Developing Zendesk® Widgets

Zendesk Inc.

Notice

Copyright and trademark notice

Developing Zendesk® Widgets

Zendesk - 989 Market St, Ste 300 - San Francisco - CA 94103 - USA

www.zendesk.com

Contents

Chapter 1

Developing Zendesk® Widgets

This document covers Zendesk widgets, which can be used in the Zendesk Classic interface and in the Web portal. You cannot add widgets to the new Zendesk agent interface; you need to use the new *Zendesk Apps framework* instead.

Introduction

Widgets are html fragments that can be embedded directly on any page of the Zendesk Classic interface and the end-user facing Web portal. Use widgets to integrate external data into your Zendesk. You can furthermore access all of the data in your Zendesk via JSON, mash-up with external data, and even replace existing elements in the help desk layout.

For advanced usage, see the *JavaScript Resource API*.

Text placholders

You can use text placeholder in your widgets to include information about the selected objects in your Zendesk pages. If you wanted to insert the name of the user who is logged in to the system you could insert `{{current_user.name}}`.

Examples

HELLO USER

```
Hi {{current_user.name}}
```

See *Widget Text Placeholders* for more info on the available placeholders.

Accessing the REST API

You can access all data in your help desk directly from the widget via JSON. This example lists the number of users in your help desk, and displays the name of the first user. The data is shown on the page in the inner HTML of the div with ID “active-views”. You can place the content of your widget in any div with an ID!

Widget access to all data in your help desk via JSON

```
<script>
   new Ajax.Request('/users.json', {
      method:'get',
      asynchronous: true,
      onSuccess: function(transport){
         var obj = transport.responseText.evalJSON();
```

```
        $('active-views').innerHTML = '<p>'
           + obj.length
           + ' users in your help desk.'
           + ' First user is '
           + obj[0].name
           + '</p>'
     }
   });
</script>
```

Integrating data from and to your enterprise

If you have pre-populated or updated the user database via REST, you can utilize the attribute `{{current_user.external_id}}` for the internal user ID for your enterprise, thus using JavaScript calls in the widget to fetch and present additional data from your repositories – e.g. license or asset data for the current user.

Integrating user data from and to your enterprise

```
<script type="text/javascript"
  src="https://database.your_enterprise.com/get_user_assets.php?
id={{current_user.external_id}}">
</script>
```

This example calls an internal method at your enterprise. The method returns JavaScript listing users assets. This could also be done with an iframe returning plain HTML from your enterprise.

JavaScript Resource API

The Zendesk Resource API allows developers to build Zendesk widgets that push data into external sources, and optionally pulls data from external sources back into ticket events. The external sources are URLs that act as *REST* resources.

The Resource API works exclusively through JavaScript. If your sources are REST enabled, all you need is JavaScript in your widget to utilize the Resource API.

The API itself is a JavaScript library (*/external/push/zd_resource.js*) that HTTP GETs and POST/PUTs your REST resources through a Zendesk server proxy. The server proxy is necessary due to *cross-site scripting* restrictions present in most browsers. The JavaScript library uses the *Prototype Framework*.

To illustrate how the Resource API is used, we will recreate the Harvest widget as a custom widget. The Harvest widget allows agents to register time spent on a ticket through *Harvest*.

Upon submit of the time registration form, values associated with the time registration are pushed back to the ticket as an event.

The contents of the Harvest widget is:

```
<div id="harvest-time-tracking"
 domain="yourharvestaccountname.harvestapp.com"
  use_ssl="true">
  <h3 id="title"></h3>
  <div id="content"></div>
</div>
```

```
<script src="/external/push/harvest.js"
  type="text/javascript">
</script>
```

The div with ID “harvest-time-tracking” includes Harvest domain definitions and content placeholders for the widget. Following the Harvest div is the JavaScript for the widget. This could be inline, but in this example it resides on the Zendesk server. It could just as easily reside on a server of your own choosing.

The following will be a walk-though of the Harvest widget JavaScript.

Resource initialization

At the bottom of the JavaScript code file for the Harvest widget (*/external/push/harvest.js*) you'll find the initialization of the API’s Resource object:

```
harvest_resource = new Zendesk.Resource(
  {
    title: 'Harvest time tracking',
    anchor: 'harvest-time-tracking',
    domain: $('harvest-time-tracking').readAttribute('domain'),
    use_ssl: $('harvest-time-tracking').readAttribute('use_ssl') ||
 'false',
    login_content: login,
    application_content: application,
    application_resources: [
      {
        resource: 'daily',
        on_success: projects_selector
      }
    ]
  }
);
```

Lets go to through each of the objects initialization parameters:

title

The title of the widget. Appears on the widget form.

anchor

DOM ID of the element where the widget will be placed.

domain

The resource domain. In this example, the value is read from the “domain” attribute in the widget div ID – i.e. yourharvestaccountname.harvestapp.com.

use_ssl

If you need to communicate with your resource domain through SSL, set this attribute to true. In this example, the value is read from the “use_ssl” attribute in the widget div ID.

cache_gets

Caches the content of the application part (see section “The application function”).

login_content

References the function in your JavaScript that generates the login form (if any). See section “The login function”.

application_content

References the function in your JavaScript that generates the application part. See section “The application function”.

application_resources

An array of resources that needs to be consulted in order for the application part of the widget to be displayed correctly. Typically values for form elements. In our Harvest example we have one application resource, “daily” that generates the values used to populate the “Select project” and “Select task” parts of the widget form.

```
{resource: 'daily', on_success: projects_selector}
```

Thus, the application resource is yourharvestaccountname.harvestapp.com/daily. The resource is processed asynchronously, and the result is sent as a JSON object to the **projects_selector** function on success. This functions traverses the JSON object, picking the project and tasks values on your Harvest account, and dynamically inserts those in the widget application drop-downs.

The Login function

Your widget might need to capture login credentials for the agent, if your resource(s) require authentication. These login credentials are subsequently passed on to the resources, when e.g. populating the application part of the widget (see next section) or pushing data to the resource.

The login function for our example widget is trivial, and can be found at the top of the JavaScript code file for the Harvest widget (*/external/push/harvest.js*). It simply writes out the login form html.

When an agent submit the login form, the login credentials will be cached by the Resource API. Subsequently, the widget will no longer display the login form, but the application part of the widget.

The application function

The application function in the Harvest example generates the time registration form – the application part of the widget. Have a look at the application form in the JavaScript. First the function defines placeholders for the tasks and projects drop-downs and the notes and hours text fields. These are all parameters that is required for the Harvest time registration. When the form is submitted, request[field] values are POST’ed to a defined resource. Secondly, various hidden form fields that are not directly associated with the time registration are defined. The value of these instructs the Zendesk proxy what to do with the form content – e.g. which resource to POST to.

Lets take a closer look at each of the hidden parameters.

resource

The resource to POST the form parameters to. In this case, yourharvestaccountname.harvestapp.com/daily/add.

media_type

application/xml or **application/json**. Use the latter if the resource returns *JSON* when GET'ing.

event_reference

Title on the ticket event we will be logging.

event_log

Values to log to the ticket event. These values correspond to values in the answer from the POST to the resource. In this case the result from POSTing the form parameters to yourharvestaccountname.harvestapp.com/daily/add. The result will always be delivered from the proxy as JSON, regardless of the original format of the resource's return.

pluck_param

Defines the form fields to pass on to the resource. In this example fields prepended with "request" – e.g. request[notes].

Widget text placeholders

You can use text placeholders in your widgets to include information about the selected objects in your help desk pages. If you want to insert the name of the user who is currently logged in, you can insert the following placeholder:

```
{{current_user.name}}
```

Depending on what page a wizard is displayed on, it has access to different text placeholders.

The current_user text placeholders

All pages have access to the `current_user` placeholders if a user is logged in.

`{{current_user.name}}`

The name of the user logged in to your help desk.

`{{current_user.first_name}}`

The first name of the user logged in to your help desk.

`{{current_user.email}}`

The email address of the user logged in to your help desk.

`{{current_user.organization.name}}`

The organization of the user logged in to your help desk.

`{{current_user.organization.notes}}`

The notes about the organization of the user logged in to your help desk.

`{{current_user.organization.details}}`

The details about the organization of the user logged in to your help desk.

{{current_user.external_id}}

The external id of the user logged in to your help desk.

{{current_user.phone}}

The phone number of the user logged in to your help desk.

{{current_user.details}}

Detailed information for the user logged in to your help desk.

{{current_user.notes}}

Notes for the user logged in to your help desk.

{{current_user.language}}

The language for the user logged in to your help desk.

{{current_user.tags}}

The tags of the user logged in to your help desk.

{{current_user.id}}

The id of the user logged in to your help desk.

The ticket text placeholders

Pages where you are viewing a ticket have access to the `ticket` placeholders.

{{ticket.title}}

Ticket subject.

{{ticket.description}}

Ticket description.

{{ticket.url}}

Full URL path to ticket (excluding “http://”).

{{ticket.id}}

Unique ticket ID.

{{ticket.external_id}}

External ticket ID. Typically populated through the API..

{{ticket.via}}

The source channel of the ticket – e.g. “Mail” or “Twitter”.

{{ticket.status}}

Ticket status.

{{ticket.priority}}

Ticket priority.

{{ticket.ticket_type}}

Ticket type.

{{ticket.score}}

Score.

{{ticket.group.name}}

Ticket group.

{{ticket.due_date}}

Ticket due date (relevant for tickets of type Task).

{{ticket.latest_comment_formatted}}

Latest comment.

{{ticket.latest_public_comment_formatted}}

Latest public comment.

{{ticket.comments_formatted}}

All comments, latest first.

{{ticket.public_comments_formatted}}

All public comments, latest first.

{{ticket.account}}

Help desk name.

{{ticket.assignee.name}}

Ticket assignee full name (if any).

{{ticket.assignee.first_name}}

Ticket assignee first name (if any).

{{ticket.assignee.last_name}}

Ticket assignee last name (if any).

{{ticket.requester.name}}

Ticket requester full name.

{{ticket.requester.first_name}}

Ticket requester first name.

{{ticket.requester.last_name}}

Ticket requester last name.

{{ticket.requester.email}}

Ticket requester email.

{{ticket.requester.language}}

Ticket requester language.

{{ticket.requester.phone}}

Ticket requester phone number.

{{ticket.requester.id}}

Ticket requester ID.

{{ticket.requester.external_id}}

Ticket requester external ID.

{{ticket.organization.name}}

Ticket requester organization.

{{ticket.cc_names}}

Ticket CCs.

{{ticket.tags}}

All tags registered on the ticket.

{{ticket.in_business_hours}}

Renders “true” if update is within business hours.

{{ticket.ticket_field_ID}}

Ticket custom field. E.g, `{{ticket.ticket_field_123}}`.

{{ticket.ticket_field_option_title_ID}}

For drop-down custom fields, this yields the title of the drop-down value selected (as opposed to the tag, which is what the previous placeholder yields). E.g `{{ticket.ticket_field_option_title_123}}`.

{{ticket.incoming_phone_number}}

Zendesk Voice inbound phone number.

The requester text placeholders

Pages where you are viewing a ticket have access to the `requester` placeholders.

{{requester.name}}

The name of the user who requested the ticket.

{{requester.first_name}}

The first name of the user who requested the ticket.

{{requester.email}}

The email address of the user who requested the ticket.

{{requester.organization.name}}

The organization of the user who requested the ticket.

{{requester.organization.notes}}

The notes about the organization of the user who requested the ticket.

{{requester.organization.details}}

The details about the organization of the user who requested the ticket.

{{requester.external_id}}

The external id of the user who requested the ticket.

{{requester.phone}}

The phone number of the user who requested the ticket.

{{requester.details}}

Detailed information for the user who requested the ticket.

{{requester.notes}}

Notes for the user who requested the ticket.

{{requester.language}}

The language for the user who requested the ticket.

{{requester.tags}}

The tags of the user who requested the ticket.

The user text placeholders

Pages where you are viewing a user have access to the `user` placeholders.

{{user.name}}

The name of the user.

{{user.first_name}}

The first name of the user.

{{user.email}}

The email address of the user.

{{user.organization.name}}

The organization of the user.

{{user.organization.notes}}

The notes about the organization of the user.

{{user.organization.details}}

The details about the organization of the user.

{{user.external_id}}

The external id of the user.

{{user.phone}}

The phone number of the user.

{{user.details}}

Detailed information for the user.

{{user.notes}}

Notes for the user.

{{user.language}}

The language for the user.

{{user.tags}}

The tags of the user.