the organization administrator enables canvas personal apps.

1. From Setup, enter *Connected Apps* in the Quick Find box, then select the option for managing connected apps.
2. Next to App Access Settings, click **Edit**.
3. Select the **Allow users to install canvas personal apps** checkbox.



4. Click **Save**.

After the administrator has done these steps, you can designate your canvas app as a canvas personal app. For details, see Making an App a Canvas Personal App on page 10.

SEE ALSO:

- User Permissions Required
- Canvas Personal App Process

Making an App a Canvas Personal App

You can enable an app as a canvas personal app when you first create it, or make your existing canvas app a canvas personal app. This process involves ensuring that an org preference is active, and activating a setting on the app's Detail page.

Note: In Spring '26, the ability to create new connected apps is disabled by default. We recommend using external client apps for all new Canvas integrations. To enable creation of new connected apps, contact Salesforce Customer Support.

You can enable an existing connected app as a Canvas Personal app.

Before you begin, the admin for the app's destination org must enable canvas personal apps in that org. From Setup, enter *Connected Apps* in the Quick Find box, then select the option for managing connected apps.

1. Open the connected app's Detail page in Setup.

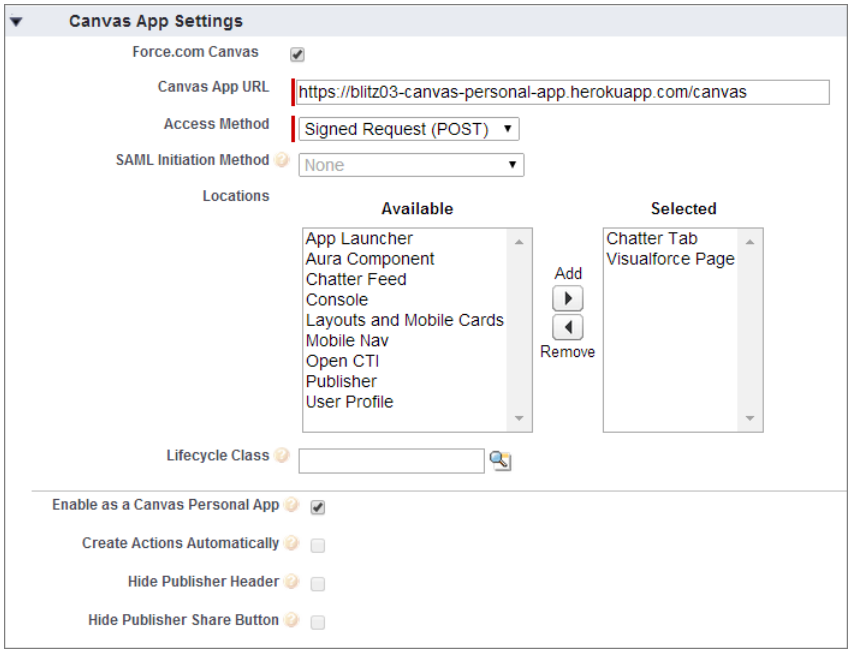
   Enter *Connected Apps* in the Quick Find box and select the app you want to make a canvas personal app.

2. In the **Locations** field, select **Chatter Tab**.

   Chatter Tab is the only location where a canvas personal app can appear.

3. Under Canvas App Settings, select the **Enable as a Canvas Personal App** checkbox.

   This setting is available only when **Chatter Tab** is selected as the app's location.



After you've created the canvas personal app and provided a link to it, end users can access the app and install it themselves.

SEE ALSO:

Canvas Personal App Process

Uninstalling a Canvas Personal App

When a user installs and allows a canvas personal app, a refresh token is created, which in turn generates an OAuth token. To stop end users from accessing the app, this refresh token must be revoked. This action effectively uninstalls the canvas personal app.

Here's how a canvas personal app can be uninstalled.

- The organization administrator removes the refresh token for one or all users in the organization via the Connected Apps OAuth Usage page in Setup.
- End users revoke the canvas personal app from their Connected Apps list.
- The administrator explicitly uninstalls the canvas personal app from the organization via the Manage Apps page in Setup. This action removes the app from the organization.

If an organization administrator disables canvas personal apps within the organization after a canvas personal app is installed, the canvas personal app continues to work. This is because the app's version number and URL persist, regardless of how the organization preference for canvas personal apps is set.

However, the administrator can explicitly uninstall the app, completely removing it from use within the organization.

SEE ALSO:

Signed Request Authentication

CHAPTER 2 Quick Start

In this chapter ...

- Prerequisites
- Create the App
- Set the Canvas App Location for External Client Apps
- Set the Canvas App Location for Connected Apps

This simple quick start shows you how to get started with 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 Canvas SDK to display information about the current user and lets you post to the current user's Chatter feed.

You can also specify where to display the Canvas app in Salesforce using the Canvas plugin for the external client app.

SEE ALSO:

Prerequisites

Create the App

Set the Canvas App Location for Connected Apps

Prerequisites

Get the appropriate access and tools to complete the quick start steps.

To complete the quick start, you must have these prerequisites.

- Access to a Developer Edition org.

  To create a Developer Edition org, go to `developer.salesforce.com/signup` and follow the instructions for signing up for a Developer Edition organization.

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

- "Customize Application" and "Modify All Data" user permissions. If you're an adm, 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`.
- Java version 1.6 or 1.7 to run the *local* instance of the "hello world" app that you create. (The Heroku instance of the app automatically downloads the correct version of Java.)

Note: In Spring '26, the ability to create new connected apps is disabled by default. We recommend using external client apps for all new Canvas integrations. To enable creation of new connected apps, contact Salesforce Customer Support.

SEE ALSO:

Quick Start

Supported Salesforce Editions

Create the App

Create the App

Create the Heroku "hello world" app and the associated canvas app in Salesforce.

1. In Salesforce, from Setup, enter *`Canvas App Previewer`* in the `Quick Find` box, then select **Canvas App Previewer**.
2. Click **Heroku Quick Start**.
3. The `Framework` field is **External Client App**. You can't change the framework selection. Your canvas app is created using the external client app framework.

   Creation of a canvas app using the connected app framework is disabled by default and is no longer recommended. To enable creation of new connected apps, contact Salesforce Customer Support.
4. In the `Template` field, select **Java – Quick Start Template**.
5. In the `Canvas App Name` field, enter a unique name of up to 30 characters.
6. 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`.
7. In the `Auth Type` field, select **API Key**. Find the API key associated with your account on the Heroku `My Account` page.
8. Click **Create**. The app displays in the left navigation pane.

   If you see the error "Error [Read timed out] executing POST to Heroku clone REST service," then the operation timed out when it tried to contact Heroku. You can check the status of Heroku at `http://status.heroku.com`.

9. Click the link to your new app.

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

You just created a canvas app—congratulations! You can see your canvas app only in the Canvas App Previewer until you set the locations where it can appear by following the steps in:

- Set the Canvas App Location for External Client Apps—Use an external client app for all new Canvas integrations after Spring '26.
- Set the Canvas App Location for Connected Apps—Older Canvas integrations can use connected apps, but new connected app creation is disabled after Spring '26.

The location setting defines where a user sees your app after it's installed in their org.

Behind the scenes, 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 created, and any users with that profile can access the canvas app.

We recommend that you inspect the code to see how the canvas app works. The source code for the app is available in the canvas-quickstart-java public GitHub repository. Use the example code to learn how to decode and validate the canvas signed request in your app.

SEE ALSO:

Quick Start

Prerequisites

Canvas Limits

Set the Canvas App Location for External Client Apps

Specify where your canvas app can display to a user in Salesforce via an external client app.

1. In Lightning Experience, from Setup, search for *External Client App* in the `Quick Find` box, then select **External Client App Manager**.
2. In the related list, click the dropdown arrow for the app you just created and then click **Edit Settings.**
3. In the Canvas Apps Settings section, in the `Locations` field, select where the canvas app can appear to the user. For this example, select **Lightning Component**.
   - **Chatter Feed**—The canvas app appears in the 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.
   - **Layouts and Mobile Cards**—The canvas app can appear on a page layout or a mobile card. If this option is selected, you choose where the canvas app appears by adding it to the page layout.
   - **Lightning Component**—The canvas app appears on your custom Lightning component using the `force:canvasApp` component.
   - **Mobile Nav**—The canvas app is accessible from the mobile app navigation menu.
   - **Publisher**—The canvas app appears in the Chatter publisher and action bar. If this option is selected, you must also create a canvas custom action and add it to the global publisher layout or to an object's page layout.
   - **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.
4. Click **Save**.
5. Embed the canvas app in a Lightning component.

a. Open the Developer Console.

b. Click **File** > **New** > **Lightning Component**.

c. Enter *myCanvasApp* in the `Name` field.

a. Copy and paste this code.

```
<!-- myCanvasApp.cmp -->
<aura:component implements="flexipage:availableForRecordHome,force:hasRecordId"
access="global" >
    <aura:attribute name="canvasParameters" type="string" />
    <aura:handler name="init" value="{!this}" action="{!c.doInit}"/>

    <aura:if isTrue="{!v.canvasParameters}">
        <force:canvasApp developerName="Hello_World"
            height="400px" width="750px"
            parameters="{!v.canvasParameters}"/>
    </aura:if>
</aura:component>
```

To verify the `developerName` value of your canvas app, in Setup, go to **External Client App Manager** and click on your app name. Then click the **Settings** tab and verify the **API Name**.

b. In the controller file, copy and paste this code.

```
({
    doInit : function(cmp, evt, helper) {
        var recordId = cmp.get("v.recordId");
        cmp.set("v.canvasParameters", JSON.stringify({
            recordId: recordId
        }));
    }
})
```

Save your changes.

6. Display the Lightning component on a record.

a. Navigate to the record page you want to customize, for example, a contact record page.

b. Click the gear icon and select **Edit Page.**

c. Find the `myCanvasApp` custom component and drag it to where you want it on the Lightning page. Save your changes.

d. Activate the page. Click **Assign as Org Default**, and then click **Next**.

Your canvas app is now displayed on contact record pages.

SEE ALSO:

Lightning Component Code Examples

Set the Canvas App Location for Connected Apps

Specify where your canvas app can display to a user in Salesforce via a connected app.

In Spring '26, the creation of new connected apps is disabled by default. We recommend that you use an external client app for your canvas integrations instead. To enable connected app creation in your org, contact Salesforce Customer Support.

Before Spring '26, when you create a canvas app via the Canvas App Previewer page, a connected app is created automatically.

1. From Setup, search for *Apps* in the `Quick Find` box, then select **App Manager** in Lightning Experience or **Connected Apps** in Classic.
2. In the related list, click the dropdown arrow for the app you just created and then click **Edit Settings.**
3. 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 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.
   - **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.
   - **Layouts and Mobile Cards**—The canvas app can appear on a page layout or a mobile card. If this option is selected, you choose where the canvas app appears by adding it to the page layout.
   - **Lightning Component**—The canvas app appears on your custom Lightning component using the `force:canvasApp` component.
   - **Mobile Nav**—The canvas app is accessible from the mobile app navigation menu.
   - **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 Chatter publisher and action bar. If this option is selected, you must also create a canvas custom action and add it to the global publisher layout or to an object's page layout.
   - **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.
4. Click **Save**.

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

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
- Deploy the Canvas External Client App to an Org
- Update the Canvas App
- Package 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 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 Canvas SDK to display the current user's name.

In this example, you:

- 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 org. You can either:
  - Use an external client app for your canvas app (recommended).
  - Use a connected app for your canvas app.

    Note: In Spring '26, the ability to create new connected apps is disabled by default. We recommend using external client apps for all new Canvas integrations. To enable creation of new connected apps, contact Salesforce Customer Support.
- Test the app in the Canvas App Previewer.
- Deploy the Web app to Heroku.
- Package and upload the canvas app.
- Install the canvas app in another Salesforce org 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 Canvas

Prerequisites

Prerequisites

Get the appropriate access and tools to complete the quick start.

To get started, you must have these prerequisites.

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

  To create a Developer Edition org, go to `developer.salesforce.com/signup` and follow the instructions for signing up for a Developer Edition organization.

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

- "Customize Application" and "Modify All Data" user permissions. If you're an admin, 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`.

  After you install Git, you might need to configure SSH by using the keygen tool. See `https://help.github.com/articles/generating-ssh-keys` for more information. If you're using Windows, this tool is located in the Git `\bin` directory, which isn't added to the path after you install Git. Add the `\bin` directory to your path by using Control Panel. Depending on your installation directory, the path might be something like `C:\Program Files (x86)\Git\bin`.

- 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`.

Note: In Spring '26, the ability to create new connected apps is disabled by default. We recommend using external client apps for all new Canvas integrations. To enable creation of new connected apps, contact Salesforce Customer Support.

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 Canvas SDK, which is located on GitHub. From GitHub, clone the project to make a copy of it on your local machine.

1. Log in to `https://github.com`.
2. Locate the project that contains the Canvas SDK and code examples by going to: `https://github.com/forcedotcom/SalesforceCanvasFrameworkSDK`.
3. To clone the Canvas SDK repository, follow the instructions in the project's `README.md` file.

After you clone the project, the `SalesforceCanvasFrameworkSDK` directory should contain a subdirectory named `src` and files, such as `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
- Use a Connected App

Create the Canvas App

To create a canvas app, you need the Customize Application and Modify All Data user permissions.

Previously, you created the canvas app as a connected app to use it in your org. Starting in Spring '26, we recommend that you use external client apps for your canvas integrations.

External client apps can't be used until they're installed in an org, unlike connected apps that can be used by any org without installation once it's created in an org locally. External client apps support the same OAuth flows with additional built-in security features while allowing you to package developer settings, keeping subscriber policies separate. External client apps also provide full metadata compliance, which makes packaging and deployment easier.

Use an External Client App
To create the canvas app in your org, configure the Canvas plugin for an external client app.

Use a Connected App
Creating the canvas app as a connected app is no longer recommended. Use external client apps for all new Canvas integrations.

SEE ALSO:

- External Client Apps

Use an External Client App

To create the canvas app in your org, configure the Canvas plugin for an external client app.

1. In Salesforce, from Setup, enter *Apps* in the Quick Find box, then select **App Manager**.
2. Click **New External Client App**.
3. In the External Client 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. Select one of these options in the Distribution State field.
   - **Local**: Develop an external client app for your local org
   - **Packaged**: Develop an external client app for packaging and distribution
7. In the Logo Image URL field, enter `https://localhost:8443/images/salesforce.png`. This is the default Salesforce "No Software" image. This image appears on the installation screen and on the detail screen for the app.
8. In the Icon URL field, enter `https://localhost:8443/examples/hello-world/logo.png`. This is the default Salesforce "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.
9. In the API (Enable OAuth Settings) section, select Enable OAuth.
10. In the Callback URL field, enter `https://localhost:8443/sdk/callback.html`.
11. In the OAuth Scopes field, select **Access the identity URL service (id, profile, email, address, phone)** from the Available OAuth Scopes list and move it to the Selected OAuth Scopes list.
12. In the Canvas App Settings section, select Enable Canvas.
13. In the Canvas App URL field, enter `https://localhost:8443/examples/hello-world/index.jsp`.
14. In the Access Method field, select Signed Request (POST).
15. In the Locations field, select **Lightning Component** from the Available list and move it to the Selected list.
16. Click **Create**
    After the canvas app is saved, the detail page appears.
17. In the **Settings** tab, expand the **OAuth Settings** section, and click **Consumer Key and Secret**. Then, verify your identity.
18. Select the consumer secret value and press Ctrl+c (Windows) to copy it. The consumer secret is used in the app to authenticate.
19. Go to the command window and stop the Jetty Web server by pressing Ctrl+c (Windows). At the prompt, enter Y.
20. 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>
```

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

SEE ALSO:

Lightning Component Code Examples

Configure Connected App Access

Use a Connected App

Creating the canvas app as a connected app is no longer recommended. Use external client apps for all new Canvas integrations.

Note: In Spring '26, the ability to create new connected apps is disabled by default. To enable creation of new connected apps, contact Salesforce Customer Support. We recommend using an external client app on page 21 instead for all new Canvas integrations.

1. In Salesforce, from Setup, enter `Apps` in the `Quick Find` box, then select **App Manager**.
2. Click **New Connected App**.
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 "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 "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 the identity URL service (id, profile, email, address, phone)**.
11. In the Canvas App Settings section, select `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. Save the app, then click **Continue.**
    After the canvas app is saved, the detail page appears.
16. On the detail page for the canvas app, click **Manage Consumer Details**, and then verify your identity.
17. Select the consumer secret value and press Ctrl+c (Windows) to copy it. The consumer secret is used in the app to authenticate.
18. Go to the command window and stop the Jetty Web server by pressing Ctrl+c (Windows). 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
- Canvas Limits
- Configure Connected App Access

Configure Who Can Access the Canvas App

To configure access for specific users, update the policies and profiles for the corresponding external client app or connected app.

Before you configure access for the canvas app, make sure you created the external client app on page 21. In Winter '26 and earlier, you created a connected app on page 23 for your canvas integrations.

Configure External Client App Access
To configure access to your canvas app, configure the policies and profiles for the corresponding external client app.

Configure Connected App Access
To configure access to your canvas app, configure the policies and profiles for the corresponding connected app.

Configure External Client App Access

To configure access to your canvas app, configure the policies and profiles for the corresponding external client app.

Note: This task assumes that you completed the steps at Use an External Client App.

1. In Lightning Experience, from Setup, enter *External Client Apps* in the Quick Find box, then select **External Client App Manager**.
2. In the External Client App list, click **Edit Policies** in the dropdown menu next to the Hello World app.
3. In the App Policies section, select the **System Administrator** profile from the Available Profiles list and move the selection to the Selected Profiles list. Repeat this step for the profiles you want to select.

   In this example, we assume that your profile is System Administrator.
4. In the OAuth Policies section, click the **Permitted Users** dropdown menu and select **Admin approved users are pre-authorized**. Click **OK** on the pop-up message that appears.

   We select **Admin approved users are pre-authorized** so users don't see the canvas app until we give them permission. If we select **All users can self-authorize**, all users in the org can authorize the app after they sign in. Save your settings.
5. From Setup, enter *Canvas App Previewer* in the Quick Find box, then select **Canvas App Previewer**. You can use the Canvas App Previewer to test your canvas app before you publish it.
6. From the Canvas App Previewer list, click **Hello World**.

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

After you configure access, make your canvas app available in a Lightning component. You can add the component to a Lightning record page so permitted users can use it there.

7. Embed the canvas app in a Lightning component.
   a. Open the Developer Console.
   b. Click **File** > **New** > **Lightning Component**.
   c. Enter *myCanvasApp* in the `Name` field.
   a. Copy and paste this code.

```
<!-- myCanvasApp.cmp -->
<aura:component implements="flexipage:availableForRecordHome,force:hasRecordId"
access="global" >
    <aura:attribute name="canvasParameters" type="string" />
    <aura:handler name="init" value="{!this}" action="{!c.doInit}"/>


    <aura:if isTrue="{!v.canvasParameters}">
        <force:canvasApp developerName="Hello_World"
            height="400px" width="750px"
            parameters="{!v.canvasParameters}"/>
    </aura:if>
</aura:component>
```

   To verify the `developerName` value of your canvas app, in Setup, go to **External Client App Manager** and click on your app name. Then click the **Settings** tab and verify the **API Name**.

   b. In the controller file, copy and paste this code.

```
({
    doInit : function(cmp, evt, helper) {
        var recordId = cmp.get("v.recordId");
        cmp.set("v.canvasParameters", JSON.stringify({
            recordId: recordId
        }));
    }
})
```

   Save your changes.

8. Display the Lightning component on a record.
   a. Navigate to the record page you want to customize, for example, a contact record page.
   b. Click the gear icon and select **Edit Page.**
   c. Find the `myCanvasApp` custom component and drag it to where you want it on the Lightning page. Save your changes.
   d. Activate the page. Click **Assign as Org Default**, and then click **Next**.

Users with the profiles you selected in the App Policies section can now view and use the canvas app on contact record pages.

SEE ALSO:

Lightning Component Code Examples

Configure Connected App Access

To configure access to your canvas app, configure the policies and profiles for the corresponding connected app.

Note: This task assumes that you completed the steps at Use a Connected App on page 23.

1. In Salesforce, from Setup, enter `Connected Apps` in the Quick Find box, then select **Manage Connected Apps**.
2. In the Connected Apps list, click **Edit** next to the Hello World app.
3. In the OAuth Policies section, click the **Permitted Users** dropdown menu and select **Admin approved users are pre-authorized**. Click **OK** on the pop-up message that appears.

   We select **Admin approved users are pre-authorized** so users don't see the canvas app until we give them permission. If we select **All users may self-authorize**, all users in the org can authorize the app after they sign in.
4. Save your settings and return to the Manage Connected Apps page.
5. To view the Connected App Detail page for your Hello World app, click **Hello World** in the Connected Apps list.

   On the Connected App Detail page, you can define who can see your canvas app by managing profiles and permission sets.
6. In the Profiles section, click **Manage Profiles**.

   In this example, we assume that your profile is System Administrator.
7. Select the **System Administrator** profile and save your settings.
8. From Setup, enter `Canvas App Previewer` in the Quick Find box, then select **Canvas App Previewer**. You can use the Canvas App Previewer to test your canvas app before you publish it.
9. From the Canvas App Previewer list, click **Hello World**.

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

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

SEE ALSO:

Quick Start—Advanced

Use a Connected App

Deploy the Web App to Heroku

Deploy the Web App to Heroku

After you run the web app locally and test it as a canvas app, deploy the web app to Heroku and run it from there.

Warning: Using dynos to complete this tutorial counts toward your usage. Scale your dynos to 0 as soon as you're done to minimize costs.

Important: Where possible, we changed noninclusive terms to align with our company value of Equality. We maintained certain terms to avoid any effect on customer implementations.

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 this command: `git init`. This re-initializes the directory as a Git repository.
3. In the command window, enter this 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 this 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 this 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 this command in the command window to commit the changes along with a comment: `git commit –m "`***`MyChangeComments`***`"`.
7. Enter this 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, enter *`Apps`* in the `Quick Find` box, then select **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 this 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 canvas environment.

SEE ALSO:

Quick Start—Advanced

Configure Connected App Access

Update the Canvas App

Deploy the Canvas External Client App to an Org

Deploy and retrieve your Canvas integration for the external client app with Metadata API. The Canvas plugin has its own settings files and policies file.

Before you can deploy or retrieve an external client app, set up an SFDX project.

If you created the Canvas external client app on page 21 in Salesforce Setup, you can retrieve the app using Metadata API to configure the policies and then deploy your changes to your org.

1. Retrieve your Canvas plugin files:

```
sf project retrieve start --manifest package.xml --target-org user@example.com
```

2. Make your changes in the `.js-meta.xml` file for the Canvas plugin. For example, to update the Canvas URL and location options, update the `canvasUrl` and `canvasLocationOptions` fields in the `.ecaCanvas-meta.xml` file that's located in the `force-app/main/default/extlClntAppCanvasSettings` folder of your SFDX project.
3. Save the metadata files with your changes.
4. Deploy your Canvas plugin files:

```
sf project deploy start --manifest package.xml --target-org user@example.com
```

5. Verify your changes in the External Client App Manager page in Setup.
   a. In External Client App Manager, click the name of your external client app.
   b. Go to the Settings tab and then expand the **Canvas App Settings** section.
   c. Check that the **Canvas App URL** and **Locations** fields show your changes.

The Canvas plugin files include the metadata for the external client app, the Canvas settings, policies, and OAuth settings. This table lists the name of the source file with its type and path, where `API_Name` corresponds to the API Name for the external client app.

Name	Metadata Type	Sample Path
API_Name	ExternalClientApplication	force-app/main/default/externalClientApps/API_Name.eca-meta.xml
API_Name_canvasSettings	ExtlClntAppCanvasSettings	force-app/main/default/extlClntAppCanvasSettings/API_Name_canvasSettings.ecaCanvas-meta.xml
API_Name_plcy	ExtlClntAppConfigurablePolicies	force-app/main/default/extlClntAppPolicies/API_Name_plcy.ecaPlcy-meta.xml
API_Name_glbloauth	ExtlClntAppGlobalOauthSettings	force-app/main/default/extlClntAppGlobalOauthSettings/API_Name_glbloauth.ecaGlblOauth-meta.xml

Name	Metadata Type	Sample Path
API_Name_oauth	ExtlClntAppOauthSettings	

SEE ALSO:

External Client Apps Creation with Metadata API

Update the Canvas App

In this step, you update the canvas app to run the "hello world" app that's now running on Heroku.

1. In Salesforce, from Setup, enter *External Client Apps* in the Quick Find box, then select **External Client App Manager**. Then, click **Edit Settings** on the dropdown menu next to the Hello World app.

   If you created a connected app instead, find your app on the **App Manager** setup page. Click **Edit** on the dropdown menu next to the Hello World app.

2. In the Logo Image URL field, enter `https://appName.herokuapp.com/images/salesforce.png`. This is the default Salesforce "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.

3. In the Icon URL field, enter `https://appName.herokuapp.com/examples/hello-world/logo.png`. This is the default Salesforce "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.

4. In the **OAuth Settings** section, update Callback URL to `https://appName.herokuapp.com/sdk/callback.html`. The *appName* is the name of Heroku app that you just created.

5. In the **Canvas App Settings** section, update Canvas App URL to `https://appName.herokuapp.com/examples/hello-world/index.jsp`. The *appName* is the name of Heroku app that you just created.

6. Click Save. After the canvas app is saved, the detail page appears.

7. In Salesforce, from Setup, enter *Canvas App Previewer* in the Quick Find box, then select **Canvas App Previewer**. You can use the Canvas App Previewer to test out your canvas app before repackaging it.

8. 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 Use an External Client 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 Connected App for Canvas

Package the Canvas App

Packaging is the first step in making your canvas app available for installation in another org.

To package the canvas app, follow one of these methods.

- If you used an external client app for your canvas app, package the app using a second-generation managed package on page 31. This is the recommended approach.
- If you used a connected app for your canvas app, package the app using a first-generation managed package.

Package the External Client App for Canvas
After you created the canvas app, tested it in the Canvas App Previewer, and deployed it to Heroku, you're ready to package and distribute it.

Package the Connected App for Canvas
After you created the canvas app, tested it in the Canvas App Previewer, and deployed it to Heroku, you're ready to package and distribute it.

Upload the Canvas Connected App Package
Now that you packaged the canvas app, you're ready to upload the package.

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 org.

Package the External Client App for Canvas

After you created the canvas app, tested it in the Canvas App Previewer, and deployed it to Heroku, you're ready to package and distribute it.

Package an external client app for your canvas integration by creating a second-generation managed package (managed 2GP) using Salesforce CLI commands from a Salesforce DX project.

1. Configure default settings for a packaged external client. The default values display in the policies tab on a distributed external client app and can be edited by admins.
2. Enable a Dev Hub org with a namespace prefix. The Dev hub org becomes the owner of your managed 2GP. A namespace is assigned to the package at the time that it's created, and can't be changed.
3. Configure a Salesforce DX project for external client apps. With a Salesforce DX project, you use command-line interface (CLI) commands to work with your Salesforce org.
4. Create a 2GP managed package using CLI commands. You can also create versions of the package to manage your changes and track what's different each time you release or deploy.

5. Install a managed 2GP in a scratch org to test the package. You can install or uninstall a managed 2GP using a Salesforce CLI command, or from the Setup page.
6. Distribute and release the managed 2GP. A new managed 2GP version is marked as beta when you create it. You can create multiple versions before creating a version that's ready to be released and distributed.

You can configure the canvas settings before or after creating the external client app. After the canvas setting is created, it can only be disabled or enabled. The canvas setting can't be deleted from Setup. However, you can delete the setting from the package.

SEE ALSO:

Package an External Client App

Workflow for Second-Generation Managed Packages

Package the Connected App for Canvas

After you 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 org. For more information about packaging a connected app for your canvas integration, see Use Managed Packages to Develop Your AppExchange Solution.

1. In Salesforce, from Setup, enter *Packaging* in the Quick Find box, then select **Package Manager** and click **New**.

   Tip: To package your canvas app for installation in other orgs, you must create a namespace prefix. A namespace prefix can be defined only in a Developer Edition org. For more information, see Register a Namespace for a First-Generation Managed Package.

2. Set the Package Name to Hello World Package and accept the defaults for the other fields.
3. Select Managed to make the package the managed package for the org.
4. A prompt appears stating you're allowed only 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, enter *Packages* in the Quick Find box, then select **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 org.

SEE ALSO:

Package an External Client App

Upload the Canvas Connected App Package

Distributing Your CanvasLifecycleHandler Class

Upload the Canvas Connected App Package

Now that you packaged the canvas app, you're ready to upload the package.

Uploading the package creates an installation link that you can provide to anyone who needs to install your canvas app in their org.

Note: This task assumes that you completed the steps at Package the Connected App for Canvas on page 32.

1. In Salesforce, from Setup, enter *Packaging* in the Quick Find box, then select **Package Manager**.
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 Spring 2026 Beta. Keep the default Version Number of 1.0 and Release Type of Managed—Beta.
5. Click **Upload**.

   Note: A canvas app can be installed only in an org 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 org. You also receive a notification email that contains the installation URL.

SEE ALSO:

Quick Start—Advanced
Package the Connected App for Canvas
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 org.

Beta packages can be installed only in sandbox or Developer Edition orgs, otherwise, you receive an error.

Note: This task assumes that you completed the steps at Upload the Canvas Connected App Package on page 32.

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 org 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 org 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 receive an email notification.

SEE ALSO:

Quick Start—Advanced
Upload the Canvas Connected 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, enter *Connected Apps* in the Quick Find box, then select the option for managing 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

CHAPTER 4 Canvas SDK

In this chapter ...

- Referencing the Canvas SDK
- Authentication
- Getting Context in Your Canvas App
- Cross-Domain XHR
- Alternatives to Cookies for User Tracking
- Resizing a Canvas App
- Implementing Canvas App Events
- Using Streaming API in a Canvas App
- Debugging in a Canvas App
- Using the <select> Tag in a Canvas App Tag in a Canvas App

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.

Important: To prepare your Canvas apps for the new web browser restrictions on third-party cookies, see How Third-Party Cookie Restrictions Impact Salesforce Canvas Apps in Salesforce Help.

The Canvas SDK and code examples are available on GitHub at `https://github.com/forcedotcom/salesforcecanvasframeworksdk`.

The Canvas SDK is versioned and matches the API version in each release. The current version is 66.0. To check the version of the SDK that you have, call the `version` method.

SEE ALSO:

- Referencing the Canvas SDK
- Authentication
- Getting Context in Your Canvas App
- Cross-Domain XHR
- Resizing a Canvas App
- Implementing Canvas App Events

Referencing the Canvas SDK

The 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/66.0/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:

Canvas SDK

Authentication

Authentication

When you create a Canvas app, you can use the signed request authentication method or the OAuth 2.0 authentication method.

Important: If your Canvas app URL contains a URL fragment identifier (#), then the hash mark (#) and all characters that follow are stripped from the URL during the authentication flow. To prevent unexpected behavior, avoid using hash marks (#) in a Canvas app URL.

- Signed request—The default method of authentication for Canvas apps. The signed request containing the consumer key, access token, and other contextual information is provided to the Canvas app in one of these ways.
  - The administrator allows access to the Canvas app for the user.
  - The user approves the Canvas app in the OAuth flow.
- OAuth 2.0—Canvas apps can use the OAuth 2.0 protocol to authorize and acquire access tokens. For more information about OAuth and the Lightning Platform, see Authorize Apps with OAuth.

Signed Request Authentication

Signed request is the default authorization method for Canvas apps. The signed request authorization flow varies depending on whether the administrator gives users access to the Canvas app or if users can self-authorize. You can verify signed request information with a client secret, then use signed requests to customize the app and make subsequent calls to Salesforce.

OAuth 2.0 Authorization

Canvas supports the OAuth 2.0 web server flow and the OAuth 2.0 user-agent flow.

SAML Single Sign-On for Canvas Apps

Whether you use signed request or OAuth authorization, you can use SAML-based single sign-on (SSO) to provide your users with a seamless authentication flow. You can leverage Salesforce as an identity provider or as a service provider. SAML SSO enables you to give your users automatic authentication into your canvas app via SAML and authentication into Salesforce via the signed request.

SEE ALSO:

Canvas SDK

Understanding Authentication

Canvas App User Flow—Signed Request

Canvas App User Flow—OAuth

Signed Request Authentication

Signed request is the default authorization method for Canvas apps. The signed request authorization flow varies depending on whether the administrator gives users access to the Canvas app or if users can self-authorize. You can verify signed request information with a client secret, then use signed requests to customize the app and make subsequent calls to Salesforce.

Important: If your Canvas app URL contains a URL fragment identifier (#), then the hash mark (#) and all characters that follow are stripped from the URL during the authentication flow. To prevent unexpected behavior, avoid using hash marks (#) in a Canvas app URL.

Permitted Users Value	Canvas App Accessibility	When the User Must Approve the Canvas App	POST or GET Behavior
Admin-approved users are pre-authorized	The app is accessible to users as soon as the administrator installs it in the organization and configures which users can see it. Users don't approve or deny access.	Never	Salesforce performs a POST to the Canvas app with all the authorization information contained in the body of the signed request, including the refresh token.
All users can self-authorize	The app is accessible to all users, but the user is prompted to approve or deny access to the app.	• The first time that the user opens the app • The administrator revokes the access token • The administrator sets a time limit on the token and that time limit is exceeded	If the user previously approved the app and access hasn't expired or been revoked, Salesforce performs a POST to the Canvas app with a signed request payload. If the user hasn't approved the app, or if access has expired or been revoked, Salesforce performs a GET to the Canvas app URL. The Canvas app must handle the GET by accepting the call and looking for the URL parameter `_sfdc_canvas_authvalue`. If the Canvas app receives this parameter value, the Canvas app initiates the approve or deny OAuth flow. `_sfdc_canvas_authvalue = user_approval_required`

Permitted Users Value	Canvas App Accessibility	When the User Must Approve the Canvas App	POST or GET Behavior
			After the OAuth flow is initiated and the user approves the app, the Canvas app calls the `repost()` method with a parameter of *`true`* to retrieve the signed request.

The signed request is a string of these elements concatenated.

- The Canvas app consumer secret encrypted with the HMAC SHA–256 algorithm
- A period (`.`)
- The context and authorization token JSON encoded in Base64

The signed request looks similar to this string, although it can be longer:

```
9Rpl6rE7R2bSNjoSfYdERk8nffmgtKQNhr5U/5eSJPI=.eyJjb250ZXh0Ijp7InVzZXIiOnsibGFuZ3V…
```

Signed request considerations:

- Salesforce performs an HTTP POST or GET when invoking the Canvas app URL, depending on the Permitted Users value and whether the refresh token is returned.
- Server-side code verifies and decodes the request.
- After the app is invoked, you can request a signed request on demand by using the SDK.

Canvas App User Flow—Signed Request

Verifying and Decoding a Signed Request

Requesting a Signed Request

If your canvas app is set to use signed request for authentication, you can request a signed request on demand by using the SDK. Your app can ask Salesforce to resend the signed request that was used when your app was loaded or to send a new signed request. Requesting a signed request is useful when you are refreshing an expired session or getting authentication information after your app has been redirected.

SEE ALSO:

Authentication

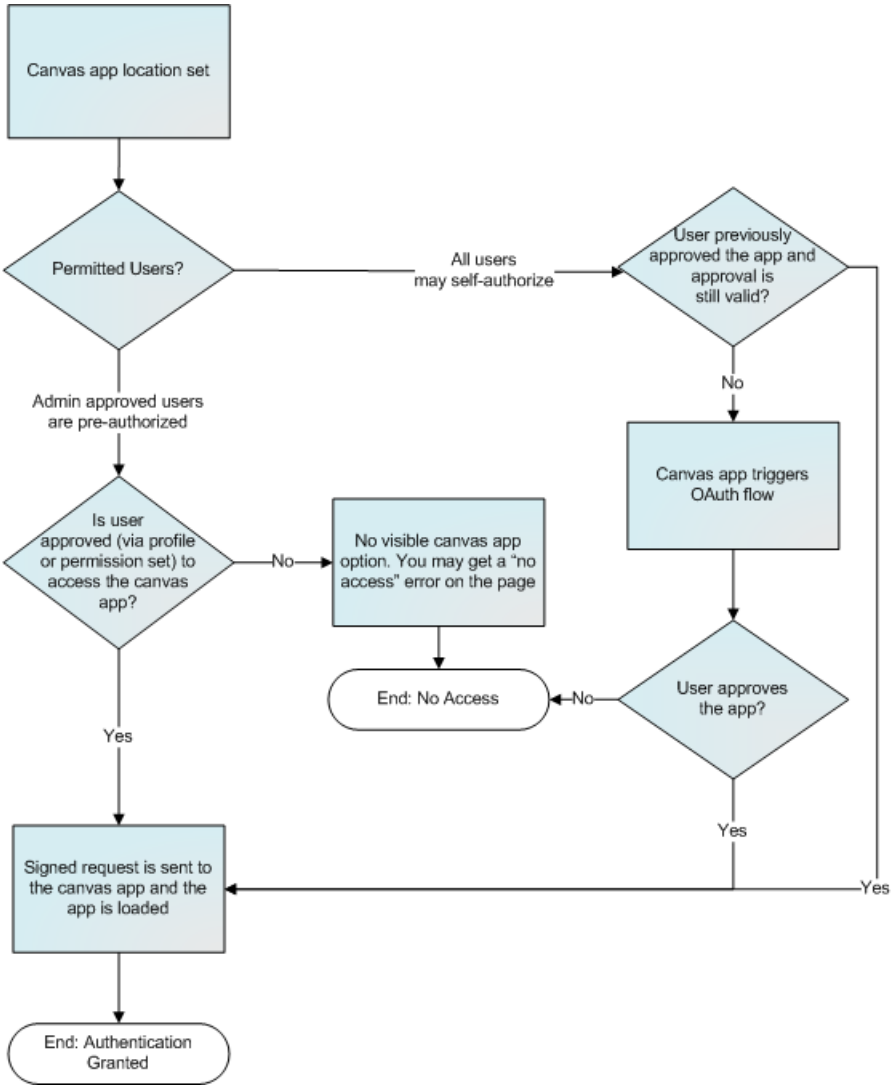
Verifying and Decoding a Signed Request

Canvas App User Flow—Signed Request

Requesting a Signed Request

Canvas App User Flow—Signed Request

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 Based64 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 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

Requesting a Signed Request

If your canvas app is set to use signed request for authentication, you can request a signed request on demand by using the SDK. Your app can ask Salesforce to resend the signed request that was used when your app was loaded or to send a new signed request. Requesting a signed request is useful when you are refreshing an expired session or getting authentication information after your app has been redirected.

You can request a signed request on demand by using the `refreshSignedRequest()` or `repost()` JavaScript methods in the SDK. `refreshSignedRequest()` returns a new signed request via a callback, while `repost()` asks the parent window to initiate a new POST to your canvas app and reloads the app page with a refreshed signed request.

Use these methods whenever you need to refresh the signed request for your app. Here are some example scenarios where you might need to refresh the signed request information.

- The OAuth token within a signed request typically expires after two hours. If the OAuth token has expired, and you need to make additional API requests, you can call `refreshSignedRequest()` to get a new OAuth token without interrupting the user.
- Canvas apps might need to use redirects, particularly when trying to provide single sign-on functionality across multiple systems. If your canvas app uses redirects, the redirect URL will not receive the original request body that contains the initial signed request information. You can use the new methods to request the signed request again.
- In Summer '14 and later, canvas apps can be user-approved apps. If your user-approved app has not been approved by the user, your app will not get an initial signed request POST. Instead, your app will need to be approved through OAuth, and then you can call `repost()` to get the signed request.

Your canvas app must be configured to use signed request for authentication to use these methods. You'll also need to reference `canvas-all.js` in your JavaScript code, as described in Referencing the Canvas SDK.

After a request succeeds, your canvas app must verify the returned SignedRequest information. For more information on verifying signed request information that is received from Salesforce, see Verifying and Decoding a Signed Request.

Using `refreshSignedRequest()` to Obtain a Signed Request

Use `refreshSignedRequest()` to request a new signed request via a callback that you provide. In this example, `refreshSignedRequest()` is called with a callback that checks the response code and then parses the signed request into the hashed Based64 context that is signed with the consumer secret and the Base64 encoded context itself.

```
// Gets a signed request on demand.
Sfdc.canvas.client.refreshSignedRequest(function(data) {
    if (data.status === 200) {
        var signedRequest =  data.payload.response;
        var part = signedRequest.split('.')[1];
        var obj = JSON.parse(Sfdc.canvas.decode(part));
    }
}
```

Using `repost()` to Obtain a Signed Request

Use `repost()` to instruct the parent window to send a new POST to your canvas app URL. The POSTcontains the signed request that was used when your app was loaded or a new signed request. Unlike `refreshSignedRequest()`, using `repost()` reloads the canvas app page. The following example calls `repost()`, asking for the original signed request.

```
// Gets a signed request on demand, without refreshing the signed request.
Sfdc.canvas.client.repost();
```

The following example calls `repost()`, asking for a new signed request.

```
// Gets a signed request on demand, first by refreshing the signed request.
Sfdc.canvas.client.repost({refresh : true});
```

SEE ALSO:

- Signed Request Authentication
- SignedRequest
- Verifying and Decoding a Signed Request

OAuth 2.0 Authorization

Canvas supports the OAuth 2.0 web server flow and the OAuth 2.0 user-agent flow.

Important: If your Canvas app URL contains a URL fragment identifier (#), then the hash mark (#) and all characters that follow are stripped from the URL during the authentication flow. To prevent unexpected behavior, avoid using hash marks (#) in a Canvas app URL.

When using OAuth with Canvas, you have two options.

- Web server flow—To integrate a Canvas app with the Salesforce API, use the OAuth 2.0 web server flow, which implements the OAuth 2.0 authorization code grant type. With this flow, the server hosting the web app must be able to protect the connected app's identity, defined by the client ID and client secret. For more information, see OAuth 2.0 Web Server Flow for Web App Integration in *Salesforce Help*.
- User-agent flow—With the OAuth 2.0 user-agent flow, users authorize a Canvas app to access data using an external or embedded browser. This flow uses the OAuth 2.0 implicit grant type. For more information, see OAuth 2.0 User-Agent Flow for Desktop or Mobile App Integration in *Salesforce Help*.

Regardless of which OAuth flow that 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 the user agent flow, all authorization is performed in the browser, and no server-side code is needed.

For more information about OAuth, see Authorize Apps with OAuth in *Salesforce Help*.

Existing Connected Apps and OAuth

If you have an existing connected app that uses OAuth authorization and you want to expose that app as a Canvas app, you have two options.

- Edit the existing app, and add the Canvas app information to it. Your app can continue to use the same client ID and secret.
- Create a new Canvas app, which gets a new client ID and consumer secret. Make sure to update your app with the client ID and secret.

Canvas App User Flow—OAuth

If your canvas app uses OAuth authorization, the user experience varies depending on where the canvas app is located in the user interface and how user access is set.

Initiate OAuth Flow

Start the authorization process in your Canvas app by using OAuth 2.0. If you store or retrieve data, such as an authentication token, from your Canvas app's local storage in the callback, use `window.opener.localStorage` instead of `window.localStorage`.

SEE ALSO:

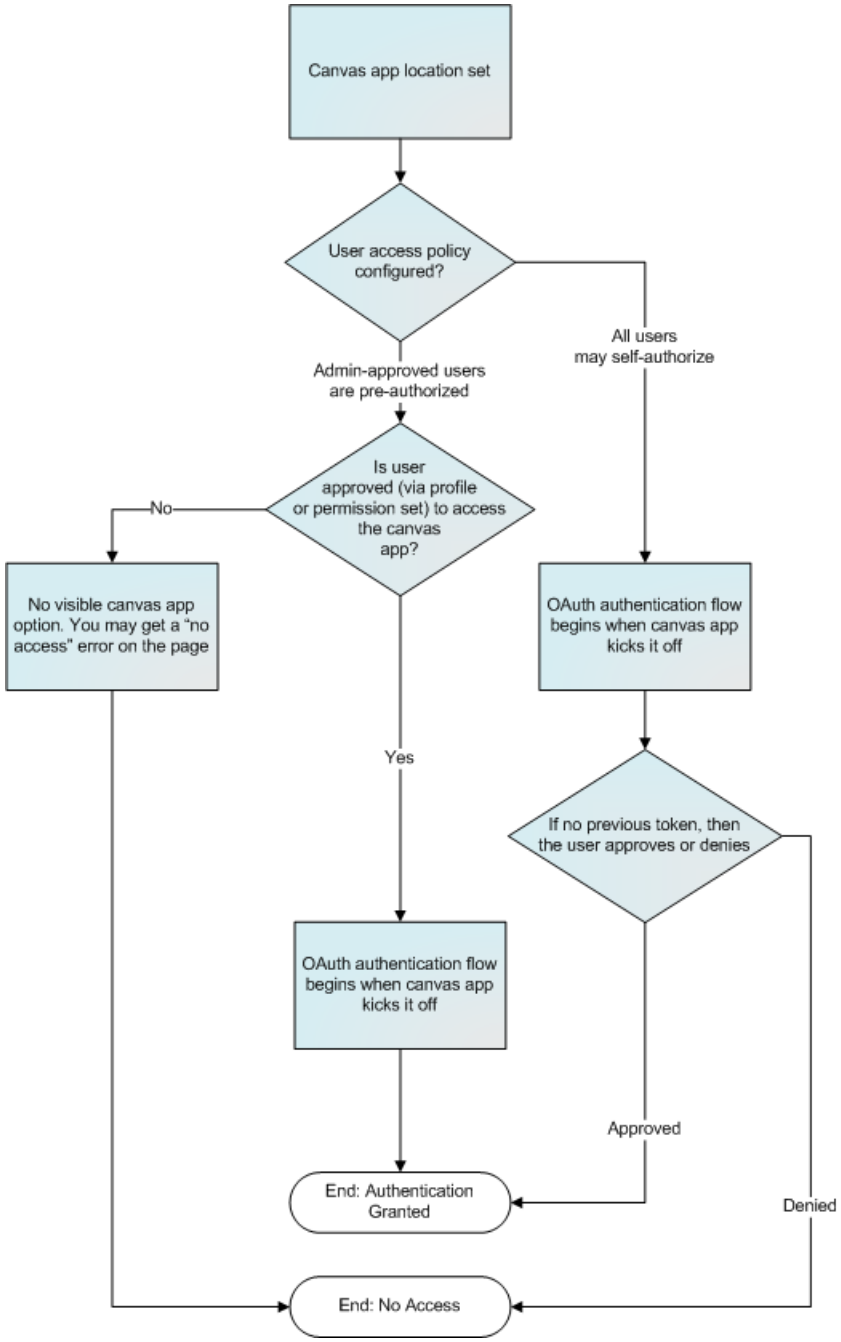
Authentication

Canvas App User Flow—OAuth

Canvas App User Flow—OAuth

If your canvas app uses OAuth authorization, the user experience varies depending on where the canvas app is located in the user interface and how user access is set.

This diagram shows the user flow for a canvas app that uses OAuth authorization.



SEE ALSO:

OAuth 2.0 Authorization

Initiate OAuth Flow

Initiate OAuth Flow

Start the authorization process in your Canvas app by using OAuth 2.0. If you store or retrieve data, such as an authentication token, from your Canvas app's local storage in the callback, use `window.opener.localStorage` instead of `window.localStorage`.

This code example shows how you can initiate the OAuth 2.0 authorization process in your Canvas app.

```
<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 after 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) {
                 // Displaying only 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">Canvas OAuth App</h1>
    <div>
        access_token = <span id="oauth"></span>
    </div>
    <div>
        <a id="login" href="#">Login</a><br/>
    </div>
```

```
</body>
</html>
```

When you implement an OAuth 2.0 web server flow, a window pops up so the user can authenticate. If your browser partitions storage, any data stored with storage APIs such as localStorage and sessionStorage is accessible only in contexts with the same top-level site. Because the callback is in a different window than the Canvas app, it has a different top-level site. Therefore, if you use `window.localStorage` in the callback to store and retrieve data, such as an authentication token, then an `Unauthorized: client` or `client.oauthToken` error occurs.

Instead, to store and retrieve data from your Canvas app's local storage in the callback, use `window.opener.localStorage`. With the opener property, you can interact with the window that opens another window if they both have the same origin. Because the callback has the same origin as the Canvas app, the local storage is accessible.

Note: Google Chrome and other major browsers now enable storage partitioning by default. For information about their browser requirements, see How Third-Party Cookie Restrictions Impact Salesforce Canvas Apps.

SEE ALSO:

OAuth 2.0 Authorization

SAML Single Sign-On for Canvas Apps

Whether you use signed request or OAuth authorization, you can use SAML-based single sign-on (SSO) to provide your users with a seamless authentication flow. You can leverage Salesforce as an identity provider or as a service provider. SAML SSO enables you to give your users automatic authentication into your canvas app via SAML and authentication into Salesforce via the signed request.

SAML (Security Assertion Markup Language) is an XML-based standard for user authentication on the Web which allows the exchange of authorization data between two domains. With this feature you can create a canvas app that begins a standard SAML authentication flow when opened by a user. After this process completes, the user is authenticated into your Web application.

For canvas apps that use signed request authentication, two methods that are included in the Canvas SDK enable your canvas app to call into Salesforce to receive a new signed request directly or enable Salesforce to repost the signed request to your Web application endpoint. This results in a complete end-to-end authentication flow.

`refreshSignedRequest` Method

Returns a new signed request via a callback. After the SAML SSO process is completed, your app can call this method and receive a new signed request. This method is intended for developers who need to retrieve the signed request by using a more client-side JavaScript approach. (The Canvas SDK sends the signed request to your app.)

`repost` Method

Requests the parent window to initiate a POST to your canvas app and reloads the app page with a refreshed signed request. After the SAML SSO process is completed, your app can call this method and a new signed request is sent to your app via a POST. This method is for developers who want to retrieve the signed request using a more server-side approach. (Salesforce POSTs the signed request to your server.)

Note: The SAML Initiation Method `Identity Provider Initiated` is not supported for canvas apps on Lightning Platform sites.

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.

Note: Getting context data from the Canvas SDK is the most reliable way to identify users. Browsers may not send your canvas app cookies or other identifying data. See Alternatives to Cookies for User Tracking on page 51.

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:

- Canvas SDK
- Context
- Filtering CanvasRequest Context Data

Cross-Domain XHR

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

Important: To prepare your Canvas apps for the new web browser restrictions on third-party cookies, see How Third-Party Cookie Restrictions Impact Salesforce Canvas Apps in Salesforce Help.

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 on page 50
- Posting to a Chatter Feed on page 50

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

- Getting a List of Chatter Users

Posting to a Chatter Feed

SEE ALSO:

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

Alternatives to Cookies for User Tracking

When creating websites and applications, the HTML `<iframe>` element lets you display third-party content within the current page window. As modern browsers have increased privacy for end users, the iframe structure is becoming more scrutinized and restricted. Third-party applications exposed as Canvas apps are surfaced in the Salesforce user interface via an iframe. To avoid problems loading your Canvas apps in Salesforce, design applications that don't rely on cookies, session storage, or local storage to track users.

Cookies and other storage have traditionally been reliable ways to add application state on top of the state-less HTTP protocol. However, cookies and storage also allow applications to track users in ways that users don't suspect. Modern browsers are working to enhance privacy and security on the web. Applications written years ago may not behave the same way in modern browsers or when used in iframe containers by applications, such as Salesforce. In the past, content hosted in an iframe would access its own storage, as if it was hosted in a first-party context. Browsers now treat content hosted in an iframe as third-party applications.

Third-party applications exposed as Canvas apps work outside of Salesforce, but when exposed in Salesforce via an iframe, the content may not load properly because the content is served from a different domain. Browsers place tracking and user-privacy restrictions on these applications when they're used in Salesforce. For example:

- Cookies are not sent in page requests and XHRs.
- Session storage and local storage access are denied.
- JavaScript APIs, such as `alert()` and `confirm()`, are blocked.

To avoid problems late in development, test Canvas apps in a Salesforce container early. To design applications that don't rely on cookies, session storage, or local storage to track users, build your Canvas apps as single-page applications (SPAs).

Warning: One side effect of not using cookies or storage is that you may not be logged into your application in multiple tabs. If SPAs don't work for your development scenarios, ask your users to temporarily allow tracking for the browser where your Canvas app is used. Modifying settings in your web browser typically affects all websites, not just Salesforce. In Chrome and Firefox, you may add specific domains to always allow cookies, and can instruct your users to do the same. See this list of Salesforce's standard domains, and add any third-party applications domains.

Use the Signed Request for Authentication

SPAs work with Canvas and iframes. These applications don't use page-level navigation after their first page load. Instead, they dynamically rewrite the page as the user interacts, which means that the same JavaScript execution context is kept for the lifetime of the app. To authorize users within your application, use the `CanvasRequest` object. After the first request, store the authorization token for your app in JavaScript instead of in cookies or local storage. Authorize HTTPS requests using authorization headers through the `fetch()` or `XMLHttpRequest` standard web APIs.

See Signed Request Authentication for details on the `CanvasRequest` object.

Single-page applications can also request temporary access to cookies and other tracking data using the Storage Access API. This API relies on explicit user interaction to allow access to cookies. Access is granted until the application navigates, which is why a single-page application is preferred over multi-page applications.

Note: The storage access API is not available in all browsers, and behaves differently in each browser. See Storage Access API for details.

Stay Up-to-Date with Browser Changes

Different browsers behave differently and are actively changing their behaviors as privacy on the web evolves. Refer to each browser's guidelines for the most up-to-date information.

- Apple Safari
- Microsoft Edge
- Mozilla Firefox
- Google Chrome

SEE ALSO:

Salesforce Developers Blog: Prepare for the Google Chrome Privacy Sandbox Initiative

Knowledge Article: Understand How Google's Privacy Sandbox Initiative Impacts Salesforce

Resizing a Canvas App

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.

Attention: For resize functions to work with your canvas app, you must declare a DOCTYPE at the top of any HTML pages that are associated with the app. For example: `<!DOCTYPE html>`.

Automatically Resizing a Canvas App

Explicitly Resizing a Canvas App

Getting the Size of a Canvas App

Subscribing to Parent Events

Handling Orientation Changes in Your Canvas App

SEE ALSO:

Canvas SDK

Automatically Resizing a Canvas App

The following code examples show 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.

Note: In Mozilla® Firefox® and Microsoft® Internet Explorer®, the `autogrow` method might not resize the frame if the content size is reduced. In this case, you can use the `resize` method to specify the exact size that you want to change the frame to.

```
// 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

Handling Orientation Changes in Your Canvas App

The `orientation` event enables you to handle changes in orientation when your canvas app appears on a mobile device. After your canvas app subscribes to the event, the event fires whenever the parent window fires a `window.orientation` event. The event returns a payload that contains these values.

Value	Description
`clientHeight`	The height of the app in pixels, specific to the device on which the canvas app renders
`clientWidth`	The width of the canvas app in pixels, specific to the device on which the canvas app renders
`orientation`	Contains one of these values. • 0: landscape to portrait • 90: portrait to landscape • -90: portrait to landscape turned counterclockwise

The following code example shows how to subscribe to the `orientation` event.

```
// Capture the orientation event of the parent window.
Sfdc.canvas.client.subscribe(sr.client,
    {name : 'canvas.orientation',
    onData : function (event) {
        console.log("Parent's orientation: " + event.orientation +
                    "Canvas app height: " + event.clientHeight +
                    "Canvas app width: " + event. clientWidth);
        }} );
});
```

Note: The `orientation` event isn't supported on Windows phones.

SEE ALSO:

Resizing a Canvas App

Subscribing to Parent Events

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.

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 or Visualforce pages 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.

Canvas App Events Considerations

Creating a Canvas App Event

Subscribing to a Canvas App Event

Unsubscribing from a Canvas App Event

SEE ALSO:

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

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.

Using the Streaming API Event

Subscribing to a Streaming API Event

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
Streaming API Developer's Guide

Using the Streaming API Event

The 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 Streaming API Developer 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

Debugging in a Canvas App

You must use Chrome, Firefox, or Safari to follow these steps.

When using the Canvas SDK, you can enable debug mode to make troubleshooting easier.

1. With your canvas app open in your browser, open the console.
2. Type `Sfdc.canvas.console.enable();` and press return.
3. Refresh your browser.

Use data from this screen to debug canvas-related problems.

Using the <select> Tag in a Canvas App Tag in a Canvas App

If you use `HTML <select>` tags in your canvas web application and intend to make your canvas app available in the Salesforce mobile app, keep in mind that behavior can be unpredictable on some devices. Consider using a jQuery or CSS alternative to using `<select>`.

CHAPTER 5 Canvas Apps and Visualforce Pages

In this chapter ...

- Visualforce Page Code Examples
- Visualforce Considerations
- apex:canvasApp Component
- Using Events between a Visualforce Page and a Canvas App

In addition to standard canvas apps, 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.

Note: The Canvas framework includes methods that you can use to circumvent the navigational limitations of the iframe that contains a canvas app, without using Visualforce pages. These methods offer a streamlined alternative for controlling navigation to or from canvas apps in the Salesforce app. For details, see Salesforce Mobile App Navigation Methods for Use with Canvas Apps on page 89.

Known Limitations

Canvas apps embedded in Visualforce using `<apex:canvasApp>` serialize parameters as part of a redirect URL during load. If the combined size of parameters exceeds 8KB, this may cause the redirect to fail with a 500 Internal Server Error. This is due to a header size limit.

To avoid the header size limit issue:

- Pass only small tokens or ID's via parameters.
- Let your Canvas app fetch large payloads server-side.
- Consider using signedRequest mode, which avoids redirects and uses POSTS.

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 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. The table below lists the component attributes.

Pass parameters that contain non-alphanumeric characters such as quotation marks, apostrophes, and so on, as JavaScript-safe objects. To do this, write an Apex class that uses the methods of the Apex JSONGenerator class to build the JSON string. Call the Apex class from the `parameters` value:

```
<apex:page controller="JSONGeneratorSample">
 <apex:canvasApp developerName="mycanvas" parameters="{!generateJSON}" />
</apex:page>
```

Alternatively, you can also use the JSENCODE function to escape the strings:

```
<apex:page standardController="Account">
 <apex:canvasApp developerName="mycanvas" parameters="{!JSENCODE(Account.Description)}"
/>
</apex:page>
```

Attribute	Type	Description
`applicationName`	String	Name of the canvas app. Either `applicationName` or `developerName` is required.

Attribute	Type	Description
`border`	String	Width of the canvas app border, in pixels. If not specified, defaults to 0 pixels.
`canvasId`	String	Unique ID of the canvas app window. Use this attribute when targeting events to the canvas app.

`containerId`	String	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 `<apex:canvasApp>` component. These code examples show valid usage of the `<div>` container and the `containerId` attribute:
		`<apex:page>`
		`  <div id="container1"></div>`
		`  <apex:canvasApp applicationName="myApp"`
		`containerId="container1"/>`
		`</apex:page>`
		
		`<apex:page>`
		`  <div id="container1">`
		`  <apex:canvasApp`
		`applicationName="myApp"`
		`containerId="container1"/>`
		`  </div>`
		`</apex:page>`
		
		This code example shows invalid usage of the `<div>` container and the `containerId` attribute:
		`<apex:page>`
		`  <apex:canvasApp applicationName="myApp"`
		`containerId="container1"/>`
		`  <div id="container1">`
		`  </div>`
		`</apex:page>`

`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.
`entityFields`	String	Specifies the fields returned in the signed request Record object when the component appears on a Visualforce page placed on an object. If this attribute isn't specified or is blank, then only the Id field is returned. Valid attribute values include: • Comma-separated list of field names. For example, to return the Account Phone and Fax fields, the attribute would look like: `entityFields="Phone,Fax"` • Asterisk "*" to return all fields from the associated object.

Attribute	Type	Description
`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.
`maxHeight`	String	The maximum height of the Canvas app window in pixels. Defaults to 2000 px; 'infinite' is also a valid value.
`maxWidth`	String	The maximum width of the Canvas app window in pixels. Defaults to 1000 px; 'infinite' is also a valid value.
`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 an expression to a JavaScript JSON object literal. 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. Valid values are: • `auto` • `no` • `yes` If not specified, defaults to `no`. If this attribute contains an invalid value, it's treated the same as `no` to prevent browser errors.
`width`	String	Canvas app window width, in pixels. If not specified, defaults to 800 pixels.

Returning Fields in the Record Object
When you use the <apex:canvasApp> component to display a canvas app on a Visualforce page, and that page is associated with an object (placed on the page layout, for example), you can specify fields to be returned from the related object. To do this, use the entityFields attribute. You can use this attribute in one of the following ways.

SEE ALSO:

Canvas Apps and Visualforce Pages
Visualforce Developer's Guide
Visualforce Page Code Examples
Returning Fields in the Record Object

Returning Fields in the Record Object

When you use the <apex:canvasApp> component to display a canvas app on a Visualforce page, and that page is associated with an object (placed on the page layout, for example), you can specify fields to be returned from the related object. To do this, use the entityFields attribute. You can use this attribute in one of the following ways.

Return Specific Object Fields

You can return specific object fields in the Record object by setting the entityFields attribute to a comma-separated list of field names. If there's an invalid field name in the list, then that field name is ignored and the valid fields are returned in the Record object.

```
<apex:canvasApp applicationName="MyApp" entityFields="Phone,Fax,BillingCity"
containerId="canvasAppDiv"/>
```

returns a Record object that looks like this:

```
"record":{
    "attributes":{
        "type":"Account",
        "url":"/services/data/v66.0/sobjects/Account/001xx000003DGWiAAO"
        },
    "Id":"001xx000003DGWiAAO",
    "Phone":"(555) 555-5555",
    "Fax":"(555) 555-5555",
    "BillingCity":"Seattle"
    }
```

Return All Object Fields

You can return all object fields in the Record object by setting the entityFields attribute to the wildcard "*".

```
<apex:canvasApp applicationName="MyApp" entityFields="*" containerId="canvasAppDiv"/>
```

returns a Record object that looks like this:

```
"record":{
    "attributes":{
        "type":"Account",
        "url":"/services/data/v66.0/sobjects/Account/001xx000003DGWiAAO"
        },
```

```
    "Id":"001xx000003DGWiAAO",
    "IsDeleted":false,
    "MasterRecordId":null,
    "Name":"Edge Communications",
    "Type":"Customer - Channel",
    "ParentId":null,
    "BillingStreet":"123 Main Street",
    "BillingCity":"Seattle",
    "BillingState":"WA",
    "BillingPostalCode":"98121",
    "BillingCountry":"USA",
...
    }
```

Return the Id Field

If the `<apex:canvasApp>` component doesn't have the `entityFields` attribute or if the attribute is blank, then only the Id field is returned in the Record object.

```
<apex:canvasApp applicationName="MyApp" containerId="canvasAppDiv"/>
```

returns a Record object that looks like this:

```
"record":{
    "attributes":{
        "type":"Account",
        "url":"/services/data/v66.0/sobjects/Account/001xx000003DGWiAAO"
        },
    "Id":"001xx000003DGWiAAO"
    }
```

SEE ALSO:

- apex:canvasApp Component
- Record
- Controlling Canvas App Behavior

Using Events between a Visualforce Page and a Canvas App

Canvas provides methods that you can use to publish and subscribe to events between your canvas app and the parent Visualforce page. This process includes publishing events from your Visualforce page, listening for events on your Visualforce page, unsubscribing from events on your Visualforce page, and resizing the Visualforce page that contains the canvas app. This process for resizing a canvas app differs from the process where the canvas app resizes itself, which is described in Resizing a Canvas App. Full reference documentation for these methods can be found in the SDK and here.

- `publish`—Publishes an event from the Visualforce page that the canvas app can subscribe to. See Publishing a Canvas Event from a Visualforce Page.
- `resize`—Used by a Visualforce page to resize a canvas app iFrame. See Resizing a Canvas App in a Visualforce Page.
- `subscribe`—Used by a Visualforce page to subscribe to events that the canvas app might publish. See Subscribing to Events.
- `unsubscribe`—Used by a Visualforce page to unsubscribe to parent events that the canvas app might publish. See Unsubscribing from Events in a Visualforce Page.

The Controller.js object is needed for the Visualforce page to use these methods. Your script in the Visualforce page might look something like this:

```
<script
type="text/javascript"src="https://yourDomain.my.salesforce.com/canvas/sdk/js/30.0/controller.js">
```

Note: You can use Lightning Platform methods to smoothly integrate navigation between a canvas app and the Salesforce mobile app, without using Visualforce. The methods are events that reside in the JavaScript library within the Canvas framework. When you call one of the navigation methods from your canvas code, you send an event into Salesforce that reads the payload and directs the user to the specified destination. For more information, see Salesforce Mobile App Navigation Methods for Use with Canvas Apps on page 89.

Publishing a Canvas Event from a Visualforce Page
Resizing a Canvas App in a Visualforce Page
Subscribing to Events
Unsubscribing from Events in a Visualforce Page

SEE ALSO:

Publishing a Canvas Event from a Visualforce Page
Resizing a Canvas App in a Visualforce Page
Subscribing to Events
Unsubscribing from Events in a Visualforce Page
apex:canvasApp Component

Publishing a Canvas Event from a Visualforce Page

The following code example shows how to call the `publish` method to publish the `myevent` event from a Visualforce page. Any canvas app that subscribes to this event will receive the event.

```
    // Target all canvas apps.
    Sfdc.canvas.controller.publish({name : 'mynamespace.myevent',
                                    payload : {}});
```

The following code example shows how to call the `publish` method to publish an event to a specified canvas app from the Visualforce page.

If an event is published to specific canvas apps, even if other canvas apps on a Visualforce page are subscribed to it, only the canvas apps specified will receive that event. Using this code example, if the Visualforce page contained canvas apps app1, app2, and app3, and they all subscribed to `myevent`, only app1 would receive the event.

```
    // Target a specific canvas app
    // where "app1" is the canvasId specified in the canvas component.
    // For example: <apex:canvasApp canvasId="app1">
    Sfdc.canvas.controller.publish({name : 'mynamespace.myevent',
                                    payload : {},
                                    target : {canvas : 'app1'}});
```

In the `target` parameter, you can specify multiple canvas apps by passing in an array of canvas apps: `target : [{canvas : 'app1'}, {canvas: 'app2'}]`.

SEE ALSO:

- Using Events between a Visualforce Page and a Canvas App
- Resizing a Canvas App in a Visualforce Page
- Subscribing to Events
- Unsubscribing from Events in a Visualforce Page
- apex:canvasApp Component

Resizing a Canvas App in a Visualforce Page

The following code example shows how to call the `resize` method to explicitly set the height and width on a specific canvas app within a Visualforce page.

```
    // Set the height and width explicitly and target a canvas app
    // Where 'mycanvas0' is the canvasId on the canvas component
    // <apex:canvasApp canvasId="mycanvas0"/>
    var target = {canvas : "mycanvas0"};
    Sfdc.canvas.controller.resize( {height : "1000px", width : "900px"}, target);
```

The following code example shows how to call the `resize` method to set the height on all canvas apps within a Visualforce page.

```
    //Set only the height on all canvas apps
    Sfdc.canvas.controller.resize( {height : "1000px"});
```

SEE ALSO:

- Using Events between a Visualforce Page and a Canvas App
- Publishing a Canvas Event from a Visualforce Page
- Subscribing to Events
- Unsubscribing from Events in a Visualforce Page

Subscribing to Events

The following code example shows how to call the `subscribe` method within a Visualforce page to subscribe to specified events published from a canvas app.

```
// Subscribe to a single event.
Sfdc.canvas.controller.subscribe({name : 'mynamespace.myevent0',
                                  onData : function (e) {}});

// Subscribe to multiple events in a single call.
Sfdc.canvas.controller.subscribe([
    {name : 'mynamespace.myevent1', onData : function(e) {}},
```

```
    {name : 'mynamespace.myevent2', onData : function(e) {}}
]);
```

SEE ALSO:

- Using Events between a Visualforce Page and a Canvas App
- Publishing a Canvas Event from a Visualforce Page
- Resizing a Canvas App in a Visualforce Page
- Unsubscribing from Events in a Visualforce Page
- Implementing Canvas App Events

Unsubscribing from Events in a Visualforce Page

The following code example shows how to call the `unsubscribe` method within a Visualforce page to unsubscribe from two events.

```
    Sfdc.canvas.controller.unsubscribe(['mynamespace.myevent2', 'mynamespace.myevent2']);
```

SEE ALSO:

- Using Events between a Visualforce Page and a Canvas App
- Publishing a Canvas Event from a Visualforce Page
- Resizing a Canvas App in a Visualforce Page
- Subscribing to Events

CHAPTER 6 Lightning Component Code Examples

The following examples show how to reference a `<force:canvasApp>` component using `applicationName`, `developerName`, and `namespacePrefix`.

`<force:canvasApp>` represents a canvas app that's embedded in a Lightning component. It provides attributes and event handlers that you can use to customize the canvas app. `Sfdc.canvas` is not supported for use with Lightning components. For more information, see the `<force:canvasApp>` reference documentation.

Note: When possible, use `developerName` instead of `applicationName`. The `developerName` (also called the API name) is the permanent name assigned to a Connected App when you create a canvas app. You can change the `applicationName`, which can break Aura components that use an outdated name.

Object Detail Page

This example displays a canvas app on an Account page using the `applicationName` and `namespacePrefix` attributes of `<force:canvasApp>`.

In `myCanvasApp.cmp`, we first define an Aura attribute called `canvasParameters`. We use this attribute to pass a `recordId` into the canvas app. Next, we define an `init` handler, which invokes the action method `doInit` when the component is initialized. In `<force:canvasApp>`, we set the size of the canvas app in pixels.

```
<!-- myCanvasApp.cmp file -->
<aura:component implements="flexipage:availableForAllPageTypes,
force:hasRecordId">
    <aura:attribute name="canvasParameters" type="string" />
    <aura:handler name="init" value="{!this}" action="{!c.doInit}"/>


    <aura:if isTrue="{!v.canvasParameters}">
        <force:canvasApp developerName="Test_Inline_Aura"
            namespacePrefix="testorg"
            height="400px" width="750px"
            parameters="{!v.canvasParameters}"/>
    </aura:if>

</aura:component>
```

In `myCanvasAppController.js`, the `doInit` method sets the `recordId` in the `canvasParameters` attribute.

```
// myCanvasAppController.js
({
    doInit : function(cmp, evt, helper) {
        var recordId = cmp.get("v.recordId");
        cmp.set("v.canvasParameters", JSON.stringify({
            recordId: recordId
        }));
    }
})
```

Important: When your code has dependencies that require a prior action to complete, don't call dependent actions until the earlier action completes. For example, render a dependent element conditionally, based on the result of the earlier action being available. Or call the dependent action from the earlier action's callback function. This ensures that the dependent call isn't made until after the earlier call completes. See "Hidden Dependencies in a Canvas Component" in Manage Synchronous Action Dependencies in the *Lightning Aura Components Developer Guide*.

Standard Page

This example displays a canvas app in an Aura component using the `developerName` and `namespacePrefix` attributes of `<apex:canvasApp>`. The code specifies the size of the canvas app to be 1000 pixels high and 800 pixels wide. It passes three custom parameters to the canvas app.

```
<aura:component>

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

</aura:component>
```

SEE ALSO:

Lightning Aura Components Developer Guide: Manage Synchronous Action Dependencies

CHAPTER 7 Canvas Apps in a Page Layout or a Mobile Card

In this chapter ...

- Where Canvas Apps Appear in a Page Layout
- Add a Canvas App to a Page Layout

You can add a canvas app to a page layout for any standard or custom object. For the Canvas Apps category to appear in the palette when you edit a page layout, you must set the canvas app location to Layouts and Mobile Cards when you create the canvas app in the Salesforce application.

SEE ALSO:

Add a Canvas App to a Page Layout

Where Canvas Apps Appear in a Page Layout

Where Canvas Apps Appear in a Page Layout

Depending on where you place the canvas app on the page layout, the canvas app might appear in the full Salesforce site or in the Salesforce mobile app.

This table describes where the canvas app appears when you add it to the page layout.

If the canvas app is added to the page layout in ...	The canvas app appears in the full Salesforce site?	The canvas app appears in the Salesforce mobile app?	Where the canvas app appears in the Salesforce mobile app
Any section (other than the Mobile Cards section)	Yes	Yes	Record detail page
Mobile Cards section	No	Yes	Mobile card

SEE ALSO:

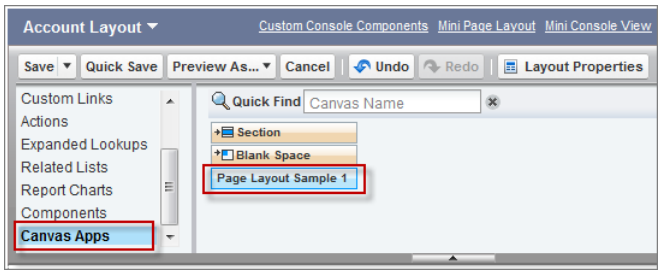
Canvas Apps in a Page Layout or a Mobile Card

Add a Canvas App to a Page Layout

Add a Canvas App to a Page Layout

Add a canvas app to the page layout for an Account.

1. To create a canvas app, follow the steps in Create the App.
2. From Setup, enter *Apps* in the Quick Find box, then select **Manage Connected Apps**.
3. In the Connected Apps list view, click **Edit** next to the app that you created.
4. In the Canvas Apps Settings section, in the Locations field, select **Layouts and Mobile Cards**.
5. Click **Save**.
6. From Setup, click **Object Manager**, then select **Account**.
7. On the Account object management page, click **Page Layouts**, then click **Account Layout**.



The Canvas Apps category is in the palette and the canvas app because you set the location of the canvas app to Layouts and Mobile Cards.

8. Drag the canvas app element to where you want it to appear in the page layout, and then click **Save**.
   - If you add the canvas app to any section other than the Mobile Cards section, the canvas app appears in the page layout in the full Salesforce site or in the record detail page in the Salesforce mobile app.

- If you add the canvas app to the Mobile Cards section, the canvas app appears only in the mobile card.

9. To see your changes, click the **Accounts** tab, and then click an account.

You can modify additional properties of the canvas app in the page layout by clicking the wrench icon .

- `Width (in pixels or %)`—The width of the canvas app; defaults to 100%. Changes to the width of the canvas app appear in both the full Salesforce site and the Salesforce mobile app. However, canvas apps are displayed in a single-column layout, so we recommend that you leave the width at 100%.
- `Height (in pixels)`—The height of the canvas app; defaults to 200 pixels. Changes to this field appear to the user for canvas apps that appear in both the full Salesforce site and the Salesforce mobile app.
- `Show scrollbars`—Whether scrollbars are displayed on the canvas app iFrame. Changes to this field render for canvas apps that appear in the full Salesforce site but not for canvas apps that appear in the Salesforce mobile app.
- `Show label`—Whether to display the page layout section label. Changes to this field appear to the user for canvas apps that appear in both the full Salesforce site and in the Salesforce mobile app.

SEE ALSO:

Canvas Apps in a Page Layout or a Mobile Card
Where Canvas Apps Appear in a Page Layout
Salesforce Help: Find Object Management Settings

CHAPTER 8 Canvas Apps in the Publisher

In this chapter ...

- Set Canvas App Location and Create the Action
- Create the Action Manually
- Canvas SDK Publisher Events
- Publisher Context Considerations
- Publisher Canvas App Access Considerations

Canvas enables you to expose your canvas apps as quick 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 custom actions of a canvas app. These actions can then integrate with the 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 quick 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 quick 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

Set Canvas App Location and Create the Action

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

1. In Salesforce, from Setup, enter *Apps* in the Quick Find box, then select **App Manager**.
2. In the Connected Apps related list, click **New**. Fill out the basic fields for your canvas app. See Use a Connected App.
3. In the Canvas App settings, select **Canvas** and in the Locations field, select Publisher. You must select this location for your canvas app to appear in both the publisher in the full Salesforce site and in the action bar in the Salesforce mobile app.
4. Select the Create Actions Automatically field. This creates a quick action for the canvas app.
5. To hide the publisher header, which contains the "What are you working on?" text, select Hide Publisher Header. This hides the header in both the full Salesforce site and in the Salesforce mobile app.
6. To hide the publisher **Share** button from users in both the full Salesforce site and in the Salesforce mobile app, select Hide Publisher Share Button. This checkbox is enabled only if Hide Publisher Header is selected.

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

SEE ALSO:

Where Canvas Apps Appear
Create the Action Manually
Canvas Apps in the Publisher

Create the Action Manually

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.

1. From Setup, enter *Actions* in the Quick Find box, then select **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 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 quick 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 quick 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

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.

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
`publisher.clearPanelState`	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.
`publisher.failure`	Fired by the publisher when an error condition is encountered such as when invalid data has been submitted. For example: • The text in the feed is too long • The canvas app you're attempting to publish to the feed doesn't exist • The canvas app URL is invalid The canvas app should listen for this event and alert the user that an error occurred and the post didn't get created.
`publisher.getPayload`	Fired by the publisher when the Share button is clicked. The payload contains information such as the text entered into the `What are you working on?` field and who the feed item is being shared with.
`publisher.setupPanel`	Fired by the publisher when the Chatter feed page is initially loaded.
`publisher.setPayload`	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 `publisher.getPayload` and contains information about the feed item you're trying to create. You can create three feed item types: • `TextPost` • `LinkPost` • `CanvasPost`

Field	Description
`publisher.setValidForSubmit`	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. This code snippet enables the Share button: `$$.client.publish(sr.client, {name : 'publisher.setValidForSubmit', payload : true});`
`publisher.showPanel`	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.
`publisher.success`	Fired by the publisher after the Share button is clicked and data is successfully submitted.

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 Manually
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:

- 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 Canvas SDK to change the values of your canvas app up to the `maxHeight` and `maxWidth`.

SEE ALSO:

Create the Action Manually
Environment
Dimensions
Resizing a Canvas App
Publisher Canvas App Access Considerations
Canvas Apps in the Chatter Feed

Publisher Canvas App Access Considerations

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

- 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 9 Canvas Apps in the Chatter Feed

In this chapter ...

- Chatter Feed Context Considerations
- Chatter Feed Canvas App Access Considerations

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 Connect REST API or Connect in Apex.
- 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. In Salesforce Classic, 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. In Lightning Experience, when the user clicks on the link or the description, the user is redirected to a dedicated canvas app page.

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

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:

- 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 Canvas SDK to change the values of your canvas app up to the `maxHeight` and `maxWidth`.

SEE ALSO:

Canvas Apps in the Chatter Feed
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:

- 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 Connect REST API or Connect in Apex, 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 10 Canvas in the Salesforce Mobile App

In this chapter ...

- Set Canvas App Location and Add it to the Navigation Menu
- Salesforce Mobile App Context Considerations
- Salesforce Mobile App Access Considerations
- Salesforce Mobile App Custom Icons
- Salesforce Mobile App Navigation Methods for Use with Canvas Apps

Canvas enables you to expose your canvas apps in the Salesforce mobile app. The Salesforce mobile 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 Canvas apps.

Developers can use canvas apps in the Salesforce mobile app to:

- Expose a canvas app as an action. An icon indicates a canvas app. You can use either the default puzzle icon or upload a custom icon for the related action.
- Post to the feed from a canvas app in Salesforce or through Connect REST API or Connect in Apex.
- Display a canvas app inside a feed item from within the Salesforce mobile app. 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.
- Add a canvas app as an option in the navigation menu. An icon indicates a canvas app. You can use the default puzzle icon or provide a custom icon URL in the connected app settings of your canvas app.

  Note: Canvas apps don't appear in the app navigation menu in Salesforce for Android. To see canvas apps in the navigation menu, log in to Salesforce mobile web.

For example, you might have a canvas app that warehouse employees use to process orders on a mobile device. You can create an action that accesses the app from the 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 Salesforce offers doesn't impact or limit existing functionality.

SEE ALSO:

Canvas Apps in the Publisher

Canvas Apps in the Chatter Feed

Salesforce Mobile App Access Considerations

Salesforce Mobile App Context Considerations

Salesforce Mobile App Custom Icons

Set Canvas App Location and Add it to the Navigation Menu

To add a canvas app to appear in the Salesforce mobile app navigation menu, you must set the location and add it to the mobile navigation.

1. In Salesforce, from Setup, enter *Apps* in the Quick Find box, then select **Apps**.
2. In the Connected Apps related list, click the app you want to make available in the navigation pane and click **Edit**.
3. In the Canvas App Settings section, select **Canvas**.
4. In the Locations field, select **Mobile Nav** and click **Add**.
5. Click **Save**.
6. For the canvas app to appear in the navigation menu you must add it to the mobile navigation. From Setup, enter *Navigation* in the Quick Find box, then select **Salesforce Navigation**.
7. From the available menu items, select your app and click **Add**.
8. Click **Save**.

Salesforce Mobile App Context Considerations

Keep these considerations in mind when you display a canvas app inside of the Salesforce mobile app.

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 Salesforce mobile app 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 Salesforce is designed for mobile, the sizes we provide for it are different than the ones we provide for the full Salesforce sites.
- To create a single-finger touch scrolling experience:
  - Ensure that the outermost div elements 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:

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

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

- In the navigation menu, the default puzzle icon size is 60 pixels by 60 pixels.

SEE ALSO:

Canvas in the Salesforce Mobile App
Salesforce Mobile App Access Considerations
Resizing a Canvas App
Salesforce Mobile App Custom Icons

Salesforce Mobile App Access Considerations

Keep these considerations in mind when modifying a canvas app that appears in the Salesforce mobile app.

By necessity, the Salesforce mobile app layout is different than the full Salesforce site layout that you're used to. Remember the following when creating a canvas app for use in the Salesforce mobile app.

- Because 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 accessing your canvas app as an action, if the canvas app uses the "What are you working on" header, the header is fixed at the top of the page and your canvas app scroll area falls below the header 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. Use the orientation event to handle changes in device orientation.
- The heights used in the Salesforce mobile app and the full Salesforce site are different. Use the Dimensions object in the signed request to render your actions correctly in the publisher.
- In the action menu, long 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 the Salesforce mobile app, tap .
- When you view the action bar or Feed, the default Canvas puzzle icon displays for canvas apps. You can override this default action icon with an image you provide.
- When a canvas app appears as an option in the navigation menu, the default Canvas puzzle icon is used. You can customize the icon in the connected app settings of your canvas app.
- Canvas apps aren't available when the user accesses Salesforce offline.

SEE ALSO:

Canvas in the Salesforce Mobile App
Salesforce Mobile App Context Considerations
Salesforce Mobile App Custom Icons
Dimensions
Handling Orientation Changes in Your Canvas App
Alternatives to Cookies for User Tracking

Salesforce Mobile App Custom Icons

Custom icons help distinguish your app in the Salesforce mobile app. If you don't customize the icon for your app, you'll get the default puzzle-piece icon.

You can customize the icon that is used in the Salesforce navigation menu. You set this icon in the Icon URL entry in the Basic Information section of the connected app settings for your canvas app. From Setup, enter *Apps* in the Quick Find box, then select **Apps** and click **Edit** for your connected app. The icon URL must be a secure HTTPS URL that points to an icon image file. The image file must be in the GIF, JPG, or PNG file format. For the Salesforce navigation menu, the icon cannot be larger than 60 pixels high by 60 pixels wide.

The custom icon that is used in the Salesforce navigation menu is also used in the Chatter tab and the Canvas App Previewer. If your canvas app will be shown in the navigation menu, we recommend that you use a 60x60 pixel size icon and let Salesforce automatically resize the icon to the smaller size that is needed for the Chatter tab and the Canvas App Previewer.

You can also customize the icon that is used in the Salesforce action bar and action menu for a canvas app. The action bar uses the custom icon set for the action that accesses the canvas app, not the custom icon that is associated with the connected app. You set the action icons by uploading a static resource file for your custom icon and then using this static resource as the icon for the global action. The static resource icon file should be in the PNG format, with a size of 120 pixels high by 120 pixels wide. See "Custom Icon Guidelines and Best Practices" in the Salesforce Mobile App Developer Guide for more guidelines on custom action icons.

SEE ALSO:

- Canvas in the Salesforce Mobile App
- Salesforce Mobile App Context Considerations

Salesforce Mobile App Navigation Methods for Use with Canvas Apps

The Canvas framework includes methods that you can use to smoothly integrate navigation between a canvas app and the Salesforce mobile app, without needing to use Visualforce.

To circumvent the navigational limitations of the iframe that contains a canvas app, use Canvas methods. These methods offer a simpler alternative to Visualforce pages for controlling navigation to or from canvas apps in the Salesforce app.

For example, your canvas app code can call Salesforce "create record" logic to navigate to the page where the **Create Account** button resides. Clicking the button triggers a navigation method to go to the record creation page, which is outside of the canvas app.

These methods within the Canvas framework are events that reside in the JavaScript library. When you call one of the navigation methods from your canvas code, you send an event into Salesforce that reads the payload and directs the user to the specified destination.

Calling the methods from your canvas app is slightly different than calling functions from a Visualforce page, because the methods are proxied to the Salesforce container through the Canvas cross-domain API.

Reference the navigation method as an event variable, with name and payload. For example:

```
var event = {name:"s1.createRecord", payload: {entityName: "Account", recordTypeId:
"00h300000001234"}};
```

The following table shows the name, payload, and purpose of the navigation methods in Salesforce.

Name	Payload	Description
`s1.back`	`{"refresh" : true}`	Navigates to the previous state that's saved in the `sforce.one` history. It's equivalent to clicking a browser's Back button. `refresh` is optional. By default, the page doesn't refresh. Pass `true` to refresh the page if possible.
`s1.navigateToSObject`	`{"recordId" : "001XXXXXXXXXXX", "view" : "chatter"}`	Navigates to an sObject record, specified by a 15-character or 18-character `recordId`. This record "home" has several views, which in the Salesforce app are available as slides that the user can swipe between. `view` is optional and defaults to `detail`. `view` specifies the slide within record home to display initially. Record IDs corresponding to ContentNote SObjects aren't supported. The possible values are: • `detail`: the record detail slide • `chatter`: the Chatter slide • `related`: the view of related slide
`s1.navigateToURL`	`{"url" : "https://salesforce.com/apex/v.apexp", "isredirect" : true}`	Navigates to the specified URL. Relative and absolute URLs are supported. Relative URLs are relative to the Lightning domain, and retain navigation history. External URLs—that is, URLs that are outside the Lightning domain—open in a separate browser window. Note: Depending on the user's device platform, device settings, version of Salesforce, and authentication requirements for the external URL being opened, the separate browser window might require authentication or reauthentication. Use relative URLs to navigate to different screens within your app. Use external URLs to allow the user to access a different site or app, where those actions don't need to be preserved in your app. To return the user to your app, the separate window that's opened by an external URL must be closed when the user is finished with the other app. The new window has a separate history from your app, and this history is discarded when the window is closed. Discarding the history of a new window prevents the user from clicking a Back button to go back to your app; the user must close the new window. `mailto:`, `tel:`, `geo:`, and other URL schemes are supported for launching external apps and attempt to "do the right thing." However, support varies by mobile platform and device. `mailto:` and `tel:` are reliable, but we recommend that you test any other URLs on a range of expected devices. `isredirect` is optional and defaults to `false`. Set it to `true` to indicate that the new URL should replace the current one in the navigation history. Be careful when using `navigateToURL` within the `onClick` handler of an `<apex:commandButton>` or any `<button type="submit">` or `<input type="submit">`. Even if `isredirect=true`, the default click action of the command button is a form post. In this scenario, the command button performs a form post and a `navigateToURL` action, requiring the

Name	Payload	Description
		user to click the back button twice to navigate to the previous page. To prevent the default click action, configure the `onClick` handler to either call `event.preventDefault()` or return `false`. URLs corresponding to ContentNote SObjects aren't supported.
`s1.navigateToFeed`	`{"subjectId" : "001XXXXXXXXXXX", "type" : "NEWS"}`	Navigates to the feed of the specified `type`, scoped to the `subjectId`. For some feed `types`, the `subjectId` is required but ignored. For those feed `types`, pass the current user's ID as the `subjectId`. The possible values for `type` are: • `BOOKMARKS`: Contains all feed items saved as bookmarks by the context user. Pass the current user's ID as the `subjectId`. • `COMPANY`: Contains all feed items except feed items of type `TrackedChange`. To see the feed item, the user must have sharing access to its parent. Pass the current user's ID as the `subjectId`. • `FILES`: Contains all feed items that contain files posted by people or groups that the context user follows. Pass the current user's ID as the `subjectId`. • `GROUPS`: Contains all feed items from all groups the context user either owns or is a member of. Pass the current user's ID as the `subjectId`. • `NEWS`: Contains all updates for people the context user follows, groups the user is a member of, and files and records the user is following. Contains all updates for records whose parent is the context user. Pass the current user's ID as the `subjectId`. • `PEOPLE`: Contains all feed items posted by all people the context user follows. Pass the current user's ID as the `subjectId`. • `RECORD`: Contains all feed items whose parent is a specified record, which could be a group, user, object, file, or any other standard or custom object. When the record is a group, the feed also contains feed items that mention the group. When the record is a user, the feed contains only feed items on that user. You can get another user's record feed. Pass the record's ID as the `subjectId`. • `TO`: Contains all feed items with mentions of the context user. Contains feed items the context user commented on and feed items created by the context user that are commented on. Pass the current user's ID as the `subjectId`. • `TOPICS`: Contains all feed items that include the specified topic. Pass the topic's ID as the `subjectId`. This value is supported in Salesforce for mobile web only. Topics aren't available in Salesforce for iOS or Salesforce for Android.
`s1.navigateToFeedItemDetail`	`{"feedItemId" : "001XXXXXXXXXXX"}`	Navigates to the specific feed item, `feedItemId`, and any associated comments.

Name	Payload	Description
`s1.navigateToRelatedList`	`{"relatedListId" : "001XXXXXXXXXXX", "parentRecordId" : "001XXXXXXXXXXX" }`	Navigates to a related list for the `parentRecordId`. For example, to display a related list for a Warehouse object, the `parentRecordId` is `Warehouse__c.Id`. `relatedListId` is the API name or ID of the related list to display.
`s1.navigateToList`	`{"listViewId" : "001XXXXXXXXXXX", "listViewName" : "myListView", "scope" : "scope"}`	Navigates to the list view that's specified by the `listViewId`, which is the ID of the list view to be displayed. `listViewName` sets the title for the list view. It doesn't need to match the actual name that's saved for the list view. To use the saved name, set `listViewName` to null. Set `scope` to the name of the sObject in the view, for example, "Account" or "MyObject__c".
`s1.createRecord`	`{"entityName" : "MyObject__c", "recordTypeId" : "001XXXXXXXXXXX"}`	Opens the page to create a record for the specified `entityName`, for example, "Account" or "MyObject__c". `recordTypeId` is optional and specifies the record type for the created object. Calling `createRecord` without providing a `recordTypeId` may result in an error.
`s1.editRecord`	`{"recordId" : "001XXXXXXXXXXX"}`	Opens the page to edit the record specified by `recordId`.

For information about navigation methods for use with Visualforce, see the Salesforce Mobile App Developer's Guide.

SEE ALSO:

- Visualforce Considerations
- Canvas in the Salesforce Mobile App

CHAPTER 11 Customizing Your App Lifecycle

In this chapter ...

- Creating a CanvasLifecycleHandler
- Associating Your CanvasLifecycleHandler with Your App
- Filtering CanvasRequest Context Data
- Controlling Canvas App Behavior
- Presenting User Error Messages
- Testing Your CanvasLifecycleHandler Implementation
- Distributing Your CanvasLifecycleHandler Class

By providing a custom Apex class, you can control the context information that's sent to your canvas app and add custom behavior when your app is rendered.

Salesforce provides several Apex interfaces and classes in the Canvas namespace that provide additional control over the canvas app lifecycle. You can use the Canvas namespace and the CanvasLifecycleHandler interface to:

- Control what sections of the CanvasRequest Context data get sent to your app. You can, for example, require that Salesforce never send the Organization information in the CanvasRequest data. Excluding sections of context data improve performance by reducing the amount of data that's sent in the request and remove the need to process the organization data in your app.
- Retrieve application context data when the app is rendered and alter the behavior of your app accordingly. You can obtain the application version and, depending on the version, change how the app runs.
- Modify some of the context data, such as the canvas app URL, custom parameters, or the list of object fields that are returned in the Record data when the app is rendered.
- Present a well-formed error message to the user in the Salesforce UI if something goes wrong.

Salesforce also provides a Test class in the Canvas namespace that you can use to create test context data and verify the behavior of your lifecycle handler without having to run your app.

Reference documentation for the Apex Canvas namespace is provided in the Apex Code Developer Guide.

Creating a CanvasLifecycleHandler

You can control your app lifecycle by providing an implementation of the Canvas.CanvasLifecycleHandler Apex interface that Salesforce can use.

The Apex Canvas.CanvasLifecycleHandler interface provides methods and callbacks for customizing app lifecycle behavior. Salesforce will use your implementation at runtime to let you run custom code. Use the following steps to create an implementation of the Canvas.CanvasLifecycleHandler interface.

1. From Setup, enter *Apex Classes* in the Quick Find box, then select **Apex Classes**.
2. Click **New** to create a Apex class.
3. Create an Apex class that implements the Canvas.CanvasLifecycleHandler interface. You must implement the `excludeContextTypes()` and `onRender()` methods. Here's a template example:

```
public class MyCanvasLifecycleHandler
implements Canvas.CanvasLifecycleHandler {

    public Set<Canvas.ContextTypeEnum> excludeContextTypes(){
        Set<Canvas.ContextTypeEnum> excluded = new Set<Canvas.ContextTypeEnum>();

        // Code goes here to add items to excluded list
        // that should be excluded from Context data

        return excluded;
    }

    public void onRender(Canvas.RenderContext renderContext) {

        // Code goes here to customize behavior when the app is rendered

    }
}
```

4. After you've finished adding your code, save the Apex class.
5. Optionally test your implementation by using the Canvas.Test class.
6. To let Salesforce know which implementation to use for your app, associate your Apex class with your app.

SEE ALSO:

Testing Your CanvasLifecycleHandler Implementation

Associating Your CanvasLifecycleHandler with Your App

Associating Your CanvasLifecycleHandler with Your App

After you've created an Apex implementation class for CanvasLifecycleHandler, you need to associate it with your canvas app by adding the class name to your canvas app configuration settings.

Use the following steps to associate a Canvas.CanvasLifecycleHandler implementation with your canvas app.

1. From Setup, enter *Apps* in the Quick Find box, then select **Apps**.
2. Under Connected Apps, select the canvas app that you need to modify and click **Edit**.

3. On the connected app edit page, under Canvas App Settings, add the Apex CanvasLifecycleHandler class name that you created into the `Lifecycle Class` field.

Filtering CanvasRequest Context Data

To filter out parts of the CanvasRequest Context data that gets sent to your canvas app, you need to provide an implementation of `Canvas.CanvasLifecycleHandler.excludeContextTypes()`.

Salesforce calls your implementation of `excludeContextTypes()` to determine what context information is sent to your app. Your implementation returns a set of context data to exclude or returns null (or an empty set) if you need to receive all context data. The set consists of values from the Canvas.ContextTypeEnum Apex enum type.

The enum values are used as follows::

- ORGANIZATION: Exclude context information about the organization in which the canvas app is running.
- RECORD_DETAIL: Exclude context information about the object record on which the canvas app appears.
- USER: Exclude context information about the current user.

Here's an example `excludeContextTypes()` implementation that excludes the Organization and User context data:

```
public Set<Canvas.ContextTypeEnum> excludeContextTypes(){
    Set<Canvas.ContextTypeEnum> excluded = new Set<Canvas.ContextTypeEnum>();

    // Exclude the Organization and User context data
    excluded.add(Canvas.ContextTypeEnum.ORGANIZATION);
    excluded.add(Canvas.ContextTypeEnum.USER);

    return excluded;
}
```

When your canvas app receives CanvasRequest Context data, the JSON data does not include the "organization" and "user" sections.

Reference documentation for the Apex Canvas.ContextTypeEnum is provided in the Apex Code Developer Guide.

Controlling Canvas App Behavior

To modify the default behavior of the signed request, you need to provide an Apex class that implements `Canvas.CanvasLifecycleHandler.onRender()` and associate this class with your canvas app. In your `onRender()` implementation, you can control app behavior with custom code.

Salesforce calls your implementation of `onRender()` just before your app is rendered. Current context information is passed to this method in the Canvas.RenderContext parameter.

In your `onRender()` implementation, you can retrieve the following context information.

- Application context data, such as the canvas app name, URL, version, and namespace.
- Environment context data, such as the display location and sublocation, object field names, and custom parameters.

You can set the following context information.

- The portion of the canvas app URL after the app domain.
- The list of object fields for which Salesforce will return Record context data if the canvas app appears on an object page. One way a canvas app can appear on an object page is if the canvas app appears on a Visualforce page through the use of the `<apex:canvasApp>` component and that Visualforce page is associated with an object.

- The custom parameters that are passed to the canvas app.

You can also use Canvas.CanvasRenderException to present an error message to the user in the Salesforce by throwing a Canvas.CanvasRenderException.

Here's an example `onRender()` implementation that:

- Checks the app version information and, if the version is unsupported, throws a CanvasRenderException.
- Overrides the current canvas app URL, appending '/alternatePath' to the domain portion of the original URL.
- Sets the list of object fields to include Name, BillingAddress, and YearStarted, anticipating that the canvas app will appear on the Account page.
- Overrides the set of custom parameters by adding a new 'newCustomParam' parameter. Note that the current set of parameters is first retrieved and cached locally. The new parameter is added to the cached list to ensure that you don't lose the current set of custom parameters when you call `setParametersAsJSON()`.

```
public void onRender(Canvas.RenderContext renderContext) {

    // Get the Application and Environment context from the RenderContext
    Canvas.ApplicationContext app = renderContext.getApplicationContext();
    Canvas.EnvironmentContext env = renderContext.getEnvironmentContext();

    // Check the application version
    Double currentVersion = Double.valueOf(app.getVersion());
    if (currentVersion <= 5){
        // Versions lower than 5 are no longer supported in this example
        throw new Canvas.CanvasRenderException('Error: Versions earlier than 5 are no
longer supported.');
    }

    // Override app URL, replacing portion after domain with '/alternatePath'
    app.setCanvasUrlPath('/alternatePath');

    // Add Name, BillingAddress and YearStarted to fields
    // (assumes we'll run from a component on the Account detail page)
    Set<String> fields = new Set<String>{'Name','BillingAddress','YearStarted'};
    env.addEntityFields(fields);

    // Add a new custom param to the set of custom params
    // First, get the current custom params
    Map<String, Object> previousParams =
        (Map<String, Object>) JSON.deserializeUntyped(env.getParametersAsJSON());
    // Add a 'newCustomParam' to our Map
    previousParams.put('newCustomParam','newValue');
    // Now, replace the parameters
    env.setParametersAsJSON(JSON.serialize(previousParams));
}
```

Presenting User Error Messages

You can use Canvas.CanvasRenderException to display error messages to the user.

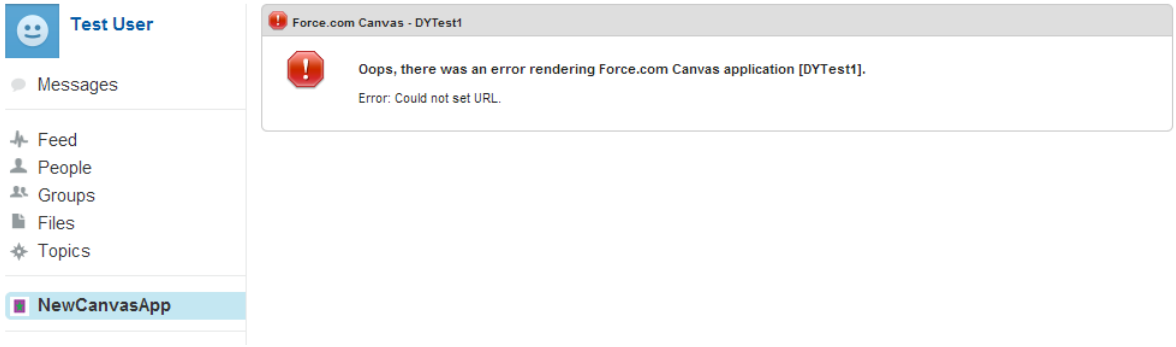
If you throw a CanvasRenderException from your `onRender()` implementation, the error message in that exception will be displayed to the user in the Salesforce UI. This exception will be managed only within the `onRender()` method.

CanvasRenderException supports all the standard Apex exception class methods and constructors. See "Exception Class and Build-In Exceptions" in the Apex Code Developer Guide for more information on Apex exceptions.

Here's an example `onRender()` implementation that throws a Canvas.CanvasRenderException. Salesforce uses this exception and displays "Error: Could not set URL" as the user error message.

```
public void onRender(Canvas.RenderContext renderContext) {
    // Code omitted that determines if the error condition occured
    ...
    throw new Canvas.CanvasRenderException('Error: Could not set URL.');
}
```

If the canvas app is run from the Salesforce Chatter tab, the error message resembles:



Testing Your CanvasLifecycleHandler Implementation

You can use the Canvas.Test class to test your `Canvas.CanvasLifecycleHandler.onRender()` implementation without having to run your canvas app.

Use Canvas.Test to create a test Canvas.RenderContext with mock application and environment context data. Call `Canvas.Test.testCanvasLifecycle()` with the mock RenderContext and your CanvasLifecycleHandler implementation to verify that your CanvasLifecycleHandler is being invoked correctly.

Use `Canvas.Test.mockRenderContext()` to create a mock RenderContext. You can provide initial mock application and environment context data or let Canvas.Test use a default set of mock data. You provide initial mock application and environment context data in two Maps of key-value pairs. Use the predefined Canvas.Test key name constants as your keys. The following example sets the app name in the application context data and the sublocation in the environment context data.

```
Map<String,String> appValues = new Map<String,String>();
appValues.put(Canvas.Test.KEY_NAME,'AppName');

Map<String,String> envValues = new Map<String,String>();
envValues.put(Canvas.Test.KEY_SUB_LOCATION,'mobileDevice');
```

For the full list of context keys that Canvas.Test provides, see "Test Constants" in the Apex Code Developer Guide.

When you've got a mock RenderContext, you can call `Canvas.Test.testCanvasLifecycleHandler()` with the RenderContext and your CanvasLifecycleHandler. This call invokes the handler's `onRender()` method, and passies the mock RenderContext as context data.

Here's an example test class that uses Canvas.Test. The test class has three methods.

- `testDefaultMockValues()` invokes MyCanvasLifecycleHandler, using the default Canvas.Test mock context data.
- `testOverriddenAppValues()` invokes MyCanvasLifecycleHandler, using a custom mock RenderContext with mock app URL and version application context data.
- `testOverriddenEnvironmentValues()` invokes MyCanvasLifecycleHandler, using a custom mock RenderContext with mock display location, location URL, and custom parameter environment context data. The custom parameters are set through the Canvas.EnvironmentContext interface after the mock RenderContext is created.

```
-@isTest
global class MyCanvasLifecycleHandlerTest {

    static testMethod void testDefaultMockValues(){
        // Test handler using the default mock RenderContext Canvas.Test creates
        MyCanvasLifecycleHandler handler = new MyCanvasLifecycleHandler();
        Canvas.Test.testCanvasLifecycle(handler,null);
    }

    static testMethod void testOverriddenAppValues(){
        // Test handler with some mock application context values
        Map<String,String> appValues = new Map<String,String>();
        appValues.put(Canvas.Test.KEY_CANVAS_URL,'https://myserver.com:6000/myAppPath');
        appValues.put(Canvas.Test.KEY_VERSION,'3.0');

        Canvas.RenderContext mock = Canvas.Test.mockRenderContext(appValues,null);
        MyCanvasLifecycleHandler handler = new MyCanvasLifecycleHandler();
        Canvas.Test.testCanvasLifecycle(handler,mock);
    }

    static testMethod void testOverriddenEnvironmentValues(){
        // Test handler with some mock environment context values
        Map<String,String> envValues = new Map<String,String>();
        envValues.put(Canvas.Test.KEY_DISPLAY_LOCATION,'Chatter');
        envValues.put(Canvas.Test.KEY_LOCATION_URL,
            'https://myinstance.salesforce.com/_ui/core/chatter/ui/ChatterPage');
        MyCanvasLifecycleHandler handler = new MyCanvasLifecycleHandler();
        Canvas.RenderContext mock = Canvas.Test.mockRenderContext(null,envValues);
        // Directly update the mock RenderContext and set the custom parameters
        mock.getEnvironmentContext().setParametersAsJSON(
            '{\"one\":1,\"two\":2,\"bool\":true,\"stringVal\":\"some string\"}');
        Canvas.Test.testCanvasLifecycle(handler,mock);
    }

}
```

Distributing Your CanvasLifecycleHandler Class

If you package and distribute your canvas app, make sure to include your CanvasLifecycleHandler class in your package.

Use the following steps to include your CanvasLifecycleHandler implementation in your canvas app package.

1. In Salesforce, from Setup, enter *Packages* in the Quick Find box, then select **Packages**.
2. Find the package for your canvas app, and then click the package name to go to the package detail page.
3. On the package detail page, under Components, click **Add**.

4. In the Add to Package page, in the `Component Type` field, select **Apex Class**. A list of available Apex classes is shown.
5. Select the checkbox next to your Apex class for your CanvasLifecycleHandler implementation, and then click **Add To Package**.

CHAPTER 12 Objects

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

- CanvasRequest
- Controller
- SignedRequest

SEE ALSO:

- CanvasRequest
- SignedRequest

CanvasRequest

When you use a signed request for authentication in your canvas app, you receive a CanvasRequest object in the initial POST message from Salesforce. 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
`algorithm`	The algorithm used to sign the request.
`issuedAt`	For internal use only.
`userId`	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://MyDomainName.my.salesforce.com:8080
                /canvas_app_path/canvas_app.jsp",
            "developerName":"my_java_app",
            "isInstalledPersonalApp": false,
            "name":"My Java App",
            "namespace":"org_namespace",
```

```
        "options":[],
        "referenceId":"09HD00000000AUM",
        "samlInitiationMethod":"None",
        "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":"0DBxx000000001r",
        "profileId":"00ex0000000jzpt",
        "profilePhotoUrl":"/profilephoto/005/F",
        "profileThumbnailUrl":"/profilephoto/005/T",
        "roleId":null,
        "siteUrl":"https://MyDomainName.my.site.com/",
        "siteUrlPrefix":"/mySite",
        "timeZone":"America/Los_Angeles",
        "userId":"005x0000001SyyEAAS",
        "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",
            "maxHeight":"2000px",
            "maxWidth":"1000px"

        },
        "record":
        {
            "attributes":
```

```
                {
                    "type":"Account",
                    "url":"/services/data/v66.0/sobjects/Account/001xx000003DGWiAAO"
                },
                "Id":"001xx000003DGWiAAO",
                "Phone":"(555) 555-5555",
                "Fax":"(555) 555-5555",
                "BillingCity":"Seattle"
            },
            "displayLocation":"Chatter",
            "locationUrl":"https://salesforce.com/some/path/index.html",
            "subLocation":null,
            "uiTheme":"Theme3",
            "version":
            {
                "api":"66.0",
                "season":"SUMMER"
            },
        },
        "organization":
        {
            "currencyIsoCode":"USD",
            "multicurrencyEnabled":true,
            "name":"Edge Communications",
            "namespacePrefix":"org_namespace",
            "organizationId":"00Dx00000001hxyEAA"
        },
        "links":
        {
            "chatterFeedItemsUrl":"/services/data/v66.0/chatter/feed-items",
            "chatterFeedsUrl":"/services/data/v66.0/chatter/feeds",
            "chatterGroupsUrl":"/services/data/v66.0/chatter/groups",
            "chatterUsersUrl":"/services/data/v66.0/chatter/users",
            "enterpriseUrl":"/services/Soap/c/66.0/00Dx00000001hxy",
            "loginUrl":"http://MyDomainName.my.salesforce.com",
            "metadataUrl":"/services/Soap/m/66.0/00Dx00000001hxy",
            "partnerUrl":"/services/Soap/u/66.0/00Dx00000001hxy",
            "queryUrl":"/services/data/v66.0/query/",
            "recentItemsUrl":"/services/data/v66.0/recent/",
            "restUrl":"/services/data/v66.0/",
            "searchUrl":"/services/data/v66.0/search/",
            "sobjectUrl":"/services/data/v66.0/sobjects/",
            "userUrl":"/005x0000001SyyEAAS"
        }
    },
    "client":
    {
        "instanceId":"06Px000000002JZ",
        "instanceUrl":"http://MyDomainName.my.salesforce.com:8080",
        "oauthToken":"00Dx0000X00Or4J!ARQAQJ34YL8gKowP65p8FDHkvk.Uq5...",
        "refreshToken":"00DD0000000K1NM!ARAAQGJVGOyMjh1jjqvShRpLuAq0...",
        "targetOrigin":"http://MyDomainName.my.salesforce.com:8080"
    },
"algorithm":"HMACSHA256",
```

```
"userId":"005x0000001SyyEAAS",
"issuedAt":null
}
```

- Client
- Context

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, `http://instance.salesforce.com`. Used to preface the URLs returned by the Links object.
`oauthToken`	The OAuth access token that's returned to the caller.
`refreshToken`	The token that the client uses to authenticate a session. This value is returned only if the canvas app has "Perform requests on your behalf at any time (refresh_token, offline_access)" as one of the selected OAuth scopes; otherwise `null` is returned.
`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...",
    "refreshToken":"00DD0000000K1NM!ARAAQGJVGOyMjh1jjqvShRpLuAq0ke...",
```

```
    "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://MyDomainName.my.salesforce.com:8080
            /canvas_app_path/canvas_app.jsp",
        "developerName":"my_java_app",
        "isInstalledPersonalApp": false,
        "name":"My Java App",
        "namespace":"org_namespace",
        "options":[],
        "referenceId":"09HD00000000AUM",
        "samlInitiationMethod": "None",
        "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":"0DBxx000000001r",
```

```
        "profileId":"00ex0000000jzpt",
        "profilePhotoUrl":"/profilephoto/005/F",
        "profileThumbnailUrl":"/profilephoto/005/T",
        "roleId":null,
        "siteUrl":"https://MyDomainName.my.site.com/",
        "siteUrlPrefix":"/mySite",
        "timeZone":"America/Los_Angeles",
        "userId":"005x0000001SyyEAAS",
        "userName":"admin@6457617734813492.com",
        "userType":"STANDARD"
    },
    "environment":
    {
        "parameters":{},
        "dimensions":
        {
            "clientHeight": "50px",
            "clientWidth": "70px",
            "height":"900px",
            "width":"800px",
            "maxHeight":"2000px",
            "maxWidth":"1000px"

        },
        "record":
        {
            "attributes":
            {
                "type":"Account",
                "url":"/services/data/v66.0/sobjects/Account/001xx000003DGWiAAO"
            },
            "Id":"001xx000003DGWiAAO",
            "Phone":"(555) 555-5555",
            "Fax":"(555) 555-5555",
            "BillingCity":"Seattle"
        },
        "displayLocation":"Chatter",
        "locationUrl":"http://MyDomainName.my.salesforce.com:8080/canvas_app_path",
        "subLocation":null,
        "uiTheme":"Theme3",
        "version":
        {
            "api":"66.0",
            "season":"SUMMER"
         }
     },
    "organization":
    {
        "currencyIsoCode":"USD",
        "multicurrencyEnabled":true,
        "name":"Edge Communications",
        "namespacePrefix":"org_namespace",
        "organizationId":"00Dx00000001hxyEAA"
    },
```

```
    "links":
    {
        "chatterFeedItemsUrl":"/services/data/v66.0/chatter/feed-items",
        "chatterFeedsUrl":"/services/data/v66.0/chatter/feeds",
        "chatterGroupsUrl":"/services/data/v66.0/chatter/groups",
        "chatterUsersUrl":"/services/data/v66.0/chatter/users",
        "enterpriseUrl":"/services/Soap/c/66.0/00Dx00000001hxy",
        "loginUrl":"http://MyDomainName.my.salesforce.com",
        "metadataUrl":"/services/Soap/m/66.0/00Dx00000001hxy",
        "partnerUrl":"/services/Soap/u/66.0/00Dx00000001hxy",
        "queryUrl":"/services/data/v66.0/query/",
        "restUrl":"/services/data/v66.0/",
        "recentItemsUrl":"/services/data/v66.0/recent/",
        "searchUrl":"/services/data/v66.0/search/",
        "sobjectUrl":"/services/data/v66.0/sobjects/",
        "userUrl":"/005x0000001SyyEAAS"
    }
}
```

You can request Salesforce exclude portions of the context data that you don't need. To control the amount of data that's excluded, you must implement your own Canvas.CanvasLifecycleHandler in Apex and provide a list of excluded context sections in your implementation of `excludeContextTypes()`.

- Application
- Environment
- Links
- Organization
- User

SEE ALSO:

- Client
- Getting Context in Your Canvas App
- Filtering CanvasRequest Context Data

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 entering *Connected Apps* in the `Quick Find` box, then selecting **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: `http://instance.salesforce.com:8080/canvas_app_path/canvas_app.jsp`.

Field	Description
`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.
`options`	An array that can contain one, two, or three of these values: • `HideHeader`—Hides the publisher header which contains the "What are you working on?" text. This value is only used when the canvas app appears in the publisher. • `HideShare`—Hides the publisher **Share** button. This value is only used when the canvas app appears in the publisher. • `PersonalEnabled`—Enables the canvas app to be installed by users as a canvas personal app.
`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",
    "isInstalledPersonalApp": false,
    "name":"My Java App",
    "namespace":"org_namespace",
    "options":["HideShare", "HideHeader"],
    "referenceId":"09HD00000000AUM",
    "samlInitiationMethod":"None",
    "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 Record object, the Version object, and parameters (if there are any) in the Parameters object.

Field	Description
`displayLocation`	The location in the application where the canvas app is currently being called from. Valid values are: • Chatter—The canvas app was called from the Chatter tab. • ChatterFeed—The canvas app was called from a Chatter canvas feed item. • MobileNav—The canvas app was called from the navigation menu. • OpenCTI—The canvas app was called from an Open CTI component. • PageLayout—The canvas app was called from an element within a page layout. If the displayLocation is PageLayout, one of the subLocation values might be returned. • Publisher—The canvas app was called from a canvas custom quick action. • ServiceDesk—The canvas app was called from a Salesforce Console component. • Visualforce—The canvas app was called from a Visualforce page. • None—The canvas app was called from the Canvas App Previewer.
`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.
`subLocation`	The sublocation in the application where the canvas app was called from when the canvas app is loaded in a mobile device. The subLocation is related to the displayLocation. Valid values for this field when the displayLocation is PageLayout are: • S1MobileCardFullview—The canvas app was called from a mobile card. • S1MobileCardPreview—The canvas app was called from a mobile card preview. The user must click the preview to open the app. • S1RecordHomePreview—The canvas app was called from a record detail page preview. The user must click the preview to open the app. • S1RecordHomeFullview—The canvas app was called from a page layout.
`uiTheme`	The default theme for the context organization.

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

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

```
        "maxWidth":"1000px"

    },
    "record": {
        "attributes": {
            "type": "Account",
            "url": "/services/data/v66.0/sobjects/Account/001xx000003DGWiAAO"
        },
            "Id": "001xx000003DGWiAAO",
            "Phone": "(555) 555-5555",
            "Fax": "(555) 555-5555",
            "BillingCity": "Seattle"
    },
    "displayLocation":"PageLayout",
    "locationUrl":"https://salesforce.com/some/path/index.html",
    "subLocation":"S1MobileCardFullview",
    "uiTheme":"Theme3",
    "version":
    {
        "api":"66.0",
        "season":"SUMMER"
     }
}
```

Dimensions

Record

Parameters

Version

SEE ALSO:

Context

Dimensions

Record

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
`clientHeight`	The values of `clientHeight` are sent to the canvas app within SignedRequest and can be used to set the height of the outermost `div` 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

Field	Description
	`sr.context.environment.dimensions.clientHeight`, where `sr` is the parsed signed request JavaScript object.
`clientWidth`	The values of `clientWidth` are sent to the canvas app within SignedRequest and can be used to set the width of the outermost `div` 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 `sr.context.environment.dimensions.clientWidth`, where `sr` is the parsed signed request JavaScript object.
`height`	The height of the iFrame in pixels.
`width`	The width of the iFrame in pixels.
`maxHeight`	The maximum height of the iFrame in pixels. Defaults to 2,000; "infinite" is also a valid value.
`maxWidth`	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

Record

The Record object is a JSON-formatted object that contains information about the object on which the canvas app appears. This could be a standard object like `Account` or a custom object like `Warehouse__c`. If the canvas app doesn't appear on an object page or in the publisher or feed associated with an object, then the Record object is empty.

This object contains the Attributes object.

Field	Description
`Id`	The ID of the related object. For example, if the canvas app appears on an Account page, this field contains the `AccountId`.

If the canvas app appears on a Visualforce page through use of the `<apex:canvasApp>` component, and that Visualforce page is associated with an object (placed on the page layout, for example), you can use the `entityFields` attribute on the component to return specific fields or all fields from the related object.

The following code snippet shows an example of the Record object returned from a canvas app that appears on an Account page. In this example, the Phone, Fax, and BillingCity fields were specified in the `entityFields` attribute.

```
"record":{
    "attributes":{
        "type":"Account",
        "url":"/services/data/v66.0/sobjects/Account/001xx000003DGWiAAO"
    },
        "Id":"001xx000003DGWiAAO",
        "Phone":"(555) 555-5555",
        "Fax":"(555) 555-5555",
        "BillingCity":"Seattle"
    }
```

Attributes

Record Object Examples

These code examples demonstrate how the Record object contains different information depending on whether your canvas app appears in the publisher or the feed.

SEE ALSO:

Environment

Attributes

Record Object Examples

Returning Fields in the Record Object

Attributes

The Attributes object is a JSON-formatted object that contains information about the object on which the canvas app appears.

Field	Description
`type`	The object that the canvas app is associated with. This value could be a standard object like `Account` or a custom object like `Warehouse__c`. If the canvas app appears in the publisher, then the type is the name of the associated object. For example, `CollaborationGroup` or `Account`. If the canvas app appears in a user feed or the feed associated with an object, then the type is `FeedItem`.
`url`	The URL of the associated object. The format of the URL is the same as the REST API resource for the given record.

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

```
"record":{
    "attributes":{
        "type":"Account",
        "url":"/services/data/v66.0/sobjects/Account/001xx000003DGWiAAO"
    },
    "Id":"001xx000003DGWiAAO"
}
```

SEE ALSO:

Record

Record Object Examples

These code examples demonstrate how the Record object contains different information depending on whether your canvas app appears in the publisher or the feed.

Canvas App in the Publisher

If your canvas app appears in the publisher, the Record object contains different information depending on whether the publisher appears on a user record, a group record, or an object record.

User Record

The Record object `Id` is the `User.Id` and the `Type` is `User`.

```
"record":{
    "attributes":{
        "type":"User",
        "url":"/services/data/v66.0/sobjects/User/001xx000003DGWiABC"
        },
    "Id":"001xx000003DGWiABC"
    }
```

Group Record

The Record object `Id` is the `CollaborationGroup.Id` and the `Type` is `CollaborationGroup`.

```
"record":{
    "attributes":{
        "type":"CollaborationGroup",
        "url":"/services/data/v66.0/sobjects/CollaborationGroup/001xx000003DGWiXYZ"
        },
    "Id":"001xx000003DGWiXYZ"
    }
```

Object Record

The Record object `Id` is the object `Id` and the `Type` is the object name, for example, `Account`.

```
"record":{
    "attributes":{
        "type":"Account",
        "url":"/services/data/v66.0/sobjects/Account/001xx000003DGWiAAO"
        },
```

```
    "Id":"001xx000003DGWiAAO"
    }
```

Canvas App in the Feed

If your canvas app appears in the feed, the Record object always contains the feed item information whether the feed is on a user record, a group record, or an object record.

The Record object `Id` is the `FeedItem.Id` and the `Type` is `FeedItem`.

```
"reoord":{
    "attributes":{
        "type":"FeedItem",
        "url":"/services/data/v66.0/sobjects/FeedItem/001xx000003DGWiAAO"
        },
    "Id":"001xx000003DGWiAAO"
    }
```

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": "66.0",
    "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.

Note: The links that are listed here are the most commonly used links. They can be helpful when you're building the REST endpoints for your canvas app. For a complete list of links and related functionality, see the Lightning Platform REST API Developer's Guide and the Connect REST API Developer Guide.

Field	Description
`chatterFeedItemsUrl`	URL to return Chatter feed items for the context organization. This link will still work for integrations created prior to API version 66.0, because the link only points to functionality within API versions 31.0 and earlier.
`chatterFeedsUrl`	URL to return Chatter feeds for the context organization. This link will still work for integrations created prior to API version 66.0, because the link only points to functionality within API versions 31.0 and earlier.
`chatterGroupsUrl`	URL to return Chatter groups for the context organization.
`chatterUsersUrl`	URL that returns Chatter users for the context organization.
`enterpriseUrl`	URL to make API calls back to Salesforce using the enterprise WSDL.
`loginUrl`	URL of the login server for the instance the context user is logged into. Used internally.
`metadataUrl`	URL to make calls to the Metadata API.
`partnerUrl`	URL to make API calls back to Salesforce using the partner WSDL.
`queryUrl`	URL to issue a SOQL query.
`recentItemsUrl`	URL to access recent items.
`restUrl`	URL to return a list of REST API resources.
`searchUrl`	URL to issue a SOSL search.
`sobjectUrl`	URL that retrieves metadata or data from Salesforce objects.
`userUrl`	URL for the context user.

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

```
"links":
{
    "chatterFeedItemsUrl":"/services/data/v31.0/chatter/feed-items",
    "chatterFeedsUrl":"/services/data/v31.0/chatter/feeds",
    "chatterGroupsUrl":"/services/data/v66.0/chatter/groups",
    "chatterUsersUrl":"/services/data/v66.0/chatter/users",
    "enterpriseUrl":"/services/Soap/c/66.0/00Dx00000001hxy",
    "loginUrl":"https://login.salesforce.com",
    "metadataUrl":"/services/Soap/m/66.0/00Dx00000001hxy",
    "partnerUrl":"/services/Soap/u/66.0/00Dx00000001hxy",
    "queryUrl":"/services/data/v66.0/query/",
    "recentItemsUrl":"/services/data/v66.0/recent/",
    "restUrl":"/services/data/v66.0/",
    "searchUrl":"/services/data/v66.0/search/",
    "sobjectUrl":"/services/data/v66.0/sobjects/",
    "userUrl":"/005x0000001SyyEAAS"
}
```

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	• `userUrl` • `loginUrl`
Access and manage your data	• `enterpriseUrl` • `metadataUrl` • `partnerUrl` • `queryUrl` • `recentItemsUrl` • `restUrl` • `searchUrl` • `sobjectUrl`
Access and manage your Chatter data	• `chatterFeedItemsUrl` • `chatterFeedsUrl` • `chatterGroupsUrl` • `chatterUsersUrl`
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
`currencyIsoCode`	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.
`multicurrencyEnabled`	Indicates whether the context organization uses multiple currencies (`true`) or does not (`false`).
`namespacePrefix`	The Salesforce namespace prefix associated with the canvas app.
`name`	The name of the context organization.
`organizationId`	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 (`true`) or off (`false`).
`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.

Field	Description
`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 an Experience Cloud site (`true`) or not (`false`). **Important:** Canvas apps are not supported for guest users. They are supported on Experience Cloud sites only for logged in users.
`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 (_) 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 Experience Cloud site.
`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 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 Experience Cloud site.
`siteUrlPrefix`	The URL prefix of the context user's currently logged-in Experience Cloud site.
`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://MyDomainName.my.salesforce.com/",
    "siteUrlPrefix":"/mySite",
    "timeZone":"America/Los_Angeles",
    "userId":"005x0000001SyyEAAS",
    "userName":"admin@6457617734813492.com",
    "userType":"STANDARD"
}
```

SEE ALSO:

Context

Controller

If your Visualforce page contains a canvas app, you can use methods in the Controller object to publish an event that the canvas app subscribes to. It contains these methods:

Method	Signature	Description
`publish`	`publish(String name, Object payload [, String[] target])`	Used to publish an event that the canvas app subscribes to. The `target` parameter is optional. It can contain a single canvas app name or an array of canvas app names.
`resize`	`resize(String height, String width, String target)`	Used by a Visualforce page to resize a canvas app iFrame.
`subscribe`	`subscribe(Subscriber event)`	Used by a Visualforce page to subscribe to events that the canvas app might publish.

Method	Signature	Description
`unsubscribe`	`unsubscribe(Unsubscriber event)`	Used by a Visualforce page to unsubscribe to parent events that the canvas app might publish.

SEE ALSO:

- Using Events between a Visualforce Page and a Canvas App
- SignedRequest

SignedRequest

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 CanvasRequest Java object.
`verifyAndDecodeAsJson`	`public static String verifyAndDecodeAsJson(String input, String secret) throws SecurityException`	Used to verify and decode a signed request; returns the CanvasRequest data in a JSON string.

SEE ALSO:

- CanvasRequest
- Verifying and Decoding a Signed Request
- Requesting a Signed Request

CHAPTER 13 Canvas Limits

Because 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 Canvas calls per day per user (24–hour period)	5,000 per Salesforce full user license in the org. This limit is shared across all users in the org. Requests from guest users don't count against this limit. This limit tracks each Canvas app reload and signed request calls. Other API calls from the Canvas app are counted against the normal org API request limit. When you call a SignedRequest method, two calls are made—one for the method and one 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 Canvas
- Quick Start
- Quick Start—Advanced

INDEX

A

apex:canvasApp component 65
Application object 106
Associating a CanvasLifecycleHandler with an app 94
Aura
    canvas app code examples 73
Authentication
    OAuth 44
    refresh signed request 43
    SAML SSO 48
    signed request 37, 43

B

Browsers supported 4

C

Canvas app process
    overview 7
Canvas app user flow
    OAuth 45
    signed request 38
canvas apps 89
Canvas apps
    events 56
    in a mobile card 75
    in a page layout 75
    in the publisher 78
    listening for Streaming API events 59
    resizing 52, 69
    Visualforce 69
    where they appear in the user interface 3
    where they can appear in a page layout 76
Canvas apps in Salesforce app
    set the location 87
Canvas apps in the publisher
    create the action manually 79
    set the location and create the action 79
canvas personal apps
    distribution of 9
    how users install 9
    revoking token for 11
    uninstalling 11
Canvas Personal Apps
    about 8
Canvas scenarios 3
Canvas SDK
    hosted on Salesforce server 36
    publisher events 80
CanvasLifecycleHandler
    Associating a CanvasLifecycleHandler with an app 94
    Controlling canvas app behavior 95
    Creating a new CanvasLifecycleHandler 94
    Distributing your CanvasLifecycleHandler class 98
    Filtering CanvasRequest context data 95
    onRender() 95
    Presenting user error messages 96
    Testing your CanvasLifecycleHandler implementation 97
CanvasRequest
    object 100
CanvasRequest object 100
Chatter Feed
    context considerations when exposing a canvas app in the feed 84
Client object 103
Code examples
    events, creating an event 57
    events, subscribing to an event 58
    events, unsubscribing from an event 59
    exposing canvas app on a Lightning page 73
    exposing canvas app on a Visualforce page 64
    getting context 48
    OAuth flow 46
    publishing a canvas app in Visualforce by calling publish 70
    resizing, automatically resizing a canvas app by calling autogrow 53
    resizing, explicitly resizing a canvas app by calling resize 53
    resizing, explicitly resizing a canvas app in a Visualforce page by calling resize 71
    resizing, getting the size of a canvas app by calling size 54
    resizing, handling orientation changes by subscribing to the orientation event 55
    resizing, subscribing to events by calling subscribe 71
    resizing, subscribing to parent events by calling subscribe 54
    resizing, unsubscribing from events by calling unsubscribe 72
    Streaming API, creating a Streaming API event 60
    Streaming API, unsubscribing from a Streaming API event by calling unsubscribe 61
    verifying and decoding a signed request 39
    XHR, getting a list of Chatter users 50

Code examples *(continued)*
XHR, posting to a Chatter feed 50
Context
Application object 106
Client object 103
Dimensions object 109
getting from canvas app 48
Organization object 116
Parameters object 113
User object 116
Controller 118
Controlling canvas app behavior 95
Creating a new CanvasLifecycleHandler 94
Cross-domain XHR 49
Customizing app lifecycle
Associating a CanvasLifecycleHandler with an app 94
Controlling canvas app behavior 95
Filtering CanvasRequest context data 95
Presenting user error messages 96

D

Debug 61
Dimensions object 109
Distributing your CanvasLifecycleHandler class 98

E

Events
considerations 56
creating, code example 57
subscribing to, code examples 58
unsubscribing from, code examples 59
Events code examples
creating an event 57
subscribing to an event 58
unsubscribing from an event 59

F

Filtering CanvasRequest context data 95

L

Limits 120

M

Mobile card
canvas apps, adding to 75

N

navigation methods 89

O

OAuth
initiating 46
OAuth authentication 44
Organization object 116
Orientation
handling in your canvas app 55

P

Page layout
add a canvas app 76
canvas apps, adding to 75
canvas apps, where they can appear 76
Parameters object 113
Presenting user error messages 96
Process for creating a canvas app 7
Publisher
canvas app access considerations when exposing a canvas app in the publisher 82
canvas apps in 78
context considerations when exposing a canvas app in the publisher 81
Publisher events 80
Publishing code examples
publishing a canvas app in Visualforce by calling publish 70

Q

Quick start
create the app 13
introduction 12
prerequisites 13
Quick start—advanced
clone the “hello world” project 18
create app introduction 20, 25
create the canvas app 23
create the canvas external client app 21
deploy the “hello world” Web app to Heroku 27
deploy the canvas external client app 29
install the canvas app 33
introduction 17
package app introduction 31
package the connected app 32
package the external client app 31
prerequisites 18
run the Web app locally 19
update the canvas app 30
upload the canvas app package 32

R

Resizing 52, 69
Resizing code examples
    automatically resizing a canvas app by calling autogrow 53
    explicitly resizing a canvas app by calling resize 53
    explicitly resizing a canvas app in a Visualforce page by calling resize 71
    getting the size of a canvas app by calling size 54
    subscribing to events by calling subscribe 71
    subscribing to parent events by calling subscribe 54
    subscribing to the orientation event 55
    unsubscribing from events by calling unsubscribe 72

S

Salesforce app
    access considerations when exposing a canvas app in Salesforce app 88
    canvas apps in 86
    context considerations when exposing a canvas app in Salesforce app 87
    icon considerations when exposing a canvas app in Salesforce app 89
SAML SSO 48
SDK Objects 100
select tag in a canvas app 62
sfdc.streamingapi event 60
Signed request
    verifying and decoding 39
Signed request authentication 37, 43
SignedRequest
    object 119
SignedRequest object 119
Streaming API 59
Streaming API code examples
    unsubscribing from a Streaming API event by calling unsubscribe 61
Streaming API event 60
Supported browsers 4

T

Testing your CanvasLifecycleHandler implementation 97

U

User object 116
User permissions required 5

V

Visualforce
    apex:canvasApp component 65
    canvas app code examples 64
    considerations when exposing a canvas app 65

W

Where canvas apps appear 3

X

XHR 49
XHR code example
    getting a list of Chatter users 50
    posting to a Chatter feed 50