
Customizing Case Feed with Code

Version 38.0, Winter '17



CONTENTS

- Introduction 1
- Chapter 1:** Customizing the Layout and Appearance of Case Feed 3
- Chapter 2:** Customizing the Email Action 9
- Chapter 3:** Customizing the Portal Action 12
- Chapter 4:** Customizing the Log a Call Action 14
- Chapter 5:** Customizing the Articles Tool 17
- Chapter 6:** Replicating a Standard Case Feed Page 20
- Chapter 7:** Create Custom Actions 22
- Chapter 8:** Creating Custom Console Components That Interact with Case Feed 26
- Chapter 9:** Learning More 32

INTRODUCTION

Case Feed gives support agents a more streamlined way of creating, managing, and viewing cases. It includes actions and a Chatter feed. The actions let agents create case notes, log calls, change the status of cases, and communicate with customers. The feed displays important case events in chronological order, so it's easy to see the progress of each case.

The standard Case Feed page is designed for organizations that want to take advantage of feed-based case management quickly and with minimal configuration. You can modify the Case Feed page in a few ways, including specifying which actions and tools are available to users. Sometimes organizations with complex or unusual case management needs need to customize the Case Feed page more.

Case Feed Visualforce components enable you to create a customized page within the Salesforce console. In addition, the Case Feed-related events that can be published through the `publish` method on the `Sfdc.canvas.publisher` object in the Publisher JavaScript API let you create custom Salesforce console components that interact with Case Feed actions.

This guide is for developers who are responsible for customizing Case Feed according to their company's needs. It includes several use cases and examples to help you create a unique Case Feed page.

EDITIONS

Available in: Salesforce Classic

Available in: **Enterprise, Performance, Unlimited, and Developer** Editions

Requirements

Before customizing Case Feed in the Salesforce console, make sure:

- Case Feed, Chatter, and feed tracking on cases are enabled in your organization. Refer to [Implementing Case Feed](#) for detailed information.
- Your organization has at least one Salesforce console app. For more information, see [Set up a Salesforce Console App](#).
- You're familiar with developing with Visualforce. Check out the [Visualforce Developer's Guide](#) for a comprehensive overview.

Limitations

Lookup field filters aren't supported on any of the Case Feed Visualforce components.

Assigning Custom Pages to Users

Generally, when you create a custom Case Feed page using Visualforce, it's not possible to assign that page only to certain users while allowing other users to see the standard Case Feed page. However, with the `support:CaseFeed` component, you can create a page that replicates the standard Case Feed page, assign that page to certain users, and then create a custom page to assign to a different set of users. See [Replicating a Standard Case Feed Page](#) on page 20 for more information.

Customization Overview

There are five Case Feed Visualforce components:

Component Name	Description	Use It To...
<code>apex:emailPublisher</code>	Displays and controls the appearance and functionality of the Case Feed Email action.	<ul style="list-style-type: none"> Create an Email action and place it anywhere on a Salesforce console page. Change the appearance of the action by specifying its dimensions and the fields it includes. Customize certain aspects of the action's functionality, such as specifying a default Subject for each outgoing email.
<code>apex:logCallPublisher</code>	Displays and controls the appearance and functionality of the Case Feed Log a Call action.	<ul style="list-style-type: none"> Create a Log a Call action and place it anywhere on a Salesforce console page. Change the appearance of the action by specifying its dimensions and the fields it includes.
<code>support:caseArticles</code>	Displays and controls the appearance and functionality of the Articles tool for cases.	<ul style="list-style-type: none"> Create an Articles tool for cases and place it anywhere on a Salesforce console page. Change the appearance of the tool by specifying its dimensions. Customize certain aspects of the tool's functionality, such as how it searches for articles.
<code>support:CaseFeed</code>	Replicates the standard Case Feed page, including all standard actions, links, and buttons.	<ul style="list-style-type: none"> Create a version of the standard Case Feed page that you can assign to certain users so that you can also create a custom page and assign it to other users.
<code>support:portalPublisher</code>	Displays and controls the appearance and functionality of the Case Feed Portal action.	<ul style="list-style-type: none"> Create a Portal action and place it anywhere on a Salesforce console page. Change the appearance of the action by specifying its dimensions and the fields it includes.

In addition, the `chatter:feed` component has two attributes related to Case Feed: `feedItemType`, which lets you specify how feed items are filtered, and `showPublisher`, which lets you display the Chatter publisher on a page.

Finally, you can also create Visualforce pages to use as custom actions in Case Feed and can use the `publisher.selectAction`, `publisher.setActionInputValues`, `invokeAction`, and `customActionMessage` events to create interactions between custom Salesforce console components and Case Feed actions.

The following chapters offer detailed information on each of these components and customization options.

CHAPTER 1 Customizing the Layout and Appearance of Case Feed

Creating a customized Case Feed page with Visualforce lets you control the overall layout and appearance, including which actions and tools are shown and where they're located on the page. You can also include other standard and custom console components to enhance the functionality of the page.

In addition to the four case-specific Visualforce components detailed in this guide, you can also use the `chatter:feed` component to customize Case Feed. The table below lists its attributes.

`chatter:feed` Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
<code>entityId</code>	<code>id</code>	Entity ID of the record for which to display the feed; for example, <code>Contact.Id</code>	Yes	25.0	
<code>feedItemType</code>	<code>String</code>	The feed item type on which the Entity or <code>UserProfileFeed</code> is filtered. See the <code>Type</code> field on the <code>FeedItem</code> object listing in the API Object Reference Guide for accepted values.		25.0	
<code>id</code>	<code>String</code>	An identifier that allows the component to be referenced by other components on the page.		20.0	global
<code>onComplete</code>	<code>String</code>	The Javascript function to call after a post or comment is added to the feed		25.0	
<code>rendered</code>	<code>Boolean</code>	A Boolean value that specifies whether the additional fields defined in the action layout should be displayed.		20.0	global
<code>reRender</code>	<code>Object</code>	The ID of one or more components that are redrawn when the result of the action method returns to the client. This value can be a single ID, a comma-separated list of IDs, or a merge field expression for a list or collection of IDs.		25.0	
<code>showPublisher</code>	<code>Boolean</code>	Displays the Chatter publisher.		25.0	

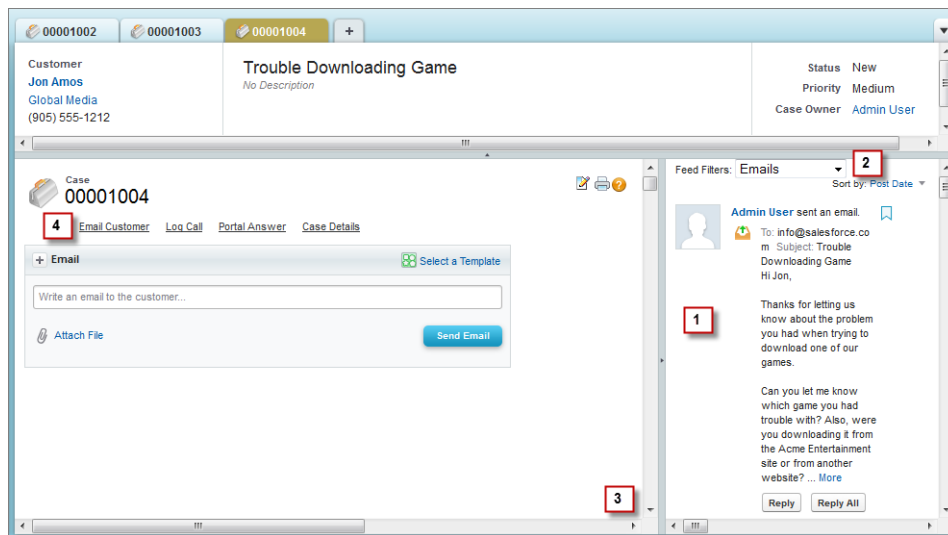
Use Case

Acme Entertainment creates online games used by more than a million people on multiple platforms. Acme's 1500 support agents use desktop computers, laptops, and tablets, and the company wanted to customize the Case Feed page to standardize its look and feel across different devices. They also wanted to make it easier for agents to track case activities using filters.

Acme used these steps to create a customized Case Feed page:

Customizing the Layout and Appearance of Case Feed

1. Using the `chatter:feed` component, they positioned the feed in the sidebar so the publisher and other Case Feed tools are always in the center of the page.
2. They repositioned the feed filter and auto-selected default filters depending on case origin:
 - If the case origin is email, the default filter is Emails.
 - If the case origin is phone, the default filter is Call Logs.
 - If the case origin is Web, the default filter is Portal Answers.
3. In `apex:emailPublisher` , `apex:logCallPublisher` , and `support:portalPublisher` , they made the width percentage-based so the publisher expands and contracts as the size of the page changes, making its appearance more consistent across different screen sizes.
4. They changed the orientation of the publisher action tabs from their standard left-side vertical arrangement to a horizontal arrangement at the top of the page.



Code Sample

This code sample shows a Visualforce page with custom Email, Portal, Log a Call, and Case Details tabs.

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

    <!-- Repositions publisher tabs to a horizontal arrangement on top of the page -->
    <ul class="demoNav" style="list-style: none; overflow: hidden">
        <li style="float:left">
            <a id="custom_email_tab" class="selected" href="javascript:void(0);"
                onclick="getDemoSidebarMenu().selectMenuItem('custom_email_tab');">
                <span class="menuItem">Email Customer</span>
            </a>
        </li>
        <li style="float:left">
            <a id="custom_log_call_tab" href="javascript:void(0);"
                onclick="getDemoSidebarMenu().selectMenuItem('custom_log_call_tab');">
                <span class="menuItem">Log Call</span>
            </a>
        </li>
    </ul>
</apex:page>
```



```

    </li>
    <li style="float:left">
        <a id="custom_portal_tab" href="javascript:void(0);"
            onclick="getDemoSidebarMenu().selectMenuItem('custom_portal_tab');">
            <span class="menuItem">Portal Answer</span>
        </a>
    </li>
    <li style="float:left">
        <a id="custom_detail_tab" href="javascript:void(0);"
            onclick="getDemoSidebarMenu().selectMenuItem('custom_detail_tab');">
            <span class="menuItem">Case Details</span>
        </a>
    </li>
</ul>

<!-- Email action -->
<div id="custom_email_pub_vf">
    <apex:emailPublisher entityId="{!case.id}"
        width="80%"
        emailBodyHeight="10em"
        showAdditionalFields="false"
        enableQuickText="true"
        toAddresses="{!case.contact.email}"
        toVisibility="readOnly"
        fromAddresses="support@cirrus.com"
        onSubmitSuccess="refreshFeed();" />
</div>

<!-- Log call action -->
<div id="custom_log_call_vf" style="display:none">
    <apex:logCallPublisher entityId="{!case.id}"
        width="80%"
        logCallBodyHeight="10em"
        reRender="demoFeed"
        onSubmitSuccess="refreshFeed();" />
</div>

<!-- Portal action -->
<div id="custom_portal_vf" style="display:none">
    <support:portalPublisher entityId="{!case.id}"
        width="80%"
        answerBodyHeight="10em"
        reRender="demoFeed"
        answerBody="Dear {!Case.Contact.FirstName},
            \n\nHere is the solution to your case.\n\nBest regards,\n\nSupport"
        onSubmitSuccess="refreshFeed();" />
</div>

<!-- Case detail page -->
<div id="custom_detail_vf" style="display:none">
    <apex:detail inlineEdit="true" relatedList="true" reRender="demoFeed" />
</div>

<!-- Include library for using service desk console API -->

```

```
<apex:includeScript value="/support/console/25.0/integration.js"/>

<!-- Javascript for switching publishers -->
<script type="text/javascript">
    function DemoSidebarMenu() {
        var menus = {"custom_email_tab" : "custom_email_pub_vf",
                     "custom_log_call_tab" : "custom_log_call_vf",
                     "custom_portal_tab" : "custom_portal_vf",
                     "custom_detail_tab" : "custom_detail_vf"};

        this.selectMenuItem = function(tabId) {
            for (var index in menus) {
                var tabEl = document.getElementById(index);
                var vfEl = document.getElementById(menus[index]);

                if (index == tabId) {
                    tabEl.className = "selected";
                    vfEl.style.display = "block";
                } else {
                    tabEl.className = "";
                    vfEl.style.display = "none";
                }
            }
        };
    }

    var demoSidebarMenu;
    var getDemoSidebarMenu = function() {
        if (!demoSidebarMenu) {
            demoSidebarMenu = new DemoSidebarMenu();
        }
        return demoSidebarMenu;
    };
</script>

<!-- Javascript for firing event to refresh feed in the sidebar -->
<script type="text/javascript">
    function refreshFeed() {
        sforce.console.fireEvent
            ('Cirrus.samplePublisherVFPage.RefreshFeedEvent', null, null);
    }
</script>
</apex:page>
```

The following sample shows an Apex class containing a controller extension to be used with the Visualforce page above.

```
public class MyCaseExtension {
    private final Case mycase;
    private String curFilter;

    public MyCaseExtension(ApexPages.StandardController stdController) {
        this.mycase = (Case)stdController.getRecord();

        // initialize feed filter based on case origin
        if (this.mycase.origin.equals('Email')) {
            curFilter = 'EmailMessageEvent';
        }
    }
}
```

Customizing the Layout and Appearance of Case Feed

```
        } else if (this.mycase.origin.equals('Phone')) {
            curFilter = 'CallLogPost';
        } else if (this.mycase.origin.equals('Web')) {
            curFilter = 'CaseCommentPost';
        }
    }

    public String getCurFilter() {
        return curFilter;
    }

    public void setCurFilter(String c) {
        if (c.equals('All')) {
            curFilter = null;
        } else {
            curFilter = c;
        }
    }

    public PageReference refreshFeed() {
        return null;
    }
}
```

This sample shows a Visualforce page with custom feed filters and Chatter feed for cases. You can use this page in the sidebar of a Salesforce console.

```
<apex:page standardController="Case" extensions="MyCaseExtension">

    <!-- Feed filter -->
    <div>
        <span>Feed Filters:</span>
        <select onchange="changeFilter(this.options[selectedIndex].value);"
            id="custom_filterSelect">
            <option value="All" id="custom_all_option">All</option>
            <option value="EmailMessageEvent"
                id="custom_email_option">Emails</option>
            <option value="CaseCommentPost"
                id="custom_web_option">Portal Answers</option>
            <option value="CallLogPost"
                id="custom_phone_option">Call Logs</option>
        </select>
    </div>

    <apex:form >
        <!-- actionFunction for refreshing feed when the feed filter is updated -->
        <apex:actionFunction action="{!refreshFeed}" name="changeFilter"
            reRender="custom_demoFeed" immediate="true" >
            <apex:param name="firstParam" assignTo="{!curFilter}" value="" />
        </apex:actionFunction>

        <!-- actionFunction for refreshing feed when there is an event fired for
            updating the feed -->
        <apex:actionFunction action="{!refreshFeed}" name="updateFeed"
            reRender="custom_demoFeed" immediate="true" />
    </apex:form>
</apex:page>
```

```
</apex:form>

<!-- Chatter feed -->
<chatter:feed entityId="{!case.id}" showPublisher="false"
    feedItemType="{!curFilter}" id="custom_demoFeed" />

<!-- Include library for using service desk console API -->
<apex:includeScript value="/support/console/25.0/integration.js"/>

<!-- Javascript for adding event listener for refreshing feed -->
<script type="text/javascript">

    var listener = function (result) {
        updateFeed();
    };

    // add a listener for the 'Cirrus.samplePublisherVFPage.RefreshFeedEvent'
    event type
    sforce.console.addEventListener('Cirrus.samplePublisherVFPage.RefreshFeedEvent',
        listener);
</script>

<!-- Javascript for initializing select option based on case origin -->
<script type="text/javascript">
    window.onload = function() {
        var caseOrigin = "{!case.origin}";
        if (!caseOrigin) {
            caseOrigin = "all";
        } else {
            caseOrigin = caseOrigin.toLowerCase();
        }
        var selectElem = document.getElementById('custom_' + caseOrigin + '_option');

        if (selectElem) {
            selectElem.selected = true;
        }
    }
</script>

</apex:page>
```

CHAPTER 2 Customizing the Email Action

The Email action in Case Feed lets support agents connect with customers via email. With the `apex:emailPublisher` component, you can:

- Customize the dimensions of the Email action.
- Define defaults and visibility for fields.
- Define the visibility and label of the send button.
- Define onSubmit functionality.
- Support email templates and attachments in the action.

`apex:emailPublisher` Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
<code>autoCollapseBody</code>	Boolean	A Boolean value that specifies whether the email body will collapse to a small height when it is empty.		25.0	
<code>bccVisibility</code>	String	The visibility of the BCC field can be 'editable', 'editableWithLookup', 'readOnly', or 'hidden'.		25.0	
<code>ccVisibility</code>	String	The visibility of the CC field can be 'editable', 'editableWithLookup', 'readOnly', or 'hidden'.		25.0	
<code>emailBody</code>	String	The default text value of the email body.		25.0	
<code>emailBodyFormat</code>	String	The format of the email body can be 'text', 'HTML', or 'textAndHTML'.		25.0	
<code>emailBodyHeight</code>	String	The height of the email body in em.		25.0	
<code>enableQuickText</code>	Boolean	A Boolean value that specifies whether the Quick Text autocomplete functionality is available in the action.		25.0	
<code>entityId</code>	id	Entity ID of the record for which to display the Email action. In the current version only Case record ids are supported.	Yes	25.0	
<code>expandableHeader</code>	Boolean	A Boolean value that specifies whether the header is expandable or fixed.		25.0	
<code>fromAddresses</code>	String	A restricted set of from addresses.		25.0	
<code>fromVisibility</code>	String	The visibility of the From field can be 'selectable' or 'hidden'.		25.0	
<code>id</code>	String	An identifier that allows the component to be referenced by other components on the page.		25.0	Global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
onSubmitFailure	String	The JavaScript invoked if the email is not successfully sent.		25.0	
onSubmitSuccess	String	The JavaScript invoked if the email is successfully sent.		25.0	
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		25.0	Global
reRender	Object	The ID of one or more components that are redrawn when the email is successfully sent. This value can be a single ID, a comma-separated list of IDs, or a merge field expression for a list or collection of IDs.		25.0	
sendButtonName	String	The name of the send button in the Email action.		25.0	
showAdditionalFields	Boolean	A Boolean value that specifies whether the additional fields defined in the action layout should be displayed.		25.0	
showAttachments	Boolean	A Boolean value that specifies whether the attachment selector should be displayed.		25.0	
showSendButton	Boolean	A Boolean value that specifies whether the send button should be displayed.		25.0	
showTemplates	Boolean	A Boolean value that specifies whether the template selector should be displayed.		25.0	
subject	String	The default value of the Subject.		25.0	
subjectVisibility	String	The visibility of the Subject field can be 'editable', 'readOnly', or 'hidden'.		25.0	
submitFunctionName	String	The name of a function that can be called from JavaScript to send the email.		25.0	
title	String	The title displayed in the Email action header.		25.0	
toAddresses	String	The default value of the To field.		25.0	
toVisibility	String	The visibility of the To field can be 'editable', 'editableWithLookup', 'readOnly', or 'hidden'.		25.0	
width	String	The width of the action in pixels (px) or percentage (%).		25.0	

Use Case

Cirrus Computers, a multinational hardware company with technical support agents in ten support centers throughout the world, wanted to customize the Email action to increase standardization in outgoing messages and to limit the fields agents could edit.

Cirrus used the `apex:emailPublisher` component to create an Email action that:

1. Has read-only To and Subject fields.

Customizing the Email Action

2. Pre-populates those fields, ensuring consistency and increasing agents' efficiency when writing email messages.

The screenshot shows a Salesforce interface for a Case record with ID 00001053. The 'Email' action is selected, and the 'To' field is pre-populated with 'info@acme.com' (highlighted with a red box and the number 1). The 'Subject' field is pre-populated with 'Your Cirrus support request' (highlighted with a red box and the number 2). The 'From' field is set to 'support@cirrus.com'. The 'Body' field is empty, with a placeholder text 'Write an email to the customer...'. There are 'Attach File' and 'Send Email' buttons at the bottom.

Code Sample

```
<apex:page standardController="Case" >
  <apex:emailPublisher entityId="{!case.id}"
    fromVisibility="selectable"
    subjectVisibility="readOnly"
    subject="Your Cirrus support request"
    toVisibility="readOnly"
    toAddresses="{!case.contact.email}"
    emailBody="" />
</apex:page>
```

CHAPTER 3 Customizing the Portal Action

The Portal action makes it easy for support agents to compose and post messages to customers on portals. With the `support:portalPublisher` component, you can:

- Customize the dimensions of the Portal action.
- Define a default value for the portal message text.
- Define the visibility and label of the submit button.
- Define onSubmit functionality.

`support:portalPublisher` Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
<code>answerBody</code>	String	The default text value of the answer body.		25.0	
<code>answerBodyHeight</code>	String	The height of the answer body in ems (em).		25.0	
<code>autoCollapseBody</code>	Boolean	A Boolean value that specifies whether the answer body is collapsed when it is empty.		25.0	
<code>entityId</code>	id	Entity ID of the record for which to display the Portal action. In the current version, only Case record ids are supported.	Yes	25.0	
<code>id</code>	String	An identifier that allows the component to be referenced by other components on the page.		25.0	Global
<code>onSubmitFailure</code>	String	The JavaScript invoked if the answer failed to be published to the portal.		25.0	
<code>onSubmitSuccess</code>	String	The JavaScript invoked if the answer was successfully published to the portal.		25.0	
<code>rendered</code>	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		25.0	Global
<code>reRender</code>	Object	The ID of one or more components that are redrawn when the answer is successfully published. This value can be a single ID, a comma-separated list of IDs, or a merge field expression for a list or collection of IDs.		25.0	
<code>showSendEmailOption</code>	Boolean	A Boolean value that specifies whether the option to send email notification should be displayed.		25.0	
<code>showSubmitButton</code>	Boolean	A Boolean value that specifies whether the submit button should be displayed.		25.0	

Attribute Name	Attribute Type	Description	Required?	API Version	Access
submitButtonName	String	The name of the submit button in the portal action.		25.0	
submitFunctionName	String	The name of a function that can be called from JavaScript to publish the answer.		25.0	
title	String	The title displayed in the portal action header.		25.0	
width	String	The width of the action in pixels (px) or percentage (%).		25.0	

Use Case

The Wellness Group is a healthcare company with 300 support agents in three tiers of support. Wellness wanted to customize the Portal action to reduce the amount of standard text, such as greetings and closings, agents had to type when replying to customers, which would help increase agents' efficiency and improve the standardization of portal communications.

Wellness used the `support:portalPublisher` component to create a Portal action that:

- Pre-populates the message body with a standard opening ("Hello {name}, and thanks for your question.") and a standard closing ("Please let me know if there's anything else I can do to help.").
- Lets agents edit the pre-populated text if needed.

Code Sample

```
<apex:page standardController="Case">
  <support:portalPublisher entityId="{!case.id}" width="800px"
    answerBody="Hello {!Case.Contact.FirstName}, and thanks for your question.
      \n\nPlease let me know if there's anything else I can do to help.">
  </support:portalPublisher>
</apex:page>
```

CHAPTER 4 Customizing the Log a Call Action

The Log a Call action lets support agents record notes and information about customer calls. With the `apex:logCallPublisher`, you can:

- Customize the appearance and dimensions of the Log a Call action.
- Specify which fields are displayed in the action.
- Define the visibility and label of the submit button.
- Define onSubmit functionality.

`apex:logCallPublisher` Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
<code>autoCollapseBody</code>	Boolean	A Boolean value that specifies whether the Log a Call body is collapsed when it is empty.		25.0	
<code>entityId</code>	id	Entity ID of the record for which to display the Log a Call action. In the current version, only Case record ids are supported.	Yes	25.0	
<code>id</code>	String	An identifier that allows the component to be referenced by other components on the page.		25.0	Global
<code>logCallBody</code>	String	The initial text value of the Log a Call body when the action is rendered.		25.0	
<code>logCallBodyHeight</code>	String	The height of the Log a Call body in em.		25.0	
<code>onSubmitFailure</code>	String	The JavaScript invoked if the call is not successfully logged.		25.0	
<code>onSubmitSuccess</code>	String	The JavaScript invoked if the call is successfully logged.		25.0	
<code>rendered</code>	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		25.0	Global
<code>reRender</code>	Object	The ID of one or more components that are redrawn when the call is successfully logged. This value can be a single ID, a comma-separated list of IDs, or a merge field expression for a list or collection of IDs.		25.0	
<code>showAdditionalFields</code>	Boolean	A Boolean value that specifies whether the additional fields defined in the action layout should be displayed.		25.0	

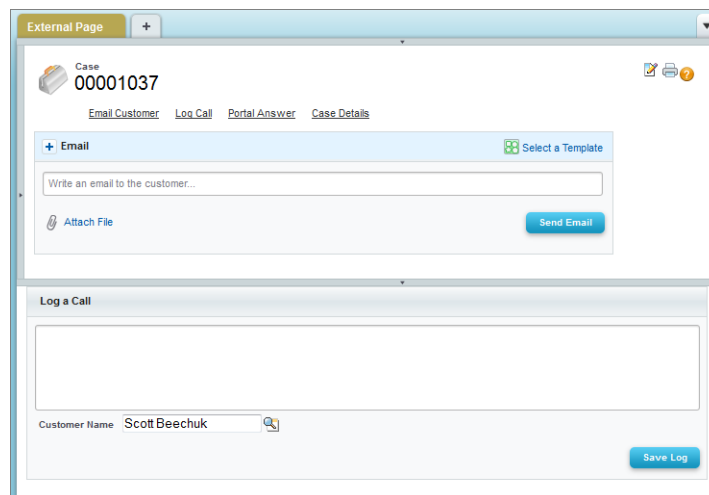
Attribute Name	Attribute Type	Description	Required?	API Version	Access
showSubmitButton	Boolean	A Boolean value that specifies whether the submit button should be displayed.		25.0	
submitButtonName	String	The name of the submit button in the Log a Call action.		25.0	
submitFunctionName	String	The name of a function that can be called from JavaScript to publish the call log.		25.0	
title	String	The title displayed in the Log a Call action header.		25.0	
width	String	The width of the action in pixels (px) or percentage (%).		25.0	

Use Case

Stellar Wireless is a mobile phone provider with several high-volume call centers, where agents are rewarded both for solving customers' issues quickly and for keeping detailed, accurate records of customer interactions. Stellar wanted to customize the Log a Call action so it was open and available to agents at all times, even when they were working with another action, giving them a quick and easy way of taking notes about incoming calls.

Stellar used the `apex:logCallPublisher` component to create a Log a Call action that:

- Appears in the footer of the page, replacing the standard interaction log.
- Is open and available by default each time a support agent opens a case.



Code Sample

```
<apex:page standardController="Case">
  <apex:logCallPublisher entityId="{!case.id}"
    width="100%"
```

Customizing the Log a Call Action

```
title="Log a Call"
autoCollapseBody="false"
showAdditionalFields="false"
submitButtonName="Save Log" />
</apex:page>
```

After you create a Visualforce page with this code, follow these steps to use the Log a Call action you create as a replacement for the standard interaction log:

1. From the object management settings for cases, go to Page Layouts.
2. Select the layout you're using from the Page Layouts for Case Feed Users list, and then select **Edit detail view**.
3. Click the **Custom Console Components** link at the top of the page.
4. In the Subtab Components section, use the lookup to select the page you created as the component to use for the bottom sidebar.
5. Specify the height of the action.
6. Click **Save**.
7. In the page layout editor, click **Layout Properties**.
8. Uncheck *Interaction Log*.
9. Click **OK**.
10. Click **Save**.

SEE ALSO:

[Salesforce Help: Find Object Management Settings](#)

CHAPTER 5 Customizing the Articles Tool

The Articles tool lets support agents browse Salesforce Knowledge articles, see whether articles are attached to a case, and share relevant articles with customers. With the `support:caseArticles` component, you can:

- Customize the appearance and dimensions of the Articles tool.
- Define how the tool's search function works, including which article types and keywords are used by default and whether advanced search is available.
- Specify whether agents can attach articles to emails.

`support:caseArticles` Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
<code>articleTypes</code>	String	Article types to be used to filter the search. Multiple article types can be defined, separated by commas.		25.0	
<code>attachToEmailEnabled</code>	Boolean	A Boolean value that specifies whether articles can be attached to emails.		25.0	
<code>bodyHeight</code>	String	The height of the body in pixels (px) or 'auto' to automatically adjust to the height of the currently displayed list of articles.		25.0	
<code>caseId</code>	id	Case ID of the record for which to display the case articles.	Yes	25.0	
<code>categories</code>	String	Data categories to be used to filter the search. The format of this value should be: 'CategoryGroup1:Category1' where CategoryGroup1 and Category1 are the names of a Category Group and a Category respectively. Multiple category filters can be specified separated by commas but only one per category group.		25.0	
<code>defaultKeywords</code>	String	The keywords to be used when the <code>defaultSearchType</code> attribute is 'keyword'. If no keywords are specified, the Case subject is used as a default.		25.0	
<code>defaultSearchType</code>	String	Specifies the default query of the article search form when it is first displayed. The value can be 'keyword', 'mostViewed', or 'lastPublished'.		25.0	
<code>id</code>	String	An identifier that allows the component to be referenced by other components on the page.		25.0	Global

Attribute Name	Attribute Type	Description	Required?	API Version	Access
language	String	The language used for filtering the search if multilingual Salesforce Knowledge is enabled.		25.0	
logSearch	Boolean	A Boolean value that specifies whether keyword searches should be logged.		25.0	
mode	String	Specifies whether the component displays articles currently attached to the case, an article search form, or both. The value can be 'attached', 'search', 'attachedAndSearch', or 'searchAndAttached'.		25.0	
onSearchComplete	String	The JavaScript invoked after an article search has completed.		25.0	
rendered	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		25.0	Global
reRender	Object	The ID of one or more components that are redrawn when the result of the action method returns to the client. This value can be a single ID, a comma-separated list of IDs, or a merge field expression for a list or collection of IDs.		25.0	
searchButtonName	String	The display name of the search button.		25.0	
searchFieldWidth	String	The width of the keyword search field in pixels (px).		25.0	
searchFunctionName	String	The name of a function that can be called from JavaScript to search for articles if the widget is currently in search mode.		25.0	
showAdvancedSearch	Boolean	A Boolean value that specifies whether the advanced search link should be displayed.		25.0	
title	String	The title displayed in the component's header.		25.0	
titlebarStyle	String	The style of the title bar can be 'expanded', 'collapsed', 'fixed', or 'none'.		25.0	
width	String	The width of the component in pixels (px) or percentage (%).		25.0	

Use Case

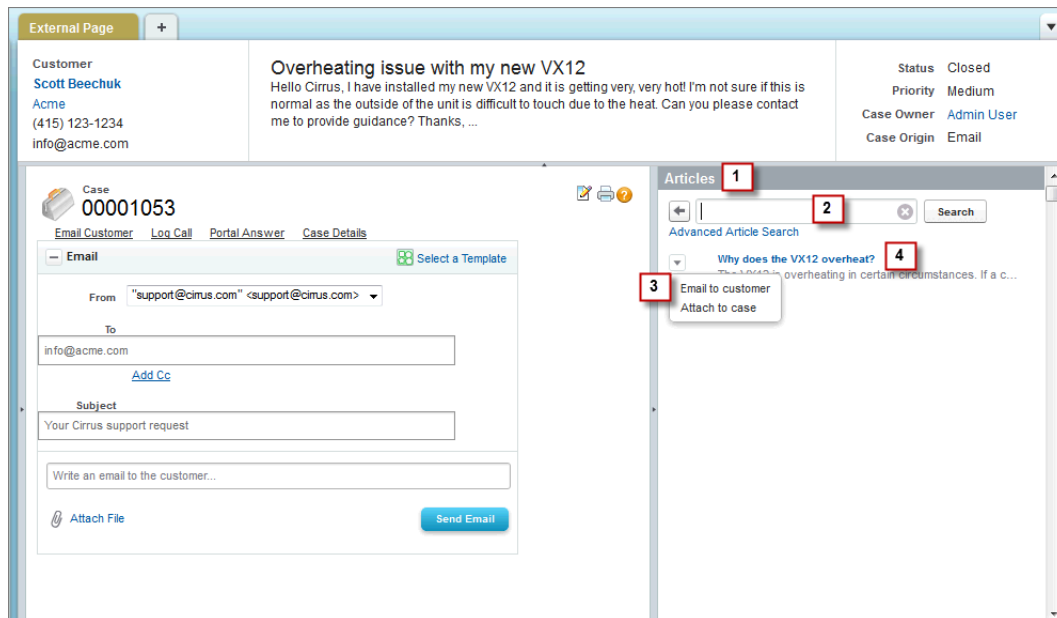
Cirrus Computers wanted to customize the Case Feed articles tool so agents could more easily find articles to help resolve customers' issues.

Cirrus used the `support:caseArticles` component to create an articles tool that:

1. Appears in the right sidebar of the page and is open by default on all case pages.

Customizing the Articles Tool

2. Uses search-as-you-type functionality to show suggested articles quickly.
3. Lets agents attach articles to messages they write with the [Email action](#).
4. Displays the most recently published articles when no articles are attached to a case.



Code Sample

```
<apex:page standardController="Case">
  <div style="margin-left:-10px;margin-right:-10px;">
    <div style="background-color: #99A3AC;color:#FFFFFF;font-size:1.1em;font-weight:
      bold;padding:3px 6px 3px 6px;">Articles</div>
    <support:caseArticles caseId="{!case.id}"
      bodyHeight="auto"
      titlebarStyle="none"
      searchButtonName="Search"
      searchFieldWidth="200px"
      defaultSearchType="lastPublished"
    />
  </div>
</apex:page>
```

CHAPTER 6 Replicating a Standard Case Feed Page

The `support:CaseFeed` component includes all of the elements of the standard Case Feed page:

- Email, Portal, Log a Call, and Case Note actions
- Case activity feed
- Feed filters
- Highlights panel
- Case following icon
- Case followers list
- Layout, print, and help links

`support:CaseFeed` Attributes

Attribute Name	Attribute Type	Description	Required?	API Version	Access
<code>caseId</code>	<code>id</code>	ID of the case record to display in Case Feed.	Yes	26.0	
<code>id</code>	String	An identifier that allows the component to be referenced by other components in the page.		26.0	global
<code>rendered</code>	Boolean	A Boolean value that specifies whether the component is rendered on the page. If not specified, this value defaults to true.		26.0	global

Use Case

National Foods is a food service company supplying restaurants and corporate cafeterias throughout the United States. National's support operations includes both call center agents who work primarily on desktop computers and field agents who work mainly on mobile devices. The company wanted a simplified Case Feed page that would be easy for its field agents to use, and also wanted to give its call center agents access to the full Case Feed functionality.

National used the `support:CaseFeed` component to recreate the standard Case Feed page for its call center agents working on desktops, and created a custom page for its field agents working on mobile devices.

Standard Case Feed page created with support : CaseFeed

The screenshot displays a Salesforce case record. At the top, the case details include the customer 'Edward Stamos' from 'Acme', the case number '00001018', the creation date '8/15/2012 5:15 PM', and the status 'New' with a 'Medium' priority. The case owner is 'Karen Williams'. The case title is 'Question about 8/20 shipment to Bravissimo', and the description states 'Customer has a question about the items included in the next shipment to Bravissimo restaurant'. On the left sidebar, there are buttons for 'Answer Customer', 'Log a Call', 'Write Case Note', 'Change Status', and 'View Case Details'. Below these are 'FEED FILTERS' for 'All Updates', 'Emails', and 'Status Changes'. The main content area has an 'Email' section with a 'Select a Template' button, a text input for writing an email, an 'Attach File' link, and a 'Send Email' button. Below the email section is an 'Articles' section. The feed shows 'All Updates for this case' sorted by 'Post Date', with a post from 'Karen Williams' stating 'Karen Williams created this case.' and a timestamp of 'Today at 5:15 PM'.

Code Sample

```
<apex:page standardController="Case"
    extensions="CasePageSelectorExtension" showHeader="true" sidebar="false">
    <apex:dynamicComponent componentValue="{!casePage}"/>
</apex:page>
```

The following sample shows an Apex class containing a controller extension to be used with the Visualforce page above.

```
public class CasePageSelectorExtension {
    boolean isFieldAgent;
    String caseId;

    public CasePageSelectorExtension(ApexPages.StandardController controller) {
        List<UserRole> roles = [SELECT Id FROM UserRole WHERE Name = 'FieldAgent'];
        isFieldAgent = !roles.isEmpty() && UserInfo.getUserRoleId() == roles[0].Id;
        caseId = controller.getRecord().id;
    }

    public Component.Apex.OutputPanel getCasePage() {
        Component.Apex.OutputPanel panel = new Component.Apex.OutputPanel();
        if (isFieldAgent) {
            Component.Apex.Detail detail = new Component.Apex.Detail();
            detail.subject = caseId;
            panel.childComponents.add(detail);
        } else {
            Component.Support.CaseFeed caseFeed = new Component.Support.CaseFeed();
            caseFeed.caseId = caseId;
            panel.childComponents.add(caseFeed);
        }
        return panel;
    }
}
```

CHAPTER 7 Create Custom Actions

You can create Visualforce pages to use as custom actions in Case Feed. For example, you can create a Map and Local Search action that lets agents look up the customer's location and find nearby service centers.

You can use any Visualforce page that uses the standard case controller as a custom action.

Use Case

Viaggio Italiano is a boutique travel agency specializing in tours of Italy. The company tracks multiple details for each client, including flights, ground transportation specifics, dietary preferences, and itineraries. Viaggio Italiano's agents needed the ability to create long case comments but were limited to 1000 characters for standard case notes. The company wanted a way to bypass this limit.

Viaggio Italiano used Visualforce to create a page that includes the ability to post a case comment, which can be up to 4000 characters long. The company then added the page as a custom action by editing the Case Feed page layout.

Customer: Helen Ingersoll

Case: 00001017 Created Date: 8/15/2012 11:07 AM

Add a Montepulciano excursion

Customer wants to add a side trip to Montepulciano to her Tuscany vacation package

Answer Customer

Log a Call

Write Case Note

Change Status

View Case Details

Post Case Comment

Customer Details

- * Solo traveler, 45 years old
- * Special occasion?: Yes--birthday
- * Tuscan Explorer package, October 10-22, 2012
- * First time in Italy, but has been to France and Spain before
- * Interests: Renaissance art, wine, cooking, farm stays
- * Found us through website

Flying from Chicago, IL; see below for specific arrival and departure details.

Booking a Montepulciano excursion. Also expressed interest in other add-ons: live entertainment

Post Case Comment

Code Samples

The following code sample shows a custom Post Case Comment action for an organization that doesn't have actions in the publisher enabled, or that has actions in the publisher enabled but uses the Case Feed Settings page, not the page layout editor, to choose and configure the actions in the Case Feed publisher.

```
<apex:page standardcontroller="Case"
    extensions="CaseCommentExtension" showHeader="false">
    <apex:includeScript value="/support/api/26.0/interaction.js"/>
    <div>
        <apex:form >
            <!-- Creates a case comment and on complete notifies the Case Feed page
                that a related list and the feed have been updated -->
            <apex:actionFunction action="{!addComment}" name="addComment" rerender="out"

            oncomplete="sforce.interaction.entityFeed.refreshObject('{!case.id}',
            false, true);"/>
            <apex:outputPanel id="out" >
                <apex:inputField value="{!comment.commentbody}" style="width:98%;
                height:160px;" />
            </apex:outputPanel>
        </apex:form>
    </div>
</apex:page>
```

Create Custom Actions

```
        </apex:outputPanel>
    </apex:form><br />
    <button type="button" onclick="addComment();" style="position:fixed; bottom:0px;

        right:2px; padding: 5px 10px; font-size:13px;" id="cpbutton" >Post Case Comment
    </button>
</div>
</apex:page>
```

This is the code to use for the custom Post Case Comment action if your organization has actions in the publisher enabled and you've opted to use the page layout editor to choose and configure actions in the Case Feed publisher.

```
<apex:page standardcontroller="Case"
    extensions="CaseCommentExtension" showHeader="false">
    <!-- Uses publisher.js rather than interaction.js -->
    <apex:includeScript value="/canvas/sdk/js/28.0/publisher.js"/>
    <div>
        <apex:form >
            <!-- Creates a case comment and on complete notifies the Case Feed page
                that a related list and the feed have been updated -->
            <apex:actionFunction action="{!addComment}" name="addComment" rerender="out"

            <!-- Different oncomplete function using publisher.js -->
            oncomplete="Sfdc.canvas.publisher.publish(
                {name : 'publisher.refresh', payload :
                {feed: true, objectRelatedLists: {}}});"/>
            <apex:outputPanel id="out" >
                <apex:inputField value="{!comment.commentbody}" style="width:98%;
                    height:160px;" />
            </apex:outputPanel>
        </apex:form><br />
        <button type="button" onclick="addComment();" style="position:fixed; bottom:0px;

            right:2px; padding: 5px 10px; font-size:13px;" id="cpbutton" >Post Case Comment
        </button>
    </div>
</apex:page>
```

The following sample shows an Apex class containing a controller extension to be used with either version of the Visualforce page above.

```
public with sharing class CaseCommentExtension {
    private final Case caseRec;
    public CaseComment comment {get; set;}

    public CaseCommentExtension(ApexPages.StandardController controller) {
        caseRec = (Case)controller.getRecord();
        comment = new CaseComment();
        comment.parentid = caseRec.id;
    }

    public PageReference addComment() {
        insert comment;
        comment = new CaseComment();
        comment.parentid = caseRec.id;
        return null;
    }
}
```

```
}  
}
```


Additional Steps

After creating a Visualforce page, make it available to users.

First, give profiles access to the page:

1. From Setup, enter *Visualforce Pages* in the **Quick Find** box, then select **Visualforce Pages**.
2. Click **Security** next to the name of the page you created.
3. Choose the profiles you want to be able to access the page.
4. Click **Save**.

Then include the page as a custom action. If you're using the Case Feed Settings page to choose and configure actions:

1. From the object management settings for cases, go to Page Layouts.
2. How you access the Case Feed Settings page depends on what kind of page layout you're working with..
 - For a layout in the Case Page Layouts section, click **Edit**, and then click **Feed View** in the page layout editor.
 - For a layout in the Page Layouts for Case Feed Users section, click  and choose **Edit feed view**. (This section appears only for organizations created before Spring '14.)
3. In Custom Actions, click **+ Add a Visualforce page**.
4. Choose the page you want to add.
5. Specify the height of the action. For the best appearance, we recommend a height of 200 pixels.
6. In Select Actions, move the custom action from **Available** to **Selected**.
7. Click **Save**.


If you've opted to use the page layout editor to choose and configure actions, you first need to create the custom action:

1. From the object management settings for cases, go to Buttons, Links, and Actions.
2. Click **New Action**.
3. Select **Custom Visualforce**.
4. Select the Visualforce page you created, then specify the height of the action window. (The width is fixed.)
5. Type a label for the action. This is the text users will see for the action in the publisher.
6. If necessary, change the name of the action.
7. Type a description for the action. The description appears on the detail page for the action and in the list on the Buttons, Links, and Actions page. The description isn't visible to your users.
8. Optionally, click **Change Icon** to select a different icon for the action. This icon appears only when you use the action through the API.

Then add the action to a page layout:

1. From the object management settings for cases, go to Page Layouts.
2. How you access the page layout editor depends on what kind of page layout you're working with..
 - For a layout in the Case Page Layouts section, click **Edit**, and then click **Feed View** in the page layout editor.

Create Custom Actions

- For a layout in the Page Layouts for Case Feed Users section, click  and choose `Edit detail view`. (This section appears only for organizations created before Spring '14.)

3. Click **Quick Actions** in the palette.
4. Drag the action from the palette to the Quick Actions in the Salesforce Classic Publisher section.
5. Click **Save**.

SEE ALSO:

[Salesforce Help: Find Object Management Settings](#)

CHAPTER 8 Creating Custom Console Components That Interact with Case Feed

Custom console components let you extend the functionality of the Salesforce console, and you can create those components so they interact with any actions you've added to the Case Feed publisher. For example, you could develop a component that generates customized, pre-written text, adds that text to a new post in the Case Feed portal action, and submits the post to the portal, all with one click.

These events can be published through the `publish` method on the `Sfdc.canvas.publisher` object in the Publisher JavaScript API to allow console components to interact with Case Feed quick actions.

EDITIONS

Available in: Salesforce Classic

Available in: **Enterprise, Performance, Unlimited,** and **Developer** Editions

`publisher.selectAction`

Description	Payload Values	Available Versions
Selects the specified action and puts it in focus.	<p><code>actionName</code>—The action to select. Supported values are:</p> <ul style="list-style-type: none">• <code>action_name</code>—A create, log a call, or custom Visualforce quick action. For example, <code>action_name</code> for a create contact action might be <code>create_contact</code>.• <code>Case.CaseComment</code>—Case Feed portal action• <code>Case.ChangeStatus</code>—Case Feed change status action• <code>Case.Email</code>—Case Feed email action• <code>Case.LogACall</code>—Case Feed log a call action• <code>FeedItem.TextPost</code>—Standard Chatter post action (Available in API versions 32.0 and later)• <code>SocialPostAPIName.SocialPost</code>—Social post action (Available in API versions 32.0 and later)	Available in API versions 29.0 and later.

Code Sample

This code snippet selects the email action and puts it in focus.

```
Sfdc.canvas.publisher.publish({name:"publisher.selectAction",payload:{actionName:"Case.Email"}});
```

`publisher.setActionInputValues`

Description	Payload Values	Available Versions
Specifies which fields on the action should be populated with specific values, and what those values are.	<p><code>actionName</code>—The action on which fields should be populated.</p> <p>The available field values depend on which action you specify.</p> <ul style="list-style-type: none">• <code>emailFields</code>—Available on <code>Case.Email</code>; the standard available fields on the Case Feed email action:<ul style="list-style-type: none">– <code>to</code>– <code>cc</code>– <code>bcc</code>– <code>subject</code>– <code>body</code>– <code>template</code>• <code>portalPostFields</code>—Available on <code>Case.CaseComment</code>; the standard available fields on the Case Feed portal action:<ul style="list-style-type: none">– <code>body</code>– <code>sendEmail</code> (boolean)• <code>targetFields</code>—Available on <code>Case.ChangeStatus</code>, <code>Case.LogACall</code>, <code>FeedItem.TextPost</code>, and the Social action; the standard available fields on those actions.<ul style="list-style-type: none">– On <code>Case.ChangeStatus</code>: <code>commentBody</code>– On <code>Case.LogACall</code>: <code>description</code>– On <code>FeedItem.TextPost</code>: <code>body</code> Attributes on <code>body</code> are <code>value</code> and <code>insertType</code> (optional). Valid values for <code>insertType</code> are <code>begin</code>, <code>end</code>, <code>cursor</code>, and <code>replace</code>. The default value is <code>replace</code>. (Available in API versions 32.0 and later)– On <code>SocialPostAPIName</code>. <code>SocialPost</code>: <code>content</code> and <code>insertType</code> (optional). Valid values for <code>insertType</code> are <code>begin</code>, <code>end</code>, <code>cursor</code>, and <code>replace</code>. The default value is <code>replace</code>. (Available in API versions 32.0 and later)• <code>parentFields</code>—Available on <code>Case.ChangeStatus</code>, <code>Case.Email</code>, and <code>Case.LogACall</code>; standard and custom fields on case. Lookup fields aren't supported.	Available in API versions 29.0 and later.

Code Samples

Creating Custom Console Components That Interact with Case Feed

This code snippet populates the fields on an email message with predefined values, and sets the status of the associated case to Closed.

```
Sfdc.canvas.publisher.publish({name:"publisher.setActionInputValues",
  payload:{actionName:"Case.Email",parentFields: {Status:{value:"Closed"}},
  emailFields: {to:{value:"customer@company.com"},cc:{value:"customer2@company.com"},
  bcc:{value:"supervisor@company.com"}},
  subject:{value:"Your Issue Has Been Resolved"},
  body:{value:"Thank you for working with our support department.
    We've resolved your issue and have closed this ticket, but
    please feel free to contact us at any time if you encounter this
    problem again or need other assistance."}}});
```

This code snippet inserts the phrase “Hello World” in the body of the Post action at the current cursor position.

```
Sfdc.canvas.publisher.publish({name:"publisher.setActionInputValues",
  payload:{actionName:"FeedItem.TextPost", targetFields:{body:{value:"Hello World",
  insertType:"cursor"}}}});
```

invokeAction

Description	Payload Values	Available Versions
Triggers the submit function (such as sending an email or posting a portal comment) on the specified action.	<p><code>actionName</code>—The action on which to trigger the submit function. Supported actions are:</p> <ul style="list-style-type: none">• <code>Case.Email</code>• <code>Case.CaseComment</code>• <code>Case.ChangeStatus</code>• <code>Case.LogACall</code>• <code>FeedItem.TextPost</code> (Available in API versions 32.0 and later)• <code>SocialPostAPIName.SocialPost</code> (Available in API versions 32.0 and later)	Available in API versions 29.0 and later.

Code Sample

This code snippet triggers the submit function on the email action, sending an email message and generating a related feed item.

```
Sfdc.canvas.publisher.publish({name:"publisher.invokeAction",
  payload:{actionName:"Case.Email"}});
```

customActionMessage

Description	Payload Values	Available Versions
Passes a custom event to a custom action. Supported for Visualforce-based custom actions only.	<p><code>actionName</code>—The Visualforce custom action to pass the event to.</p> <p><code>message</code>—The event to pass to the custom action.</p>	Available in API versions 29.0 and later.

Creating Custom Console Components That Interact with Case Feed

Code Sample

This code snippet passes the Hello world event to the action my_custom_action.

```
Sfdc.canvas.publisher.publish({name:"publisher.customActionMessage",  
payload:{actionName:"my_custom_action", message:"Hello world"}});
```

This code snippet is what my_custom_action uses to listen to the Hello world event.

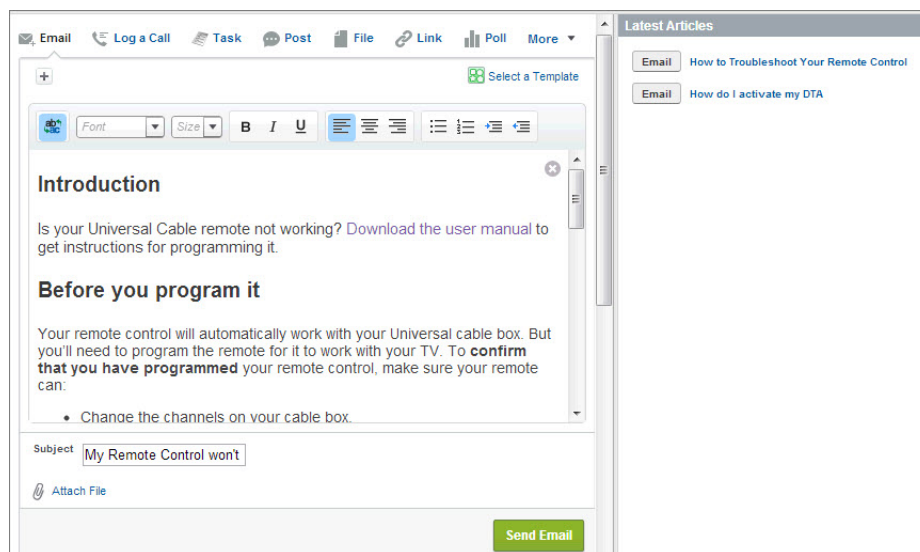
```
Sfdc.canvas.publisher.subscribe([{name : "publisher.customActionMessage", onData :  
function(e) {alert(e.message);}}]);
```

Use Case

Universal Cable serves millions of phone and cable customers throughout the United States, with 4000 support agents in call centers of varying sizes around the country. Universal wanted to make it easy for agents to access the company's extensive collection of articles in Salesforce Knowledge and share them with customers through email to help keep support costs in check.

Universal used the events on `publish` to create a custom console component that:

- Displays a list of Knowledge articles, from most recently published to oldest.
- Lets agents view an article by clicking its title.
- Lets agents add the full, formatted text of an article to a message in the Case Feed email action by clicking the **Email** button in the console component.



Code Sample

This code sample shows an Apex class containing a custom controller used by the Visualforce page below.

```
public with sharing class KBController {  
    public List<FAQ__kav> articles {get; set;}  
  
    public KBController() {  
        articles = [select knowledgearticleid, id, title, content__c from FAQ__kav where  
publishstatus = 'Online' and language='en_US' order by lastpublisheddate];  
    }  
}
```

Creating Custom Console Components That Interact with Case Feed

```
}  
}
```

This code sample shows the Visualforce page that's used as the custom console component in the use case above.

```
<apex:page sidebar="false" controller="KBController">  
  <script type='text/javascript' src='/canvas/sdk/js/publisher.js'/>  
  <style>  
    .sampleTitle { background-color: #99A3AC;color:#FFFFFF;font-size:1.1em;  
    font-weight: bold;padding:3px 6px 3px 6px; }  
    .sampleHeader { }  
    .sampleArticleList { min-width: 250px; padding: 8px 0 5px 0;}  
    .sampleUl { padding: 0; margin: 0; list-style: none;}  
    .sampleLi { display: block; position: relative; margin: 0;}  
    .sampleRow { min-height: 16px; padding: 4px 10px;}  
    .emailBtn { margin: 1px 1px 1px 3px; padding: 3px 8px; color: #333;  
      border: 1px solid #b5b5b5; border-bottom-color: #7f7f7f; background: #e8e8e9;  
      font-weight: bold; font-size: .9em; -moz-border-radius: 3px;  
      -webkit-border-radius: 3px; order-radius: 3px; }  
    .emailBtn:active { background-position: right -60px; border-color: #585858;  
      border-bottom-color: #939393; }  
    .sampleArticle { padding-left: 4px; padding-bottom: 2px; font-weight: bold;  
      font-size: 1em; color: #222; }  
    .sampleLink { color: #015ba7; text-decoration: none; font-weight: bold;  
      font-size: .9em; }  
  </style>  
  <script>  
    function emailArticle(content) {  
      Sfdc.canvas.publisher.publish({name: 'publisher.selectAction',  
      payload: { actionName: 'Case.Email'}});  
      Sfdc.canvas.publisher.publish({name: 'publisher.setActionInputValues',  
      payload: {  
        actionName: 'Case.Email',  
        emailFields: { body: { value:content, format:'richtext', insert: true}}  
      }}});  
    }  
  </script>  
  <div style="margin-left:-10px;margin-right:-10px;">  
    <div class="sampleTitle">Latest Articles</div>  
    <div class="sampleHeader" style=""></div>  
    <div class="sampleArticleList">  
      <apex:repeat value="{!articles}" var="article">  
        <ul class="sampleUl">  
          <li class="sampleLi">  
            <div class="sampleRow">  
              <div style="display:none;" id="content_{!article.id}">  
                <apex:outputText value="{!article.content__c}" escape="false"/>  
              </div>  
              <input type="button" title="Email" value="Email" class="emailBtn"  
                onclick="emailArticle(document.getElementById  
                ('content_{!article.id}').innerHTML);"/>  
              <span class="sampleArticle">  
                <a href="/{!article.knowledgearticleid}"  
                  title="{!article.title}" class="sampleLink">  
                  {!article.title}</a>  
            </div>  
          </li>  
        </ul>  
      </apex:repeat>  
    </div>  
  </div>  
</apex:page>
```

Creating Custom Console Components That Interact with Case Feed

```
                </span>
            </div>
        </li>
    </ul>
</apex:repeat>
</div>
</div>
</apex:page>
```

CHAPTER 9 Learning More

Use these resources to learn more about Case Feed and Visualforce:

- [Implementing Case Feed](#)
- [Getting to Know Case Feed](#)
- [Visualforce Developer's Guide](#)