Gadgets and Dashboards Documentation

Atlassian Gadgets and Dashboards Documentation

Administration Guide
User Guide
Development Hub
FAQ
Glossary
Release Notes

About Gadgets and Dashboards

What are Atlassian Gadgets?

Atlassian gadgets are similar to Google gadgets. A gadget is a small object (i.e. a piece of functionality) offering dynamic content that you can put onto a dashboard, a wiki page or some other web page. Atlassian gadgets allow interaction with Atlassian applications such as JIRA and Confluence.

What is an Atlassian Dashboard?

A dashboard is usually the landing page of an application such as JIRA or Confluence. With the newer Atlassian dashboards, you can add gadgets and personalise the dashboard display by moving the gadgets around, changing their colour, and changing other preferences to suit you.

Resources

If you have a question about using an Atlassian gadget or dashboard within an Atlassian application, please contact our support team.

Other handy links:

- Javadoc for Atlassian Gadgets and for the Atlassian Gadgets API
  Click through to the 'Atlassian Gadgets API' page for the version you need, then select 'Project Reports', then 'JavaDocs'.
- Issue tracker for Atlassian Gadgets
- Atlassian and OpenSocial
- More about making your Atlassian applications work together
- Guide to Installing an Atlassian Integrated Suite

Download the Documentation

You can download the Gadgets and Dashboards documentation in PDF, HTML or XML formats.

All Versions

Gadgets and Dashboards Documentation

Current released version
Atlassian Gadgets **1.0.3** has now been released — see the Atlassian Gadgets 1.0.3 Release Notes.

Table of Contents

Introduction to Atlassian Gadgets and Dashboards

Gadgets and Dashboards Administration Guide

Gadgets and Dashboards User Guide

Gadgets and Dashboards Development Hub

Gadgets and Dashboards FAQ

Gadgets and Dashboards Glossary

Atlassian Gadgets Release Notes

Introduction to Atlassian Gadgets and Dashboards

This page introduces you to Atlassian gadgets and the new dashboards that support gadgets.

On this page:

Overview

What are Atlassian Gadgets?

Atlassian gadgets are similar to Google gadgets. A gadget is a small object (i.e. a piece of functionality) offering dynamic content that you can put onto a dashboard, a wiki page or some other web page. Atlassian gadgets allow interaction with Atlassian applications such as JIRA and Confluence.

What is an Atlassian Dashboard?

A dashboard is usually the landing page of an application such as JIRA or Confluence. With the newer Atlassian dashboards, you can add gadgets and personalise the dashboard display by moving the gadgets around, changing their colour, and changing other preferences to suit you.

Where Does a Gadget's Information Come From?

Gadgets can supply information from a number of places. When you choose a gadget, check where its information is coming from. A gadget's information will come from one or more of the following sources:

- The application where your dashboard is running, such as JIRA or iGoogle.
- Another installation of the same application. For example, you may use a gadget to collect information from two or more JIRA servers.
- Any location on the web. For example, the Google Map Search gadget gets information from Google Maps.

Some gadgets allow you to edit the source of the information so that, when the gadget is running on your own dashboard, the gadget displays information from a specific server. See our guide to editing a gadget's settings.

Making Gadgets Work for You

What do the new Atlassian gadgets and dashboards do for you?

1. Bring in content from all over, Atlassian and non-Atlassian applications, and display it on your (JIRA) dashboard:
    - From multiple Atlassian applications e.g. from JIRA, Confluence, Bamboo etc all into one dashboard.
    - From different instances of each application.
    - From external sources that are not Atlassian applications.

2. Make it easier to integrate other applications with JIRA (and vice versa).

3. More modern UI for dashboards:
    - Better tabs so you can flip between them more quickly.
    - Drag-and-drop to re-arrange portlets.
    - Ajax for quick reaction e.g. when delete a portlet you don't have to re-load the entire page to see the effect.
    - Re-configure layout to multiple columns — also happens dynamically.

4. Adopt open standards:
    - Base it on the Google Gadgets spec. Already in use by Google, LinkedIn, Salesforce to both consume and expose gadgets.
    - We want to be able to consume data from other apps in JIRA
    - We want also to expose JIRA data to other portals e.g. iGoogle. So you should be able to display JIRA data in iGoogle and other portals.

5. Re-focus the purpose of the dashboard:
    - Make it about teams, projects and tasks, not about tools. So you don't have to go to your Bamboo dashboard(s), your JIRA dashboard(s) etc. Instead, you can pull it all into one spot. So you go to the dashboard that represents your project, or a particular task that you're focused on, and it gives you all the info you need for that particular task/project, all in one place.

RELATED TOPICS

Gadgets and Dashboards Documentation

- Introduction to Atlassian Gadgets and Dashboards
- Gadgets and Dashboards Administration Guide
- Gadgets and Dashboards User Guide
- Gadgets and Dashboards Development Hub
- Gadgets and Dashboards FAQ

- Gadgets and Dashboards Glossary
- Atlassian Gadgets Release Notes

Gadgets and Dashboards Administration Guide

> ⚠ **Security implications**
> Add only gadgets from sources that you trust. Gadgets can allow unwanted or malicious code onto your web page and into your application. A gadget specification is just a URL. The functionality it provides can change at any time.

- Adding a Gadget to the Directory of Available Gadgets
- Configuring OAuth
- Removing a Gadget from the Directory of Available Gadgets

RELATED TOPICS

Gadgets and Dashboards Documentation

Adding a Gadget to the Directory of Available Gadgets

The gadget directory displays all the gadgets that are available to users. These are the gadgets that users can add to their dashboard.

You need to have administrator privileges to add a gadget to the directory. If you have permission to add gadgets to and remove gadgets from the directory itself, you will see the '**Add Gadget to Directory**' and '**Remove**' buttons on the 'Add Gadget' screen, as shown below.

If you have administrator privileges, you can add gadgets from Atlassian applications such as Confluence, JIRA and others. You can also add gadgets from other websites such as iGoogle. Many public gadgets will work on an Atlassian dashboard. Some gadgets may rely on specific iGoogle features that will not work properly on Atlassian dashboards. In that case, you can simply remove the gadget from the directory.

On this page:

- Adding a Gadget that is Not a Plugin
- Adding a Gadget that must be Installed as a Plugin

> ⚠ **Security implications**
> Add only gadgets from sources that you trust. Gadgets can allow unwanted or malicious code onto your web page and into your application. A gadget specification is just a URL. The functionality it provides can change at any time.

There are two types of gadgets: those that must be installed as plugins, and those that can be added as simple gadget URLs.

Adding a Gadget that is Not a Plugin

If the gadget is hosted on another server and can be added to the directory as a simple URL, then you can simply add it via your dashboard's 'Add Gadget' option.

To add a gadget to your directory,

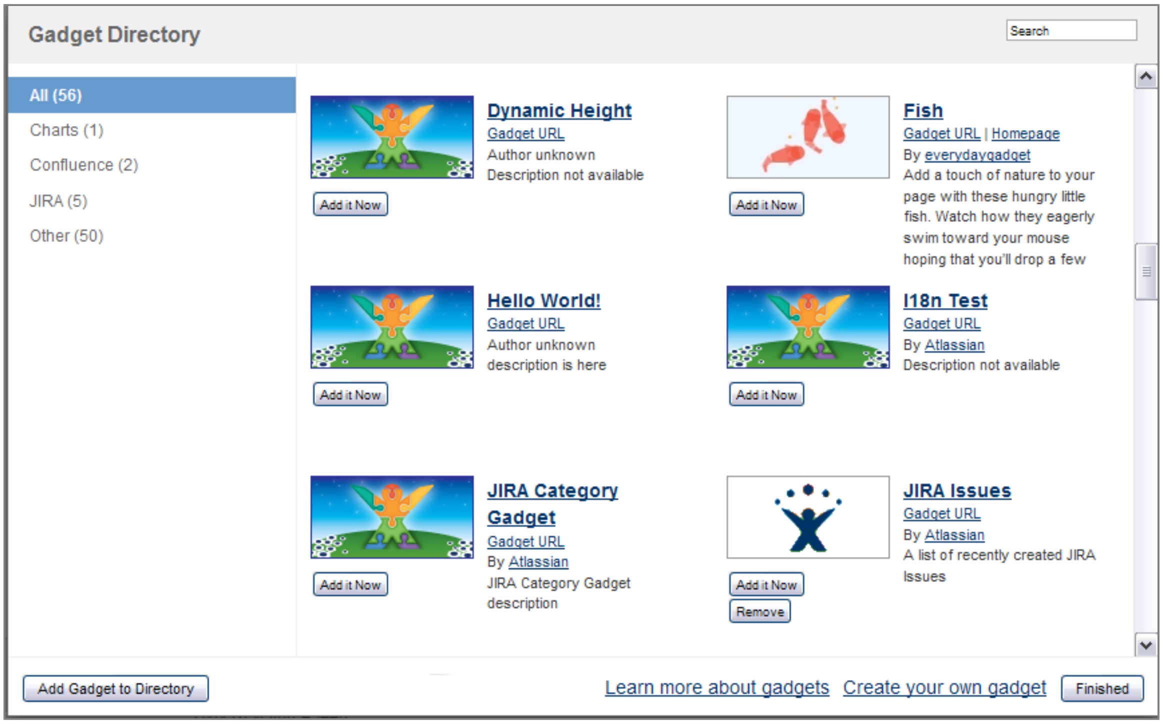
1. First you need to find the URL for the gadget's XML specification file. A gadget's URL looks something like this:
   `http://example.com/my-gadget-location/my-gadget.xml`
   Gadget authors and publishers make their gadget URLs available in different ways. Below are the instructions for a Google gadget and an Atlassian gadget.

   - Follow the steps below to find the URL for a Google gadget:
     a. Go to the Google gadget directory. (You can also get there by clicking 'Add Stuff' from your iGoogle home page.)
     b. Search for the gadget you want.
     c. Click the link on the gadget to open its home page.
     d. Find the '**View source**' link near the bottom right of the page. Right-click the link and copy its location to your clipboard. This is the gadget's URL.

   - Follow the steps below to find the URL for a gadget that is published by an Atlassian application, such as JIRA or Confluence:
     a. Open the host application, such as JIRA.
        Currently, only JIRA 4.0 has a dashboard that supports gadgets.
     b. Go to the host application's dashboard by clicking the '**Dashboard**' link or the '**Home**' link at the top left of the screen.
     c. Click '**Add Gadget**' to see the list of gadgets in the host application's directory. The list will look something like screenshot 1 below.
     d. Find the gadget you want, then copy the location of the '**Gadget URL**'.
        The URL will have this format:
        `http://my-app.my-server.com:port/rest/gadgets/1.0/g/my-plugin.key:my-gadget/my-pa`

        For example:
        `http://mycompany.com/jira/rest/gadgets/1.0/g/com.atlassian.streams.streams-jira-p`
2. Now you can add the gadget to your directory. Go to the dashboard by clicking the '**Dashboard**' link or the '**Home**' link at the top left of the screen.
3. The dashboard will appear. Click '**Add Gadget**'.
4. The '**Add Gadget**' screen appears, showing the list of gadgets in your directory. See screenshot 1 below. Click '**Add Gadget to Directory**'.
   You will only see this button if you have administrator permissions for your dashboard.
5. The '**Add Gadget to Directory**' screen appears. See screenshot 2 below. Type or paste the gadget URL into the text box.
6. Click '**Add Gadget**'.
7. The gadget appears in your gadget directory. (It will be highlighted for a short time, so that you can see it easily.)

Screenshot 1: Gadget directory with 'Add Gadget to Directory' button



Dynamic Height
Gadget URL
Author unknown
Description not available

Fish
Gadget URL | Homepage
By everydaygadget
Add a touch of nature to your page with these hungry little fish. Watch how they eagerly swim toward your mouse hoping that you'll drop a few

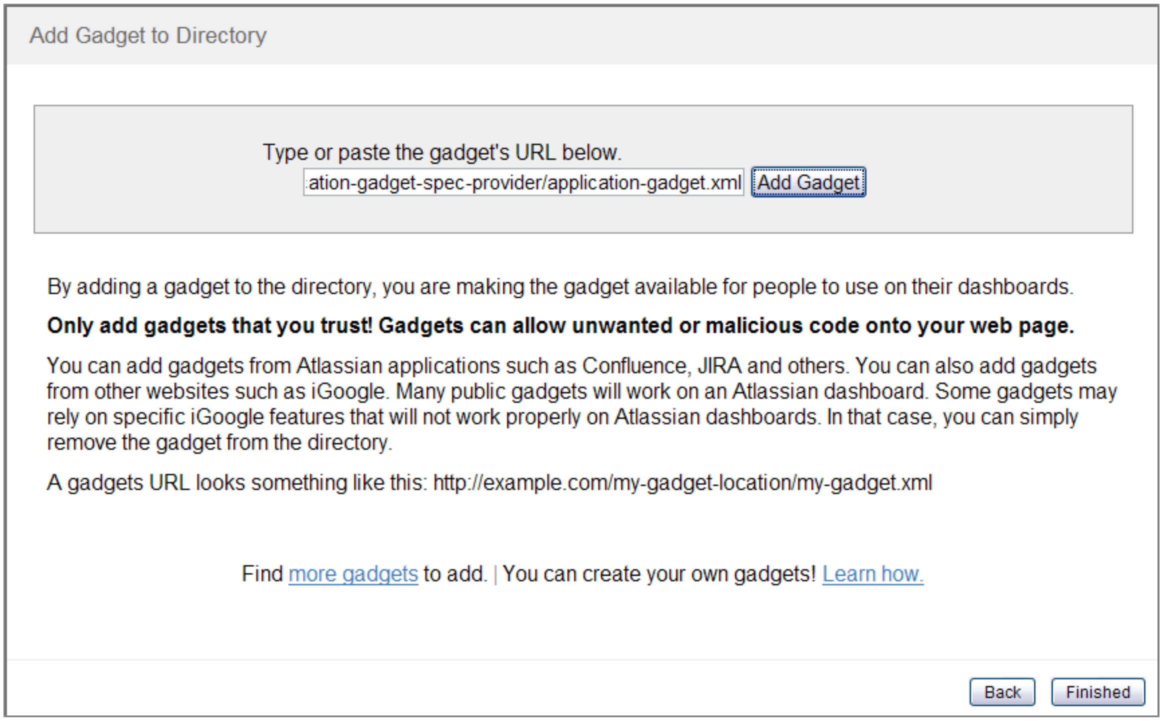
Hello World!
Gadget URL
Author unknown
description is here

I18n Test
Gadget URL
By Atlassian
Description not available

JIRA Category Gadget
Gadget URL
By Atlassian
JIRA Category Gadget description

JIRA Issues
Gadget URL
By Atlassian
A list of recently created JIRA Issues

Screenshot 2: Adding a gadget to the directory



Type or paste the gadget's URL below.

:ation-gadget-spec-provider/application-gadget.xml

By adding a gadget to the directory, you are making the gadget available for people to use on their dashboards.

Only add gadgets that you trust! Gadgets can allow unwanted or malicious code onto your web page.

You can add gadgets from Atlassian applications such as Confluence, JIRA and others. You can also add gadgets from other websites such as iGoogle. Many public gadgets will work on an Atlassian dashboard. Some gadgets may rely on specific iGoogle features that will not work properly on Atlassian dashboards. In that case, you can simply remove the gadget from the directory.

A gadgets URL looks something like this: http://example.com/my-gadget-location/my-gadget.xml

Find more gadgets to add. | You can create your own gadgets! Learn how.

Adding a Gadget that must be Installed as a Plugin

If the gadget must be installed as a plugin, you cannot add it via the gadget directory user interface.

Instead, you will need to follow your application's plugin management instructions.

RELATED TOPICS

Gadgets and Dashboards User Guide
Gadgets and Dashboards Administration Guide

- Adding a Gadget to the Directory of Available Gadgets
- Configuring OAuth
- Removing a Gadget from the Directory of Available Gadgets

Plugin management for Confluence
Plugin management for JIRA

Configuring OAuth

This page provides a brief introduction to the OAuth protocol and Atlassian's implementation of this protocol. It also shows you how to use your Atlassian application's 'OAuth Administration' page.

On this page:

- Introduction to OAuth
- Using Your Atlassian Application's OAuth Administration Page

Introduction to OAuth

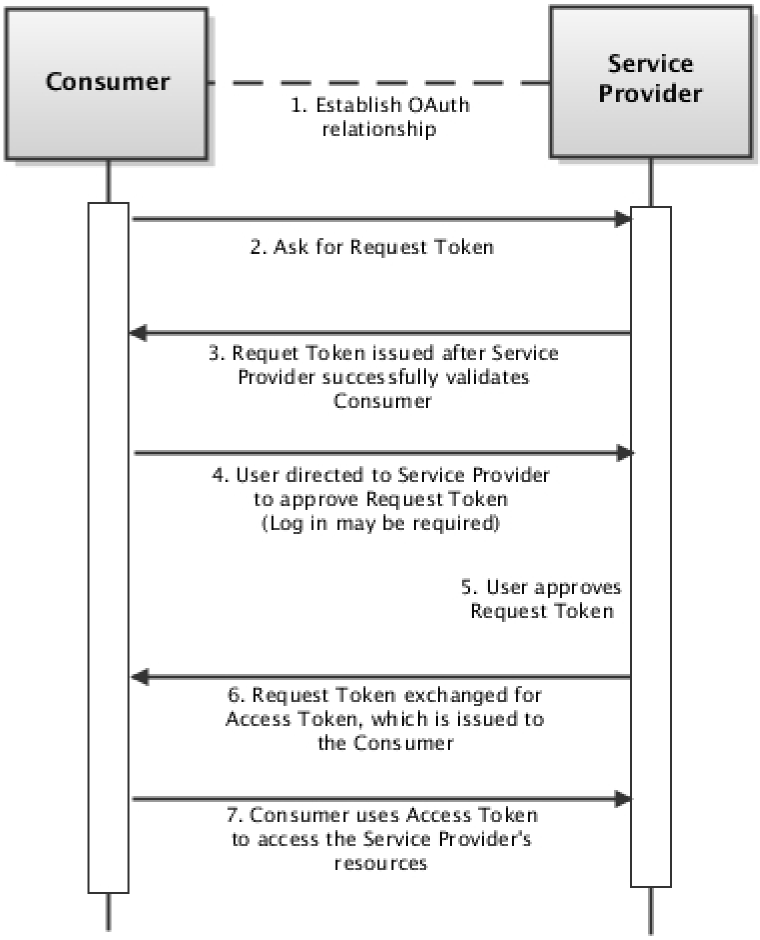
OAuth is a protocol that allows a web application such as JIRA, to share a finite set of its private resources and data (through JIRA dashboard gadgets, for example) with any other OAuth-compliant external application. These external applications could be another web application (such as a different JIRA installation or an iGoogle home page), a desktop application or a mobile device application and they must be accessible within your network or available on the Internet.

Using OAuth, you can access your JIRA site's data through a JIRA dashboard gadget on an iGoogle page.

The key advantage of OAuth is that a web application's resources can be shared without the web application having to hand out other important, confidential information, such as user authentication details. Instead, access to the web application's resources is handled via an '**access token**'. Access tokens typically define what web application resources can be accessed by an external application and the duration of this access. Access tokens are also dissociated from a user's account and password details.

In OAuth terminology, an application that shares its resources is known as a **service provider** and an application that accesses a service provider's resources is known as a **consumer**.

The OAuth protocol has the following typical workflow:


Consumer
Service Provider
1. Establish OAuth relationship
2. Ask for Request Token
3. Requet Token issued after Service Provider successfully validates Consumer
4. User directed to Service Provider to approve Request Token (Log in may be required)
5. User approves Request Token
6. Request Token exchanged for Access Token, which is issued to the Consumer
7. Consumer uses Access Token to access the Service Provider's resources

1. Establish an OAuth relationship between a consumer and a service provider. A relationship is typically 'unidirectional', in which one application is the consumer and the other is the service provider. However, it could be established on either the consumer or service provider application's OAuth configuration areas. This step only needs to be performed once, but it must be conducted so that your consumer and service provider can communicate via OAuth.
2. When the consumer needs to access the service provider's resources, the consumer asks the service provider for a '**request token**', which is the initial step in the consumer's request to access the service provider's resources. Like access tokens, request tokens are also dissociated from a user's authentication details.
3. The service provider first validates that this request came from one of its registered consumers (established in step 1), then creates a request token and gives it to the consumer.
4. Once the consumer receives the request token, the consumer asks the user to approve the request token by sending the user to the service provider. If the user is required to log in to the service provider, they will be prompted to do so.
5. The user is then prompted to either approve or deny the consumer's access to the request token.
6. Once the user approves access, the service provider exchanges the consumer's approved request token for an access token.

7. The consumer then uses the access token to access the service provider's permitted resources. The consumer can continue to access the service provider's resources until either the access token expires, or the user revokes the access token on the service provider. If the consumer needs to access the service provider's resources after either of these events occurs, then the OAuth workflow would start again from step 2 (above).
   ⓘ Access tokens issued by Atlassian applications expire after seven days.

Atlassian applications establish OAuth consumer and service provider relationships (step 1 of the workflow above) via their OAuth Administration page, which is described below. These relationships can be established using two approaches:

1. **Applications that can add others as a consumer of their resources** — Atlassian applications fit this category and any other application can be added as a consumer of its resources. Hence, if you are the administrator of an application (either an Atlassian or non-Atlassian one) and wish to access a different Atlassian application's resources, you can request the administrator of that Atlassian application to add your application as a consumer. The administrator of that Atlassian application can do this via the OAuth consumers configuration page.
2. **Applications that require you (the consumer) to initiate the OAuth consumer-service provider relationship** — Unlike Atlassian's applications, many 3rd-party applications may not provide the ability to add your application (either an Atlassian or non-Atlassian one), as a consumer of their application's resources. Instead, they will provide you with a '**consumer key**' and '**shared secret**', with which you can configure the OAuth consumer-service provider relationship. If you administer an Atlassian application and want it to consume the resources of such a 3rd-party application, you can establish this relationship via your Atlassian application's OAuth Service Providers configuration page.

Using Your Atlassian Application's OAuth Administration Page

An Atlassian application's OAuth administration page is available from within the application's administration area.

To use your Atlassian application's OAuth administration page,

- On the 'OAuth Administration' page:
  - Click the '**Consumers**' tab to configure consumer applications that will be accessing the resources of your Atlassian application. Refer to Configuring OAuth Consumers for more information.
  - Click the '**Consumer Info**' tab to view or edit your Atlassian application's Consumer information. Refer to Configuring OAuth Consumer Information for more information.
  - Click the '**Service Providers**' tab to configure service providers whose resources your Atlassian application will be consuming. Refer to Configuring OAuth Service Providers for more information.

Configuring OAuth Consumer Information

This page provides an introduction to the 'Consumer Info' section of an Atlassian application's OAuth administration page. It also shows you how to edit this information.
ⓘ You should be familiar with Atlassian's implementation of the OAuth protocol (refer to the Introduction to OAuth) before reading the information on this page.

On this page:

- Viewing or Editing Your Atlassian Application's OAuth Consumer Information

The 'Consumer Info' page contains information that identifies your Atlassian application (as a consumer) to a service provider application. If you require your Atlassian application to access the resources of a service provider, which could be another Atlassian application, the service provider could obtain and keep a record your Atlassian application's consumer information for identification purposes. As long as this record is kept by the service provider, your Atlassian application should have permission to access the service provider's resources via OAuth. If the service provider is another Atlassian application, then the administrator of that service provider can add your Atlassian application as a consumer via the 'Consumers ' tab of their OAuth administration page.

Be aware that your Atlassian application's OAuth consumer information is only useful if a service provider's OAuth administration features are capable of obtaining and recording your Atlassian application's consumer information. If, however, a service provider does not possess these features, then you must obtain a 'consumer key' and 'shared secret' from the service provider and establish an OAuth relationship with that service provider via your Atlassian application's 'Service Providers' tab.

An Atlassian application's OAuth consumer information consists of the following components, all of which are used by the service provider to identify your Atlassian application as a consumer of its resources:

- **Consumer Key** — For Atlassian applications, this usually takes the form of the application name, followed by a colon and then a series of digits. This is automatically generated by your Atlassian application and cannot be customised.
- **Name** — Any short, descriptive name that helps the administrator of your service provider identify your Atlassian application instance as a consumer. This component is editable.
- **Description** — Any brief description of your Atlassian application instance. Be aware that your service providers may choose to show this information to their users. This component is editable.
- **Public Key** — A complex string of characters (self-signed certificate) that is used to prove the identity of your Atlassian application to a service provider via a digital signature. This is automatically generated by your Atlassian application and cannot be customised.

- **Callback URL** — This is the URL that users will be directed to after they approve or deny the OAuth request. A consumer application usually supplies its own callback URL when receiving an OAuth request token. However, if the consumer application does not provide its own callback URL, this URL will be used instead.

Viewing or Editing Your Atlassian Application's OAuth Consumer Information

The OAuth Consumer Information page shows your Atlassian application's current consumer information and also allows you to edit it.

To view or edit your Atlassian application's OAuth consumer information,

1. On the 'OAuth Administration' page, click the '**Consumer Info**' tab to view your Atlassian application's current consumer information.
2. To edit the current consumer information:
   - Click the '**Edit**' link at the end of the consumer information page.
   - In the '**Name**' field, provide a short, descriptive name that will help the administrator of your service provider identify your Atlassian application instance as a consumer. This is the only editable field on this page that is mandatory.
   - In the '**Description**' field, provide a brief description of your Atlassian application instance.
     ⚠ Your service providers may choose to show this information to their users.
   - In the '**Callback URL**' field, add a URL that users will be directed to after they approve or deny the OAuth request. In most situations, this field can be left blank, as the consumer application usually supplies its own callback URL when receiving an OAuth request token.
   - Click the '**Update**' button to save your changes.

Configuring OAuth Consumers

This page provides an introduction to the 'consumers' list of an Atlassian application's OAuth administration page. It also shows you how to add other applications as consumers of your Atlassian application's resources and to how edit or remove existing consumers.
ℹ You should be familiar with Atlassian's implementation of the OAuth protocol (refer to the Introduction to OAuth) before reading the information on this page.

On this page:

The 'Consumers' page contains a list of OAuth-compliant applications that have access to your Atlassian application's resources. If a prospective consumer application, which could be another Atlassian application, provides its own consumer information and wants to access your Atlassian application's resources, you can use the Consumers page to add the application as a consumer of your Atlassian application's resources.

Upon first viewing your Atlassian application's Consumers page, you are not likely to see any consumers listed on it (unless there are other administrators of your application who have been adding consumers). On this page, you can add a prospective consumer to your Atlassian application, edit its details and remove it.

Adding a Consumer of Your Atlassian Application's Resources

The OAuth Consumer page allows you to add an application as a consumer of your Atlassian application's resources.

To add an application as a consumer of your Atlassian application's resources,

1. On the 'OAuth Administration' page, click the '**Consumers**' tab if necessary to view your current list of consumers.
2. Click the '**Add OAuth Consumer**' link.
3. If the prospective consumer is another Atlassian application that provides its own consumer information via URL and it is currently connected to your network or the Internet:
   - Enter the consumer's URL into the '**Consumer Base URL**' field.
   - Click the '**Add**' button. The other Atlassian application's consumer information is added as a new record into your Atlassian application's consumers list.
4. If the prospective consumer application is not an Atlassian application but you have access to its consumer key and public key (or self-signed certificate), then enter these details manually along with the appropriate additional consumer information, into the following fields:
   - **Consumer Key** — The application's consumer key. This field is mandatory and its contents must match the consumer key supplied by the application.
   - **Name** — Any descriptive name for the application. This field is mandatory, although the exact wording and format of the application's name is your choice. It is prudent to be accurate, however, as this name will be shown to your Atlassian application's users when they grant permission for a consumer to access your application's resources.
   - **Description** — A short description of the application. By convention, it is useful to include the name of the application and if applicable, its URL, somewhere within the description.
   - **Public Key** — The application's public key or self-signed certificate. This field is mandatory and its contents must match the public key or self-signed certificate supplied by the consumer.
   - **Callback URL** — A consumer application usually supplies its own callback URL when receiving an OAuth request token. Hence, this field is usually left blank. However, if the consumer application does not provide its own callback URL, this URL will be used instead.
   - Click the '**Add**' button. If all the information is valid (in particular, the public key format), then the application's consumer information is added as a new record into your Atlassian application's consumers list.

ⓘ At any time, you can click '**Return to Consumer List**' to cancel adding consumer details to your Atlassian application.

Editing a Consumer of Your Atlassian Application's Resources

In addition to adding consumers, the OAuth Consumer page allows you to edit the details of existing consumers of your Atlassian application's resources.

ⓘ It is not possible to modify an existing consumer application's consumer key. If you must do this (because a consumer application's consumer key has changed), you will need to add that application as a new consumer of your Atlassian application's resources.

To edit the details of an existing consumer of your Atlassian application's resources,

1. On the 'OAuth Administration' page, click the '**Consumers**' tab if necessary to view your current list of consumers.
2. Click the '**Edit**' link next to the consumer whose details you wish to edit.
3. The following consumer details can be modified:
   - **Name** — Any descriptive name for the application. This field is mandatory, although the exact wording and format of the application's name is your choice. It is prudent to be accurate, however, as this name will be shown to your Atlassian application's users when they grant permission for a consumer to access your application's resources.
   - **Description** — A short description of the application. By convention, it is useful to include the name of the application and if applicable, its URL, somewhere within the description.
   - **Public Key** — The application's public key or self-signed certificate. This field is mandatory and its contents must match the public key or self-signed certificate supplied by the consumer.
   - **Callback URL** — A consumer application usually supplies its own callback URL when receiving an OAuth request token. Hence, this field is usually left blank. However, if the consumer application does not provide its own callback URL, this URL will be used instead.
4. Click the '**Add**' button. If all the modified information is valid (in particular, the public key format), then the application's consumer information is added as a new record into your Atlassian application's consumers list.

ⓘ At any time, you can click '**Return to Consumer List**' to cancel editing a consumer's details.

Removing a Consumer of Your Atlassian Application's Resources

In addition to adding consumers, the OAuth Consumer page allows you to remove existing consumers of your Atlassian application's resources.

⚠ Once you remove a consumer application from your Altassian application's consumer list, then that application will no longer be able to access your Atlassian application's resources.

To remove an existing consumer of your Atlassian application's resources,

1. On the 'OAuth Administration' page, click the '**Consumers**' tab if necessary to view your current list of consumers.
2. Click the '**Remove**' link next to the consumer you wish to remove.
   You will be prompted to confirm this action. Be aware that any request tokens created by this consumer application will be removed and consequently, this application will no longer be able to access your Atlassian application's resources.
3. The consumer application is removed from your Atlassian application's consumer list.

Configuring OAuth Service Providers

This page provides an introduction to the 'service providers' list of an Atlassian application's OAuth administration page. It also shows you how to add other applications as service providers to your Atlassian application and to how edit or remove existing service providers.

You should be familiar with Atlassian's implementation of the OAuth protocol (refer to the Introduction to OAuth) before reading the information on this page.

On this page:

- Adding a Service Provider of Resources to Your Atlassian Application
- Removing a Service Provider of Resources to Your Atlassian Application

The 'Service Providers' page contains a list of OAuth-compliant applications whose resources can be accessed by your Atlassian application. If a prospective service provider issues you with consumer key and shared secret to access their application's resources via OAuth, you can add these service provider details using the Service Providers page, thereby establishing an OAuth relationship between your application as a consumer and their application as a service provider.

Atlassian applications as service providers

Atlassian applications do not issue shared secrets. Hence, if your prospective service provider is another Atlassian application, do not use the Service Providers page to add that application as a service provider.

Instead, contact the administrator of the prospective Atlassian service provider application, requesting that they add your application as a consumer of their application's resources.

- If your application is an Atlassian application too, you can provide the administrator of the service provider application with your application's URL. The administrator can then add your application as a consumer via the **Consumer Base URL** field of their OAuth Consumers page.
- If your application is not an Atlassian application, you will need to provide the administrator of the service provider application with your application's consumer key and public key (or self-signed certificate). The administrator will then need to add your application as a consumer manually.

Upon first viewing your Atlassian application's Service Providers page, you are not likely to see any service providers listed on it (unless there are other administrators of your application who have been adding service providers). On this page, you can add a prospective service provider to your Atlassian application or remove an existing one. It is not possible to edit an existing service provider's details.

Adding a Service Provider of Resources to Your Atlassian Application

The OAuth Service Provider page allows you to add an application as a service provider of resources to your Atlassian application.

To add an application as a service provider of resources to your Atlassian application,

1. On the 'OAuth Administration' page, click the '**Service Providers**' tab if necessary to view your current list of service providers.
2. Click the '**Add Service Providers**' link.
3. Complete the following fields which relate to the service provider application that you are adding:
   - **Service Provider Name** — Any descriptive name for the application. This field is mandatory, although the exact wording and format of the application is your choice.
   - **Consumer Key** — The application's consumer key. This field is mandatory and its contents must match the consumer key supplied by the application.
   - **Shared Secret** — The share secret supplied to you by the service provider. This secret is used to digitally sign all request tokens sent from your application as a consumer to the service provider. Hence, this information should remain known to you and your service provider only.
   - **Description** — A short description of the application. By convention, it is useful to include the name of the application and if applicable, its URL, somewhere within the description.
4. Click the '**Add**' button and the service provider is added as a new record into your Atlassian application's service providers list.

> At any time, you can click '**Return to Service Provider List**' to cancel adding service provider details to your Atlassian application.

Removing a Service Provider of Resources to Your Atlassian Application

In addition to adding service providers, the OAuth Service Provider page allows you to remove existing service providers to your Altassian application.

Once you remove a service provider application from your Altassian application's service providers list, then your Atlassian application will no longer be able to access that service provider's resources.

To remove a service provider of resources to your Atlassian application,

1. On the 'OAuth Administration' page, click the '**Service Providers**' tab if necessary to view your current list of service providers.
2. Click the '**Remove**' link next to the service provider you wish to remove.
   You will be prompted to confirm this action. Be aware that any tokens which relate to this service provider will be removed and consequently, your application will no longer be able to access your service provider's resources.
3. The service provider is removed from your Atlassian application's service provider list.

Removing a Gadget from the Directory of Available Gadgets

This page tells you how to remove a gadget from the gadget directory.

On this page:

The gadget directory displays all the gadgets that are available to users. These are the gadgets that users can add to their dashboard.

There are two types of gadgets: those that were installed into the application as plugins, and those that were added as simple gadgets. You will need to remove them in different ways.

Removing a Gadget that was Not Installed as a Plugin

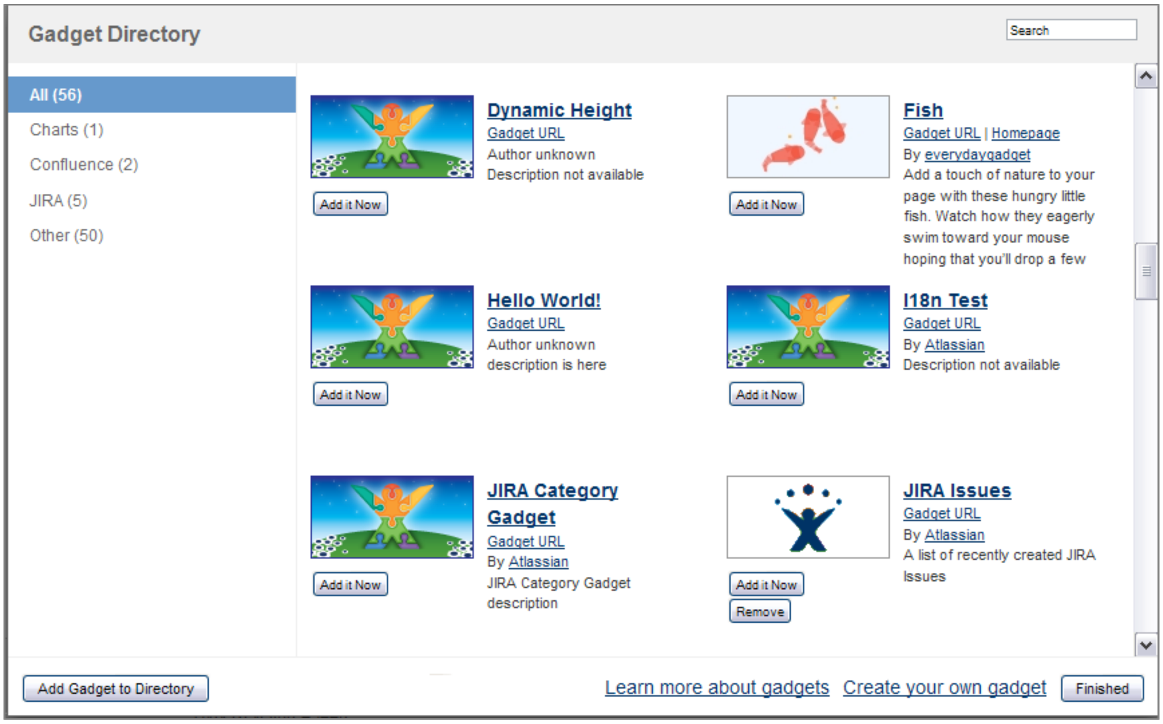
If the gadget was added to the directory as a simple URL, not as a plugin, then you can simply remove it from the directory.

You need to have administrator privileges to remove a gadget from the directory.

To remove a gadget from your directory,

1. Go to the dashboard by clicking the '**Dashboard**' link or the '**Home**' link at the top left of the screen.
2. The dashboard will appear. Click '**Add Gadget**'.
3. The '**Add Gadget**' screen appears, showing the list of gadgets in your directory. See screenshot 1 below. Find the gadget that you want to remove.
4. Click '**Remove**' under the gadget you want to remove.
   You will only see this button if you have administrator permissions on your application/dashboard. Also, it will only appear for gadgets that are *not* installed as plugins.
5. Click '**OK**' to confirm the removal of the gadget.

Screenshot 1: Gadget directory with 'Remove' button



Dynamic Height
Gadget URL
Author unknown
Description not available

Fish
Gadget URL | Homepage
By everydaygadget
Add a touch of nature to your page with these hungry little fish. Watch how they eagerly swim toward your mouse hoping that you'll drop a few

Hello World!
Gadget URL
Author unknown
description is here

I18n Test
Gadget URL
By Atlassian
Description not available

JIRA Category Gadget
Gadget URL
By Atlassian
JIRA Category Gadget description

JIRA Issues
Gadget URL
By Atlassian
A list of recently created JIRA Issues

Removing a Gadget that was Installed as a Plugin

If the gadget was installed into the application as a plugin, there will not be a 'Remove' button in the gadget directory.

You will need to remove the plugin that supplies the gadget in your application. Follow your application's plugin management instructions. For example:

- Plugin management for Confluence
- Plugin management for JIRA

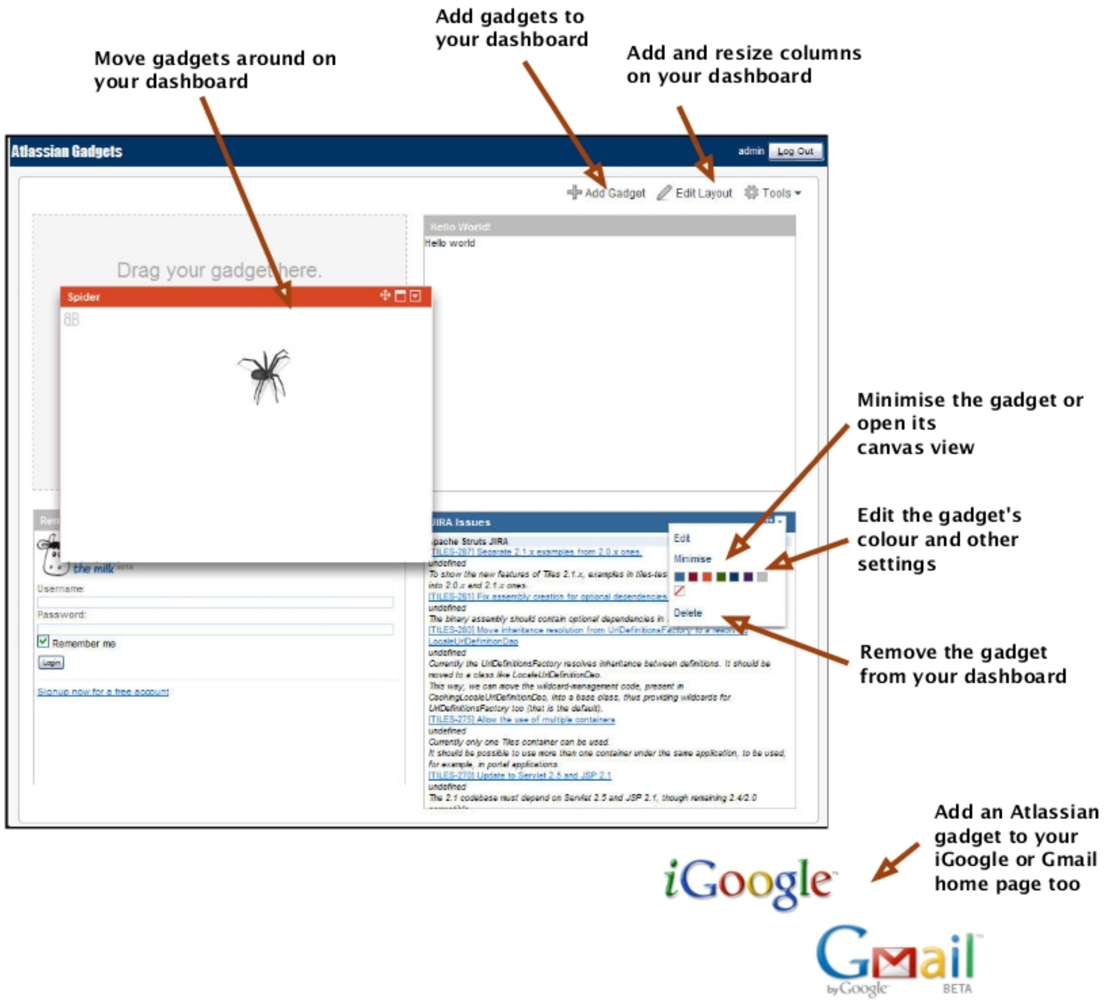
RELATED TOPICS

Gadgets and Dashboards User Guide
Gadgets and Dashboards Administration Guide

- Adding a Gadget to the Directory of Available Gadgets
- Configuring OAuth
- Removing a Gadget from the Directory of Available Gadgets

Gadgets and Dashboards User Guide

Atlassian Dashboard and Gadgets User Guide


Move gadgets around on your dashboard
Add gadgets to your dashboard
Add and resize columns on your dashboard
Atlassian Gadgets
admin
Log Out
Add Gadget
Edit Layout
Tools
Drag your gadget here.
Spider
Hello World!
Hello world
JIRA Issues
Edit
Minimise
Delete
Username:
Password:
☑ Remember me
Login
Signup now for a free account
Minimise the gadget or open its canvas view
Edit the gadget's colour and other settings
Remove the gadget from your dashboard
Add an Atlassian gadget to your iGoogle or Gmail home page too
iGoogle
Gmail by Google BETA

pache Struts JIRA
[TILES-287] Separate 2.1.x examples from 2.0.x ones.
undefined
To show the new features of Tiles 2.1.x, examples in tiles-tes
into 2.0.x and 2.1.x ones.
[TILES-281] Fix assembly creation for optional dependencies
undefined
The binary assembly should contain optional dependancies in
[TILES-280] Move inheritance resolution from UrlDefinitionsFactory to a reaching LocaleUrlDefinitionDao
undefined
Currently the UrlDefinitionsFactory resolves inheritance between definitions. It should be moved to a class like LocaleUrlDefinitionDao.
This way, we can move the wildcard-management code, present in CachingLocaleUrlDefinitionDao, into a base class, thus providing wildcards for UrlDefinitionsFactory too (that is the default).
[TILES-275] Allow the use of multiple containers
undefined
Currently only one Tiles container can be used.
It should be possible to use more than one container under the same application, to be used, for example, in portal applications.
[TILES-270] Update to Servlet 2.5 and JSP 2.1
undefined
The 2.1 codebase must depend on Servlet 2.5 and JSP 2.1, though remaining 2.4/2.0

This guide explains the the functionality of the Atlassian dashboards that are based on the new Atlassian Gadgets framework.

At this time, **JIRA 4.0 beta** is the only Atlassian application that supports the new Atlassian Gadgets framework.

Every application that presents a dashboard based on the new gadgets framework will support at least the functionality described below.

Some applications may have additional functionality. For example, JIRA will allow you to create multiple dashboards on separate tabs. The extra functionality is described in the application-specific user guides.

> **Writing your own Gadget**
> It is not difficult to write a gadget, especially if you have some HTML and JavaScript skills. You can create a gadget that will work on an Atlassian dashboard. Take a look at our guide to getting started with gadget development.

User Guide Table of Contents

Changing the Layout of your Dashboard

Changing the Look and Behaviour of a Gadget

Adding an Atlassian Gadget to iGoogle and Other Web Sites

RELATED TOPICS

Introduction to Atlassian Gadgets and Dashboards
Gadgets and Dashboards Administration Guide
Gadgets and Dashboards Documentation

Allowing Other Applications to Access Data on Your Behalf

This page tells you how to access the data and resources on an Atlassian application from another application via OAuth.

On this page:

- Allowing a Consumer Application to Access the Resources of an Atlassian Servicer Provider
- Revoking Access Tokens

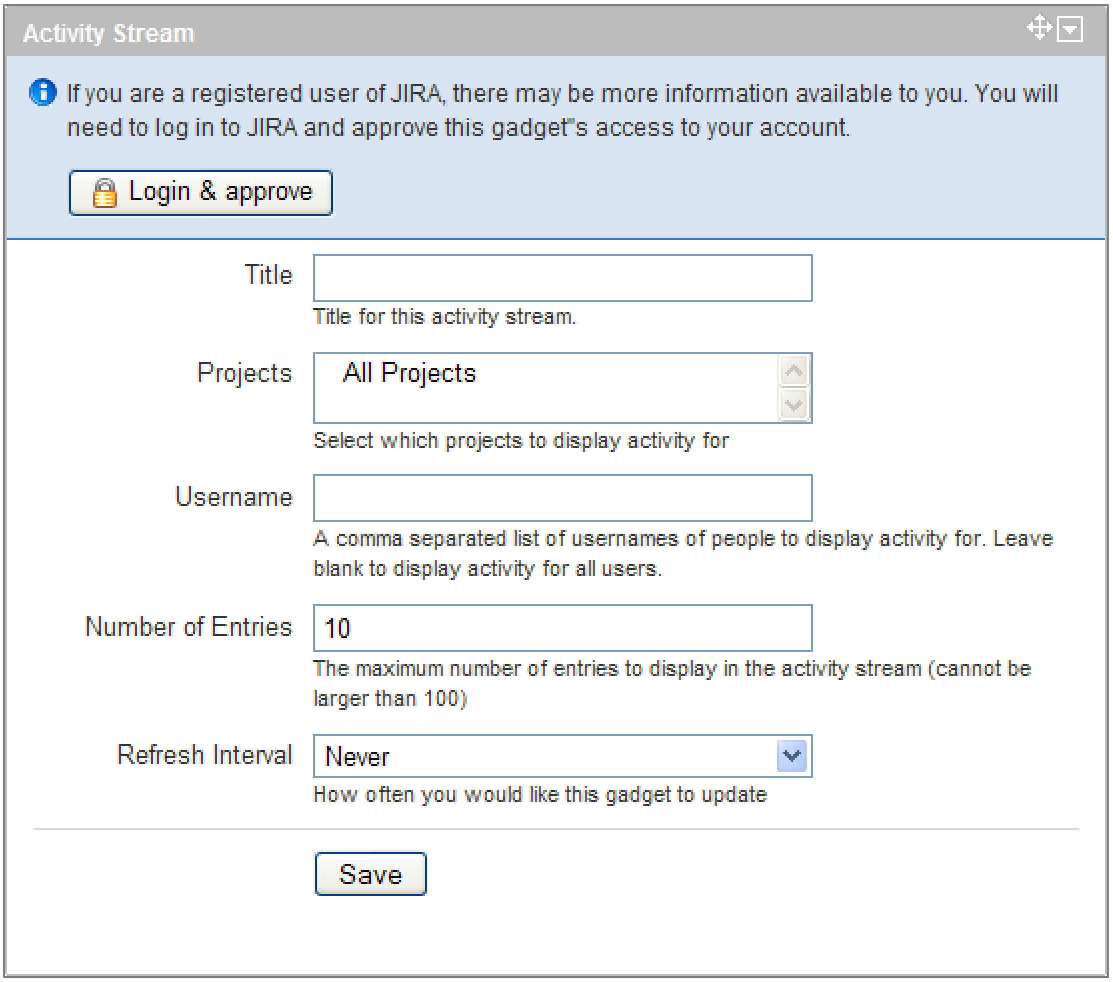
Allowing a Consumer Application to Access the Resources of an Atlassian Servicer Provider

The following procedure uses an Atlassian gadget to demonstrate how an application (the consumer) can access the resources of your Atlassian application via OAuth, on your behalf. For the consumer application to gain access to your Atlassian application's resources, you must have a user account with your Atlassian application.

To allow a consumer application to access your Atlassian application's resources via a gadget,

1. Ensure that the consumer application has been configured as a consumer of your Atlassian application's resources via OAuth.
2. Add a gadget of your Atlassian application to the consumer application. For more information on how to do this, please refer to the following topics:
    - Adding an Atlassian Gadget to iGoogle and Other Web Sites
    - Adding a Gadget to your Dashboard
3. Once your gadget has been added to the consumer application, the gadget displays a message along its top edge, indicating that if you are a registered user of your Atlassian application, more of its resources may be available to you.
   ℹ If you do not see this message, you may need to log in to the consumer application.
4. Click the '**Login & approve**' button.
5. A new window is displayed with your Atlassian application's login screen. Enter your Atlassian application's user account login details into this page.
   ℹ If you are already logged in to your Atlassian application on the same computer, this step might be skipped.
6. If your login credentials were recognised by your Atlassian application, the 'Request for Access' page is displayed, requesting that you approve or deny the consumer application access to your Atlassian application's resources. Click the '**Approve Access**' button to approve this access.
7. The Request for Access window closes and the message along the top of your gadget on the consumer application will be removed. This gadget will now be able to access the resources of your Atlassian application, based on your Atlassian application's user account credentials.

Screenshot: Approving a Gadget's Access to your Resources



If you are a registered user of JIRA, there may be more information available to you. You will need to log in to JIRA and approve this gadget"s access to your account.

Title:
Title for this activity stream.

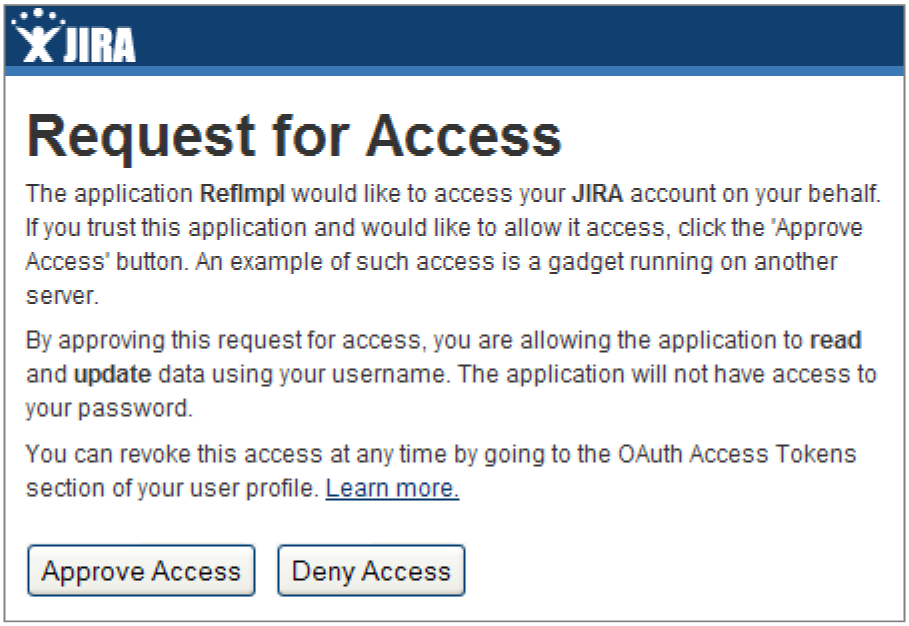
Projects: All Projects
Select which projects to display activity for

Username:
A comma separated list of usernames of people to display activity for. Leave blank to display activity for all users.

Number of Entries: 10
The maximum number of entries to display in the activity stream (cannot be larger than 100)

Refresh Interval: Never
How often you would like this gadget to update

Screenshot: Request for Access Message



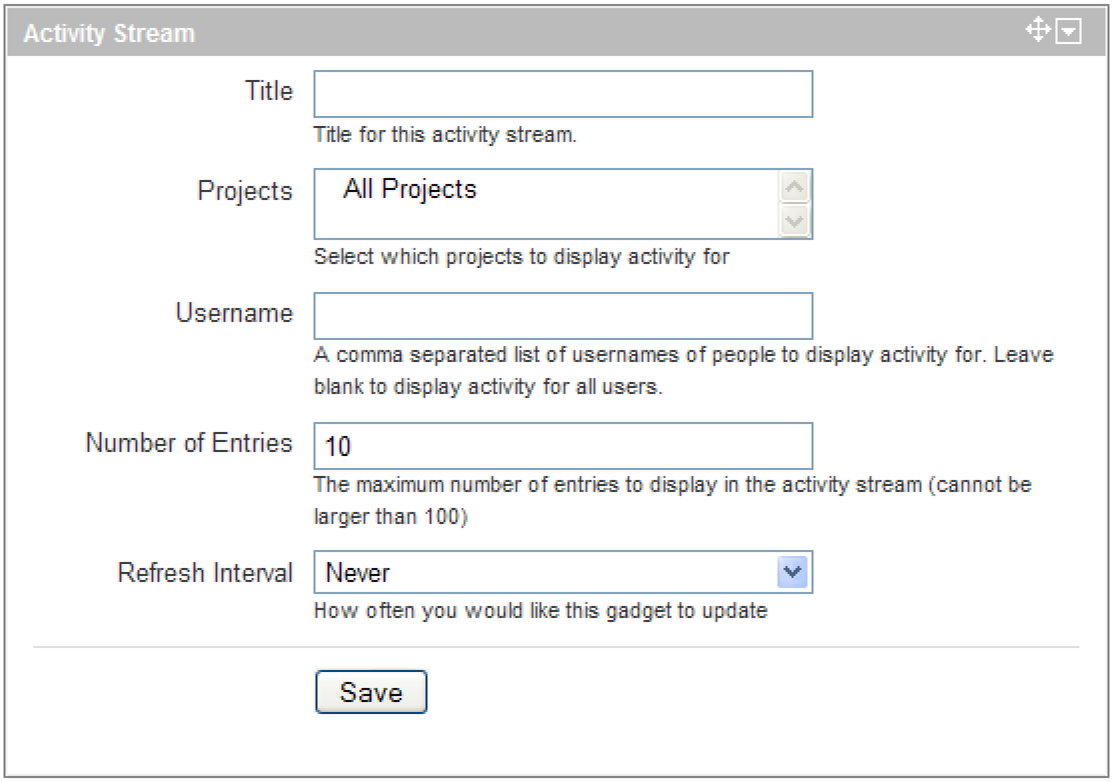
Request for Access

The application **RefImpl** would like to access your **JIRA** account on your behalf. If you trust this application and would like to allow it access, click the 'Approve Access' button. An example of such access is a gadget running on another server.

By approving this request for access, you are allowing the application to **read** and **update** data using your username. The application will not have access to your password.

You can revoke this access at any time by going to the OAuth Access Tokens section of your user profile. Learn more.

Screenshot: A Gadget after OAuth Access has been Approved



Activity Stream

Title	 Title for this activity stream.
Projects	All Projects Select which projects to display activity for
Username	 A comma separated list of usernames of people to display activity for. Leave blank to display activity for all users.
Number of Entries	10 The maximum number of entries to display in the activity stream (cannot be larger than 100)
Refresh Interval	Never How often you would like this gadget to update

Revoking Access Tokens

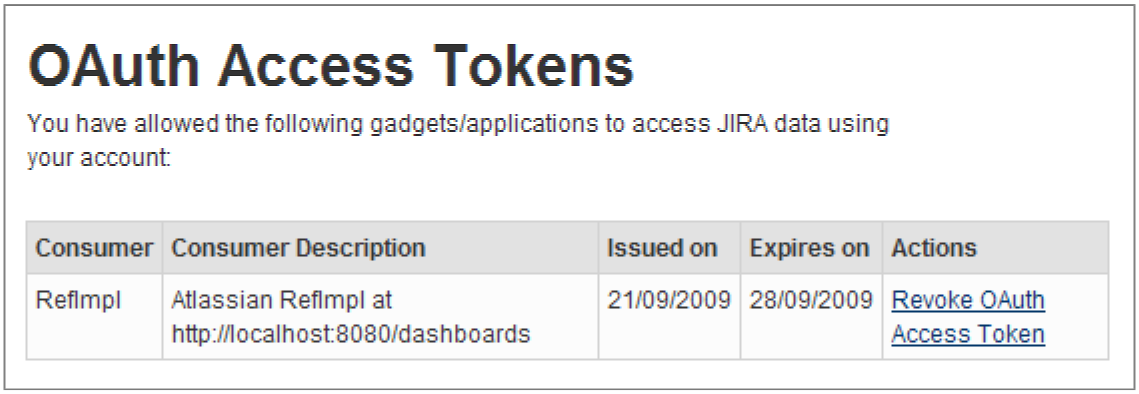
Once you have allowed a consumer application to access an Atlassian service provider's resources on your behalf, the access token issued to the consumer will be accessible from your account area of the Atlassian service provider application. This area shows all access tokens issued by the service provider to all consumer applications, to whom you have granted access to your service provider's resources.

You can revoke any access token from this area. Doing this prevents the applicable consumer application from accessing the service provider's resources on your behalf until you grant that consumer access again.

To revoke an access token,

1. Go to your 'OAuth Access Tokens' page of the Atlassian service provider application.
2. Click the 'Revoke OAuth Access Token' link next to the access token of the consumer application whose access you wish to revoke.
3. The consumer's access token is removed from your 'OAuth Access Tokens' list.

Screenshot: OAuth Access Tokens page



OAuth Access Tokens

You have allowed the following gadgets/applications to access JIRA data using your account:

Consumer	Consumer Description	Issued on	Expires on	Actions
RefImpl	Atlassian RefImpl at http://localhost:8080/dashboards	21/09/2009	28/09/2009	Revoke OAuth Access Token

Adding a Gadget to your Dashboard

This page tells you how to add a gadget to an Atlassian application's dashboard. These instructions assume that you are using an application that supports the new Atlassian Gadgets framework.

On this page:

- Adding a Gadget to your Atlassian Application's Dashboard
- Differing Look and Functionality
- Permission to Update a Dashboard
- Who Will See the Gadgets you have Added to the Dashboard?

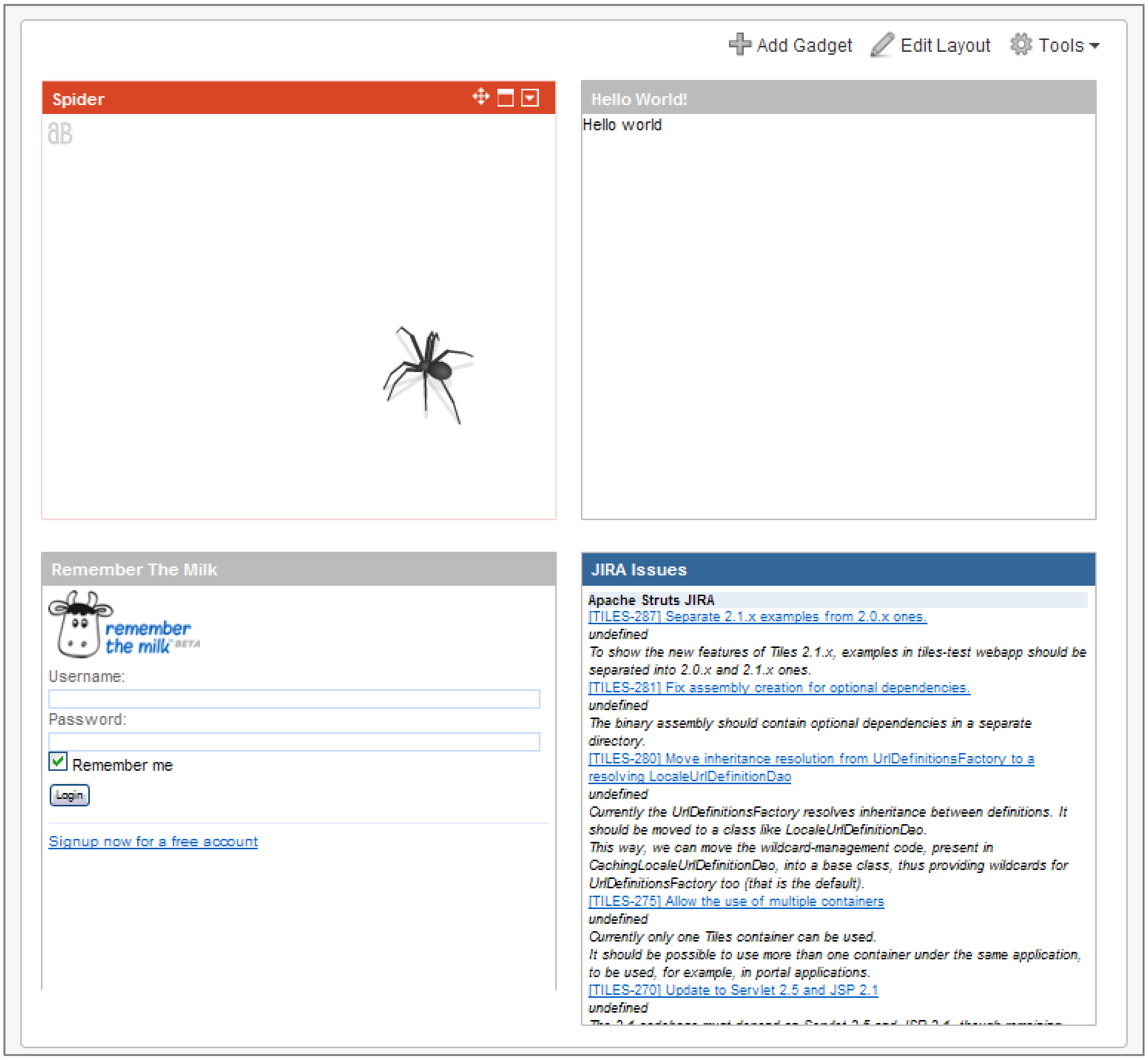
Adding a Gadget to your Atlassian Application's Dashboard

You can add a gadget from the directory of gadgets that are available to your Atlassian application.

To add a gadget to your Atlassian dashboard,

1. Go to the dashboard by clicking the '**Dashboard**' link or the '**Home**' link at the top left of the screen.
2. The dashboard will appear, looking something like screenshot 1 below. Click '**Add Gadget**'.
3. The '**Add Gadget**' screen will display a list of available gadgets in your gadget directory, as shown in screenshot 2 below. Find the gadget you want, using one or more of the following tools:
    - Use the scroll bar on the right to move up and down the list of gadgets.
    - Select a category in the left-hand panel to display only gadgets in that category.
    - Start typing a key word for your gadget in the '**Search**' textbox. The list of gadgets will change as you type, showing only gadgets that match your search term.
4. When you have found the gadget you want, click the '**Add it Now**' button to add the gadget to your dashboard.

Screenshot 1: An Atlassian dashboard

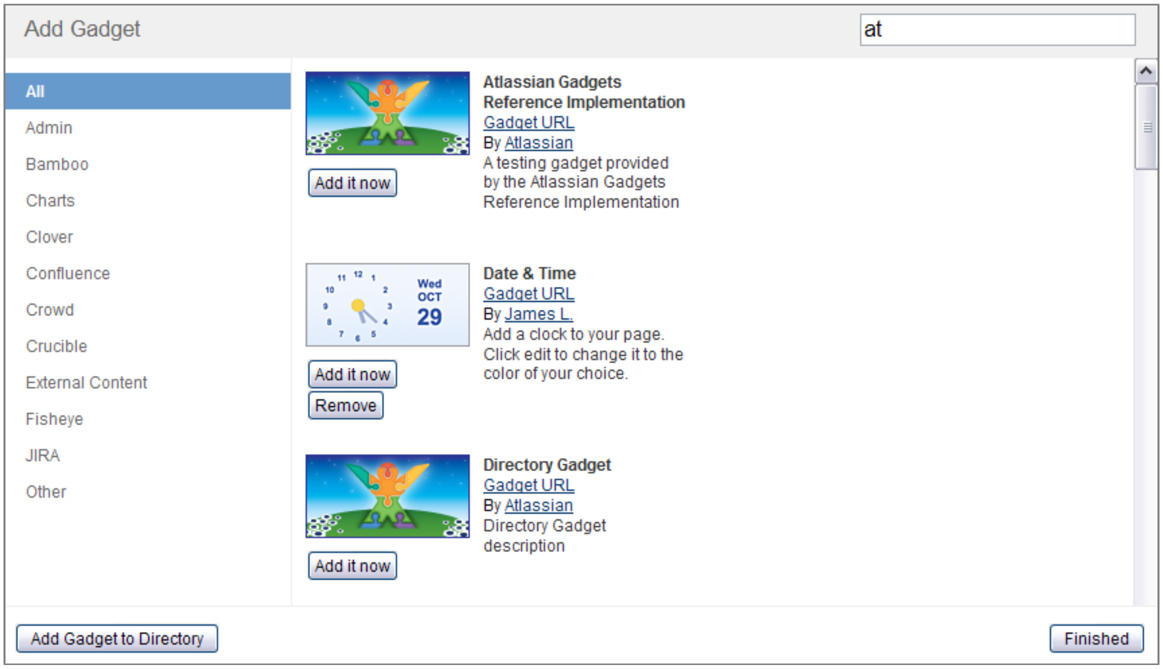


Hello world

Apache Struts JIRA

[TILES-287] Separate 2.1.x examples from 2.0.x ones.
undefined
To show the new features of Tiles 2.1.x, examples in tiles-test webapp should be separated into 2.0.x and 2.1.x ones.
[TILES-281] Fix assembly creation for optional dependencies.
undefined
The binary assembly should contain optional dependencies in a separate directory.
[TILES-280] Move inheritance resolution from UrlDefinitionsFactory to a resolving LocaleUrlDefinitionDao
undefined
Currently the UrlDefinitionsFactory resolves inheritance between definitions. It should be moved to a class like LocaleUrlDefinitionDao.
This way, we can move the wildcard-management code, present in CachingLocaleUrlDefinitionDao, into a base class, thus providing wildcards for UrlDefinitionsFactory too (that is the default).
[TILES-275] Allow the use of multiple containers
undefined
Currently only one Tiles container can be used.
It should be possible to use more than one container under the same application, to be used, for example, in portal applications.
[TILES-270] Update to Servlet 2.5 and JSP 2.1
undefined
The 2.1 codebase must depend on Servlet 2.5 and JSP 2.1, though remaining

Screenshot 2: Adding a gadget from the gadget directory



Atlassian Gadgets Reference Implementation
Gadget URL
By Atlassian
A testing gadget provided by the Atlassian Gadgets Reference Implementation

Date & Time
Gadget URL
By James L.
Add a clock to your page. Click edit to change it to the color of your choice.

Directory Gadget
Gadget URL
By Atlassian
Directory Gadget description

Adding a Gadget to the Directory of Available Gadgets
You need to have administrator privileges to add a gadget to the list of available gadgets. If you have permission to add gadgets to and remove gadgets from the directory itself, you will see the '**Add Gadget to Directory**' and '**Remove**' buttons on the 'Add Gadget' screen. Please refer to the Gadgets and Dashboards Administration Guide.

Differing Look and Functionality

The screenshots on this page show the basic Atlassian dashboard layout. The layout will be similar in all applications that support the new Atlassian Gadgets framework. However, there may be slight differences in the look and feel for a particular application and some applications may have additional functionality. For example, JIRA will allow you to create multiple dashboards on separate tabs. The extra functionality is described in the documentation for the application concerned.

Permission to Update a Dashboard

Your ability to update a dashboard depends on the application hosting the dashboard. Most applications will require you to log in before you can update the dashboard. Your ability to update the dashboard may then depend on the permissions assigned to your username.

Who Will See the Gadgets you have Added to the Dashboard?

This depends on the application hosting the dashboard:

- In most cases, you will need to log in to the application before you can update the dashboard, and your updates will apply only to your personal dashboard.
- Some applications allow dashboards that are shared by groups of people. If you have permission to update a shared dashboard, the other people sharing the dashboard will see your changes too.
- If you have permission to update an application's default dashboard, your changes to the default dashboard will be seen by anyone who views that dashboard.

RELATED TOPICS

Gadgets and Dashboards Administration Guide
Gadgets and Dashboards User Guide

- Allowing Other Applications to Access Data on Your Behalf
- Adding a Gadget to your Dashboard
- Removing a Gadget from your Dashboard
- Changing the Layout of your Dashboard
- Changing the Look and Behaviour of a Gadget

- Adding an Atlassian Gadget to iGoogle and Other Web Sites

Removing a Gadget from your Dashboard

This page tells you how to remove a gadget from an Atlassian application's dashboard. These instructions assume that you are using an application that supports the new Atlassian Gadgets framework.

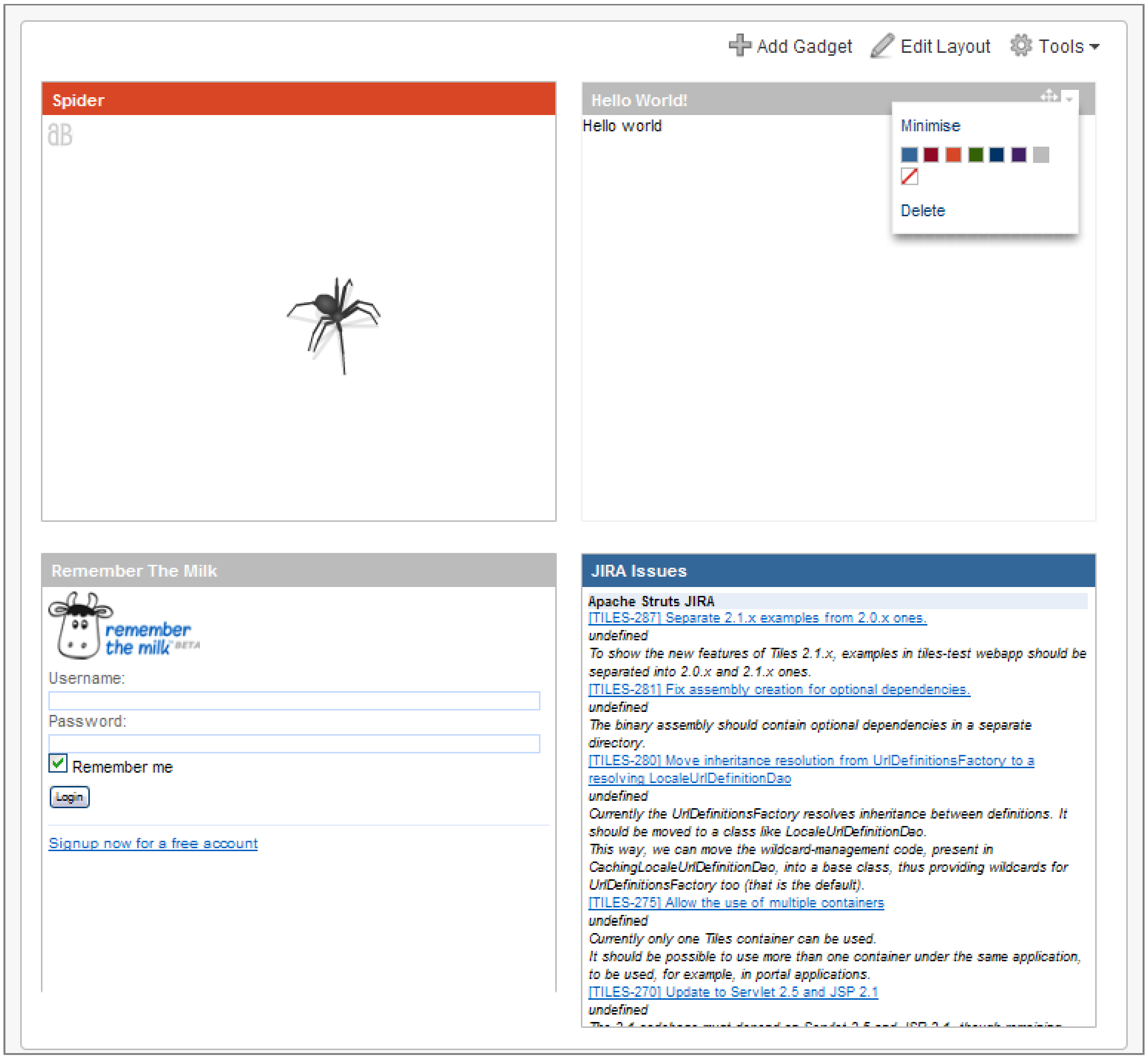
On this page:

Removing a Gadget from your Dashboard

To remove a gadget from your Atlassian dashboard,

1. Go to your application's dashboard. In most cases, you will do this by clicking the '**Dashboard**' link or the '**Home**' link at the top left of the application screen.
2. The dashboard will appear, looking something like the screenshot below. Find the gadget you want to remove.
3. Hover your cursor over the gadget and click the dropdown menu icon ▼ at top right of the gadget frame.
4. The dropdown menu will appear, as shown in the screenshot below. Click the '**Delete**' option in the menu.
5. A popup dialogue will appear, asking you to confirm the deletion. Click '**OK**'.
6. The gadget will disappear from your dashboard and the other gadgets will move to fill its place.

Screenshot: Deleting a gadget from a dashboard



Hello world

Apache Struts JIRA

[TILES-287] Separate 2.1.x examples from 2.0.x ones.
undefined
To show the new features of Tiles 2.1.x, examples in tiles-test webapp should be separated into 2.0.x and 2.1.x ones.

[TILES-281] Fix assembly creation for optional dependencies.
undefined
The binary assembly should contain optional dependencies in a separate directory.

[TILES-280] Move inheritance resolution from UrlDefinitionsFactory to a resolving LocaleUrlDefinitionDao
undefined
Currently the UrlDefinitionsFactory resolves inheritance between definitions. It should be moved to a class like LocaleUrlDefinitionDao.
This way, we can move the wildcard-management code, present in CachingLocaleUrlDefinitionDao, into a base class, thus providing wildcards for UrlDefinitionsFactory too (that is the default).

[TILES-275] Allow the use of multiple containers
undefined
Currently only one Tiles container can be used.
It should be possible to use more than one container under the same application, to be used, for example, in portal applications.

[TILES-270] Update to Servlet 2.5 and JSP 2.1
undefined

Removing a Gadget from the Directory of Available Gadgets
You need to have administrator privileges to remove a gadget from the list of available gadgets. Please refer to the Gadgets and Dashboards Administration Guide.

Permission to Remove a Gadget

Your ability to update a dashboard depends on the application hosting the dashboard. Most applications will require you to log in before you can update the dashboard. Your ability to update the dashboard may then depend on the permissions assigned to your username.

Will the Removal of the Gadget Affect other People?

This depends on the application hosting the dashboard:

- In most cases, you will need to log in to the application before you can update the dashboard, and your updates will apply only to your personal dashboard.
- Some applications allow dashboards that are shared by groups of people. If you have permission to update a shared dashboard, the other people sharing the dashboard will see your changes too.
- If you have permission to update an application's default dashboard, your changes to the default dashboard will be seen by anyone who views that dashboard.

RELATED TOPICS

Gadgets and Dashboards Administration Guide
Gadgets and Dashboards User Guide

- Allowing Other Applications to Access Data on Your Behalf

- Adding a Gadget to your Dashboard
- Removing a Gadget from your Dashboard
- Changing the Layout of your Dashboard
- Changing the Look and Behaviour of a Gadget
- Adding an Atlassian Gadget to iGoogle and Other Web Sites

Changing the Layout of your Dashboard

This page tells you how to move a gadget to different position on an Atlassian application's dashboard. These instructions assume that you are using an application that supports the new Atlassian Gadgets framework.

On this page:

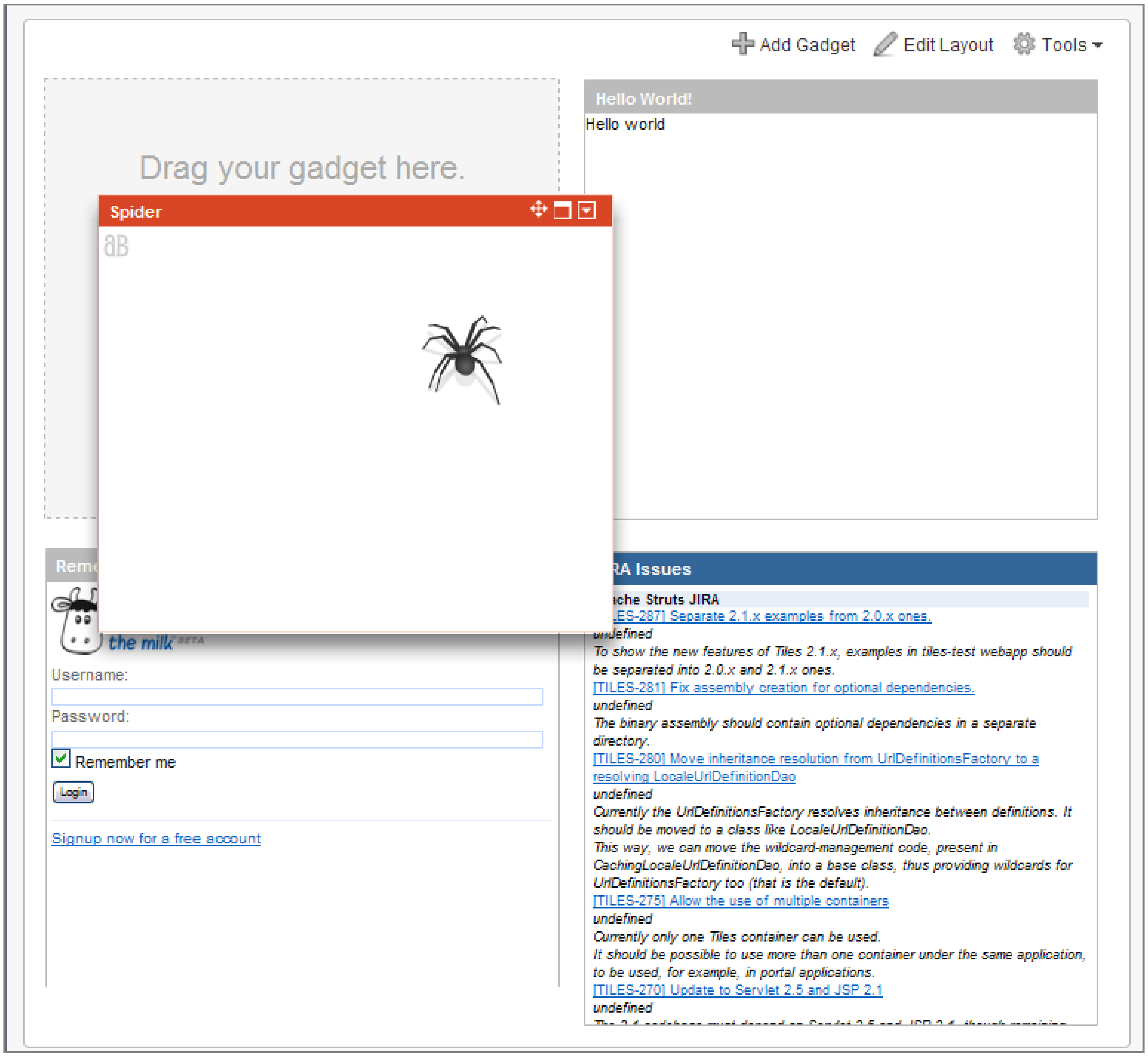
Moving Gadgets Around on your Dashboard

You can drag a gadget and drop it onto a different place on your dashboard.

To move a gadget,

1. Go to your application's dashboard. In most cases, you will do this by clicking the '**Dashboard**' link or the '**Home**' link at the top left of the application screen.
2. Move your mouse pointer over the top frame of the gadget, where the pointer changes to a 'move' icon, such as a **hand** or a set of **arrows** ✥
3. Click the top frame of the gadget and move your mouse to drag the gadget. As you move it across the dashboard, empty frames will open in places where you can choose to place the gadget. The frames contain the words '**Drag your gadget here**'. The gadget you are moving will become slightly transparent, so that you can see the dashboard beneath it. See the screenshot below.
4. Release the mouse button to drop the gadget where you want it.

Screenshot: Moving a gadget



LES-287] Separate 2.1.x examples from 2.0.x ones.
undefined
To show the new features of Tiles 2.1.x, examples in tiles-test webapp should be separated into 2.0.x and 2.1.x ones.
[TILES-281] Fix assembly creation for optional dependencies.
undefined
The binary assembly should contain optional dependencies in a separate directory.
[TILES-280] Move inheritance resolution from UrlDefinitionsFactory to a resolving LocaleUrlDefinitionDao
undefined
Currently the UrlDefinitionsFactory resolves inheritance between definitions. It should be moved to a class like LocaleUrlDefinitionDao.
This way, we can move the wildcard-management code, present in CachingLocaleUrlDefinitionDao, into a base class, thus providing wildcards for UrlDefinitionsFactory too (that is the default).
[TILES-275] Allow the use of multiple containers
undefined
Currently only one Tiles container can be used.
It should be possible to use more than one container under the same application, to be used, for example, in portal applications.
[TILES-270] Update to Servlet 2.5 and JSP 2.1
undefined

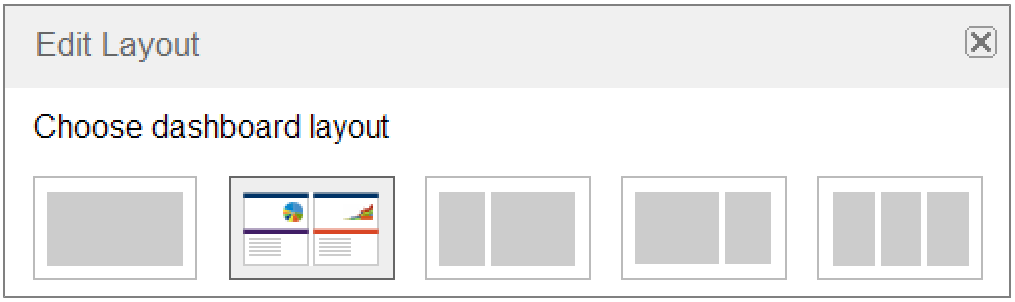
Adding and Removing Columns on your Dashboard

You can change the number of columns in your dashboard. If you reduce the number of columns, the gadgets already on the dashboard will move to fit into the new layout. If you increase the number of columns, the new columns will be empty until you move gadgets into them.

To change the number of columns in your dashboard,

1. Go to your application's dashboard. In most cases, you will do this by clicking the '**Dashboard**' link or the '**Home**' link at the top left of the application screen.
2. Click '**Edit Layout**'. You can see this link at top right of the screenshot above.
3. The 'Edit Layout' popup window appears, as shown in the screenshot below. The popup window displays diagrams of the available dashboard layouts, representing the number of columns in each layout and the relative width of each column. Click one of the diagrams to select the layout you want.
4. Now you can drag and drop gadgets into the columns as described above.

Screenshot: Choosing a dashboard layout



Resizing Columns on your Dashboard

The dashboard's 'Edit Layout' feature allows you limited control over the width of the columns on your dashboard. To choose a wider left-hand or right-hand column, follow the steps described above for adding and removing columns and choose the layout that you want, based on the diagram in the 'Edit Layout' popup window, as shown in the screenshot above.

Notes about Different Dashboards

Differing Look and Functionality

The screenshots on this page show the basic Atlassian dashboard layout. The layout will be similar in all applications that support the new Atlassian Gadgets framework. However, there may be slight differences in the look and feel for a particular application and some applications may have additional functionality. For example, JIRA will allow you to create multiple dashboards on separate tabs. The extra functionality is described in the documentation for the application concerned.

Permission to Update a Dashboard

Your ability to update a dashboard depends on the application hosting the dashboard. Most applications will require you to log in before you can update the dashboard. Your ability to update the dashboard may then depend on the permissions assigned to your username.

Who Will See the Changes You have Made to the Dashboard?

This depends on the application hosting the dashboard:

- In most cases, you will need to log in to the application before you can update the dashboard, and your updates will apply only to your personal dashboard.
- Some applications allow dashboards that are shared by groups of people. If you have permission to update a shared dashboard, the other people sharing the dashboard will see your changes too.
- If you have permission to update an application's default dashboard, your changes to the default dashboard will be seen by anyone who views that dashboard.

RELATED TOPICS

Gadgets and Dashboards User Guide

- Allowing Other Applications to Access Data on Your Behalf
- Adding a Gadget to your Dashboard
- Removing a Gadget from your Dashboard
- Changing the Layout of your Dashboard
- Changing the Look and Behaviour of a Gadget
- Adding an Atlassian Gadget to iGoogle and Other Web Sites

Changing the Look and Behaviour of a Gadget

This page tells you how to change the way a gadget looks and behaves on your dashboard, based on the configurable properties supplied by the gadget.

On this page:

- Permission to Update a Dashboard and its Gadgets
- Visibility of Your Changes

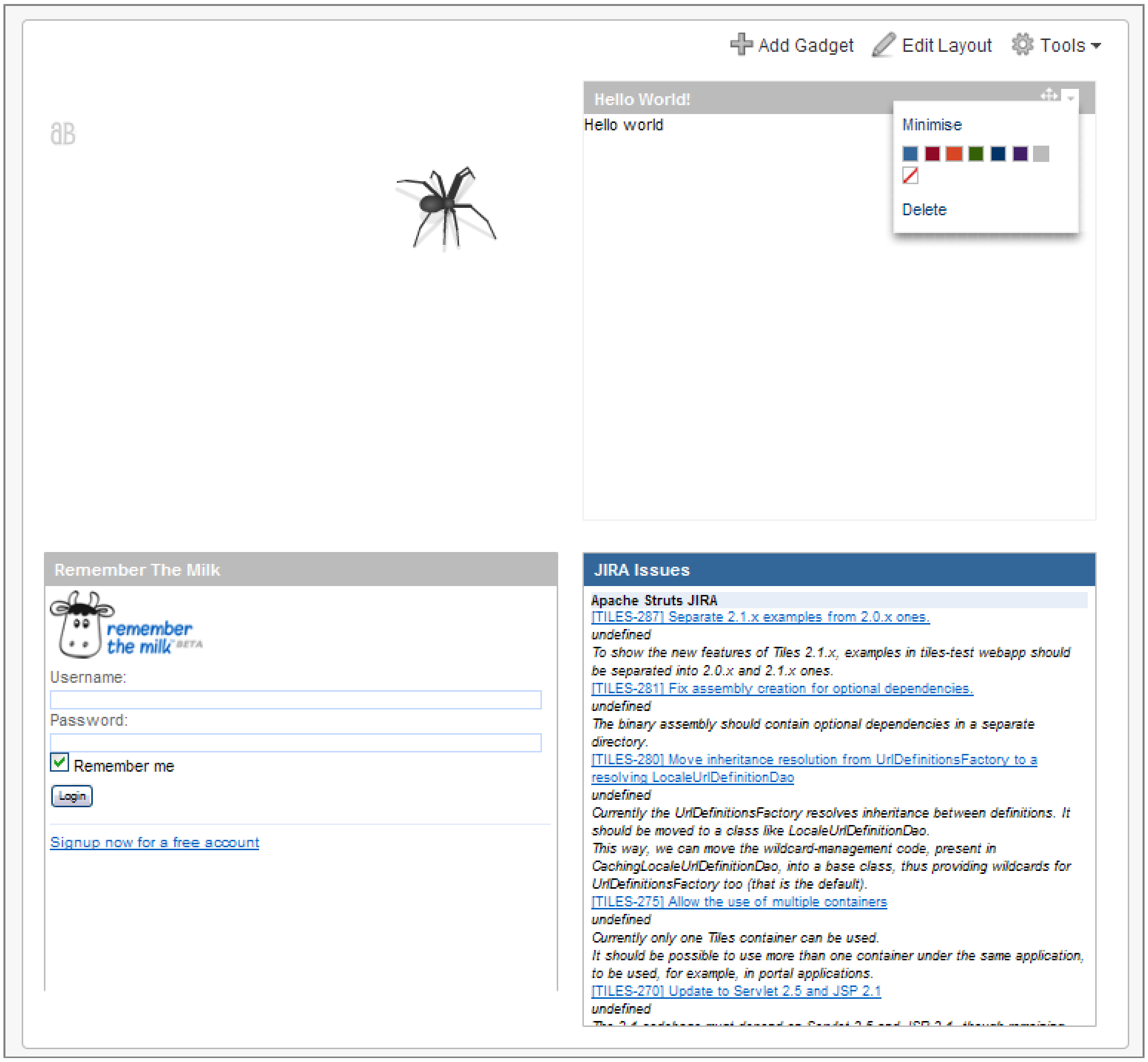
Hiding or Changing the Colour of the Gadget's Frame

You can change the colour of the frame surrounding a gadget on your dashboard. You can even hide the gadget's frame altogether, so that it only shows when you move your mouse pointer over the gadget. In the screenshot below, the top two gadgets have hidden frames. The frame for the top gadget on the left is not visible. The frame for the top gadget on the right is currently visible because the mouse pointer is hovering over the gadget.

To hide or change the colour of a gadget's frame,

1. Go to the dashboard by clicking the '**Dashboard**' link or the '**Home**' link at the top left of the screen.
2. The dashboard will appear, looking something like the screenshot below. Move your mouse pointer over the gadget you want to change. If the gadget's frame is hidden, the frame will appear now.
3. Click the dropdown menu icon ▾ at top right of the gadget frame.
4. The dropdown menu will appear, as shown in the screenshot below. Click the colour you want for your gadget's frame. To hide the gadget's frame, select the white colour box with the red line through it. ▨

Screenshot: Hiding or changing the colour of a gadget's frame



Hello world

Username:

Password:

☑ Remember me

Signup now for a free account

Apache Struts JIRA

[TILES-287] Separate 2.1.x examples from 2.0.x ones.
undefined
To show the new features of Tiles 2.1.x, examples in tiles-test webapp should be separated into 2.0.x and 2.1.x ones.

[TILES-281] Fix assembly creation for optional dependencies.
undefined
The binary assembly should contain optional dependencies in a separate directory.

[TILES-280] Move inheritance resolution from UrlDefinitionsFactory to a resolving LocaleUrlDefinitionDao
undefined
Currently the UrlDefinitionsFactory resolves inheritance between definitions. It should be moved to a class like LocaleUrlDefinitionDao.
This way, we can move the wildcard-management code, present in CachingLocaleUrlDefinitionDao, into a base class, thus providing wildcards for UrlDefinitionsFactory too (that is the default).

[TILES-275] Allow the use of multiple containers
undefined
Currently only one Tiles container can be used.
It should be possible to use more than one container under the same application, to be used, for example, in portal applications.

[TILES-270] Update to Servlet 2.5 and JSP 2.1
undefined

Minimising and Expanding a Gadget

You can shrink (minimise) a gadget on your dashboard so that it displays only the top bar of the gadget frame. In the screenshot below, the top left gadget ('**Spider**') has been minimised.

- If you minimise a gadget that has a hidden frame, the gadget will not be visible on the dashboard until you move your mouse pointer over the gadget. See the section above on hiding or changing the colour of the gadget frame.
- You can minimise/expand a gadget even if you do not have update permissions on the dashboard.
- The minimise/expand setting is stored in a cookie, and is not saved to the dashboard server.

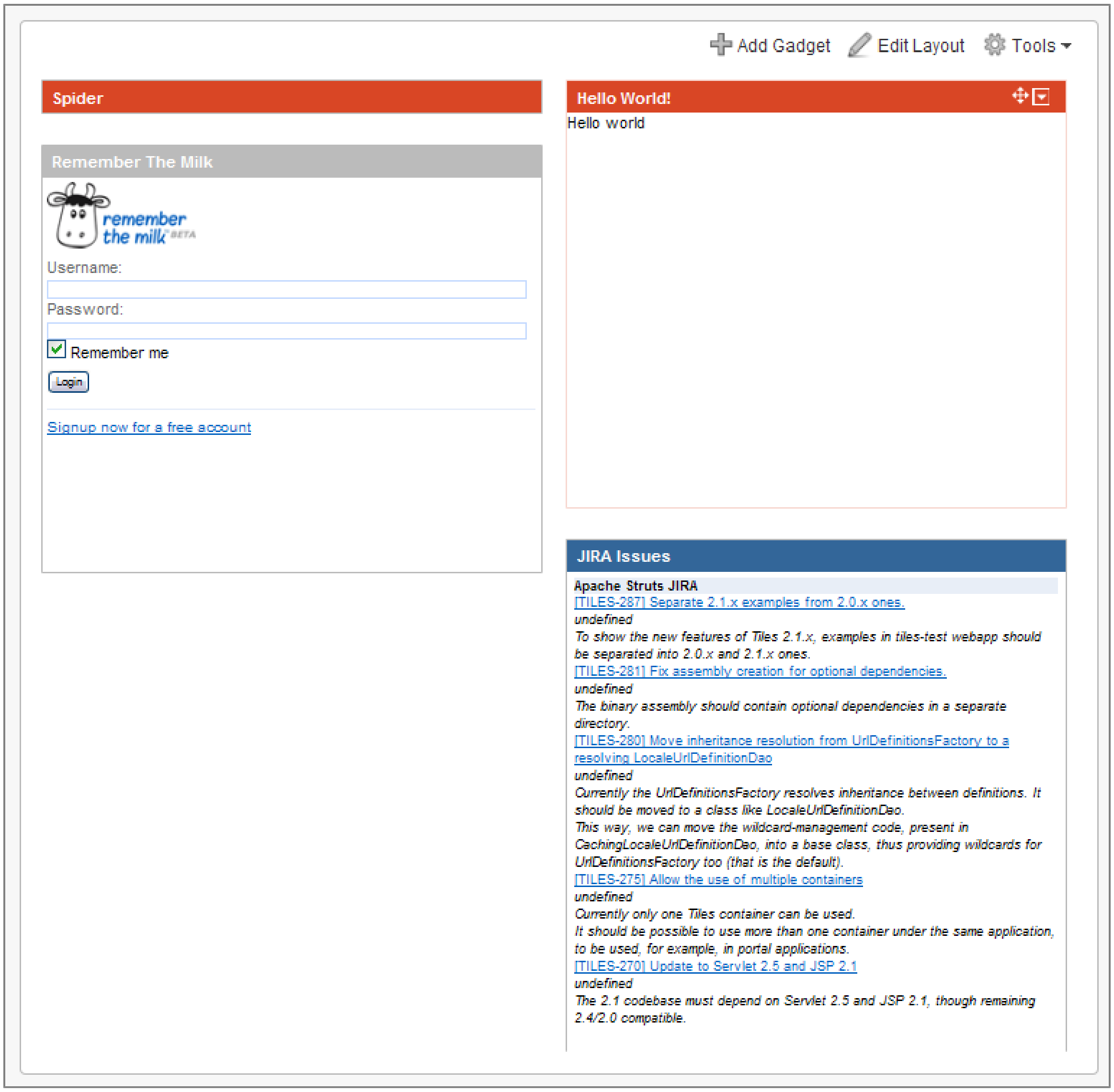
To minimise a gadget,

1. Move your mouse pointer over the gadget you want to change.
2. The gadget menu icons will appear. Click the dropdown menu icon ▼ at top right of the gadget frame.
3. The dropdown menu will appear, as shown in the screenshot above. Click '**Minimise**'.

To expand a gadget that has been minimised,

1. Move your mouse pointer over the gadget you want to change.
2. The gadget menu icons will appear. Click the dropdown menu icon ▼ at top right of the gadget frame.
3. The dropdown menu will appear. Click '**Expand**'.

Screenshot: A minimised gadget



Hello world

Apache Struts JIRA

[TILES-287] Separate 2.1.x examples from 2.0.x ones.

undefined

To show the new features of Tiles 2.1.x, examples in tiles-test webapp should be separated into 2.0.x and 2.1.x ones.

[TILES-281] Fix assembly creation for optional dependencies.

undefined

The binary assembly should contain optional dependencies in a separate directory.

[TILES-280] Move inheritance resolution from UrlDefinitionsFactory to a resolving LocaleUrlDefinitionDao

undefined

Currently the UrlDefinitionsFactory resolves inheritance between definitions. It should be moved to a class like LocaleUrlDefinitionDao.
This way, we can move the wildcard-management code, present in CachingLocaleUrlDefinitionDao, into a base class, thus providing wildcards for UrlDefinitionsFactory too (that is the default).

[TILES-275] Allow the use of multiple containers

undefined

Currently only one Tiles container can be used.
It should be possible to use more than one container under the same application, to be used, for example, in portal applications.

[TILES-270] Update to Servlet 2.5 and JSP 2.1

undefined

The 2.1 codebase must depend on Servlet 2.5 and JSP 2.1, though remaining 2.4/2.0 compatible.

Opening the Maximised or Canvas View of a Gadget

Some gadgets allow you to expand themselves so that they take up the entire space allowed by the dashboard. This is also known as 'canvas view'.

- The maximised or canvas view of a gadget often provides additional functionality, i.e. more than is available in the standard view of the gadget.
- This is not the same as minimising and then expanding a gadget (see above).
- Only some gadgets provide the maximised or canvas view.
- You can open the canvas view of a gadget even if you do not have update permissions on the dashboard.
- The maximised/canvas view setting is stored in a cookie, and is not saved to the dashboard server.

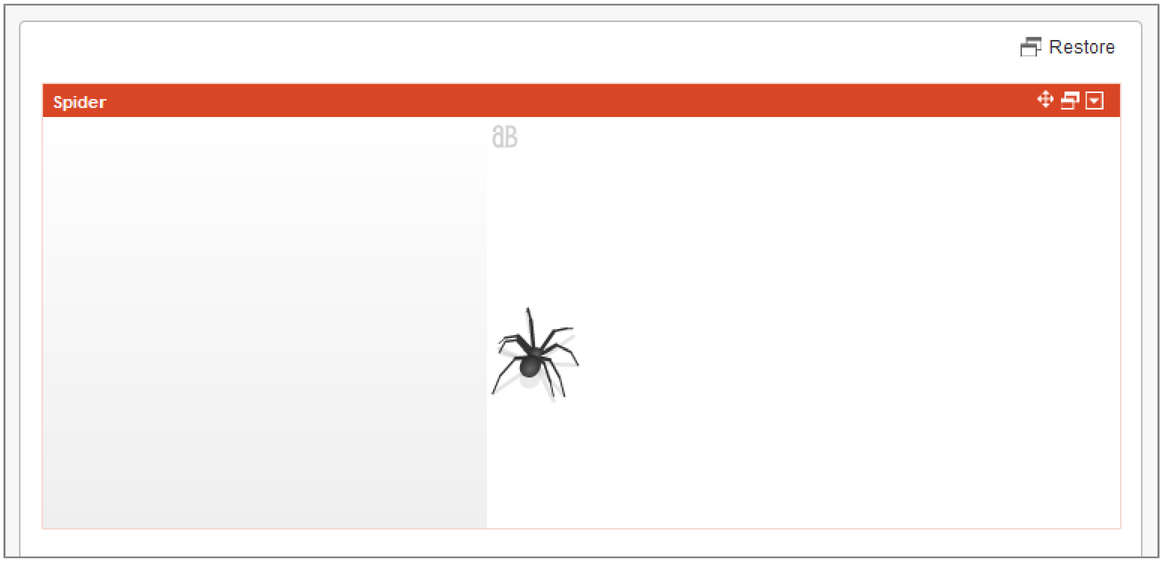
To open the maximised or canvas view of a gadget,

1. Move your mouse pointer over the gadget you want to change.
2. The gadget menu icons will appear. Click the maximise icon 🟥 at top right of the gadget frame. This icon will appear only if the gadget provides a maximised or canvas view.
3. The gadget's maximised view will open, as shown in the screenshot below.

To close the canvas view and return to your dashboard,

1. Click the '**Restore**' option at the top right of the screen, or the '**Restore**' icon at top right of the gadget frame.

Screenshot: The maximised or canvas view of a gadget



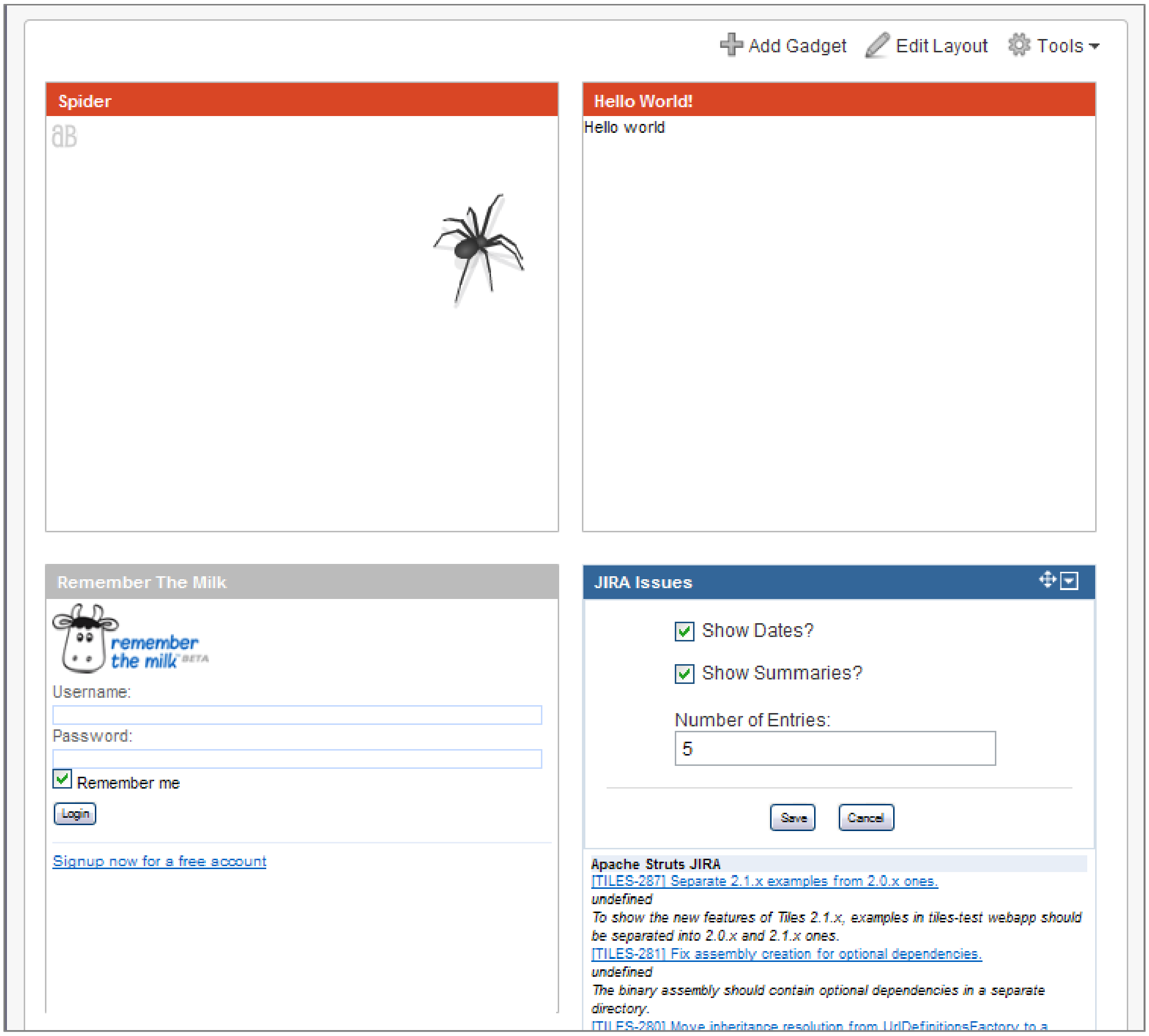
Editing a Gadget's Settings

Some gadgets provide specific properties or settings that you can edit. These settings will be different for each gadget. For example, a gadget may allow you to customise its welcome message, or to define the server where the gadget will find its information.

To edit a gadget's settings,

1. Move your mouse pointer over the gadget you want to change.
2. The gadget menu icons will appear. Click the dropdown menu icon ▼ at top right of the gadget frame.
3. The dropdown menu will appear. Click '**Edit**'.
4. A panel will open, showing the settings offered by the selected gadget. In the screenshot below, the bottom two gadgets have their settings panels open.
5. Adjust the settings as required then click '**Save**'.

Screenshot: Editing a gadget's settings



Hello world

Username:

Password:

☑ Remember me

☑ Show Dates?

☑ Show Summaries?

Number of Entries: 5

Apache Struts JIRA

[TILES-287] Separate 2.1.x examples from 2.0.x ones.

undefined

To show the new features of Tiles 2.1.x, examples in tiles-test webapp should be separated into 2.0.x and 2.1.x ones.

[TILES-281] Fix assembly creation for optional dependencies.

undefined

The binary assembly should contain optional dependencies in a separate directory.

[TILES-280] Move inheritance resolution from UrlDefinitionsFactory to a

Notes about Different Dashboards

Differing Look and Functionality

The screenshots on this page show the basic Atlassian dashboard layout. The layout will be similar in all applications that support the new Atlassian Gadgets framework. However, there may be slight differences in the look and feel for a particular application and some applications may have additional functionality. For example, JIRA will allow you to create multiple dashboards on separate tabs. The extra functionality is described in the documentation for the application concerned.

Permission to Update a Dashboard and its Gadgets

Your ability to update the gadgets on a dashboard depends on the application hosting the dashboard. Most applications will require you to log in before you can update the dashboard. Your ability to update the dashboard may then depend on the permissions assigned to your username.

Visibility of Your Changes

Who will see the changes you have made to the dashboard? This depends on the application hosting the dashboard:

- In most cases, you will need to log in to the application before you can update the dashboard, and your updates will apply only to your personal dashboard.
- Some applications allow dashboards that are shared by groups of people. If you have permission to update a shared dashboard, the other people sharing the dashboard will see your changes too.
- If you have permission to update an application's default dashboard, your changes to the default dashboard will be seen by anyone who views that dashboard.

When you minimise/expand a gadget or open the canvas view, the dashboard will 'remember' the change by saving the setting in a cookie. The change is **not** saved to the dashboard server. This has a few implications:

- The minimise/expand/canvas setting does not affect other users, regardless of your permission level.
- You can minimise/expand gadgets or open the canvas view on any dashboard you can view, regardless of whether you have permission to update the dashboard.
- If you switch browsers or computers, each browser and computer will store independent minimise/expand/canvas states.

RELATED TOPICS

Gadgets and Dashboards User Guide

- Allowing Other Applications to Access Data on Your Behalf
- Adding a Gadget to your Dashboard
- Removing a Gadget from your Dashboard
- Changing the Layout of your Dashboard
- Changing the Look and Behaviour of a Gadget
- Adding an Atlassian Gadget to iGoogle and Other Web Sites

Adding an Atlassian Gadget to iGoogle and Other Web Sites

This page tells you how to add an Atlassian gadget to external websites.

On this page:

Introduction

Gadgets that display information from Atlassian applications, such as JIRA, should be able to run on iGoogle, Gmail and other web sites that provide OpenSocial containers. Below are specific instructions for iGoogle and Gmail. You can experiment by adding a gadget to other sites, such as a Ning community like The Content Wrangler.

> ⚠ **External integration is experimental**
> The external integrations described below should work, but they are experimental at this stage. Have fun playing around with them. Atlassian does not support the use of Atlassian gadgets on iGoogle or other external web sites. See the section on limitations below.

Overview of Adding a Gadget to a Website

The exact procedure for adding a gadget depends on the website where you want to add the gadget. The basic steps are the same for all websites:

1. Find the Atlassian gadget's URL, i.e. the URL for the gadget's XML specification file.
2. Follow the procedure provided by the website where you want to add the gadget.

Finding an Atlassian Gadget's URL

First you need to find the URL for the gadget's XML specification file. A gadget's URL looks something like this:

```
http://example.com/my-gadget-location/my-gadget.xml
```

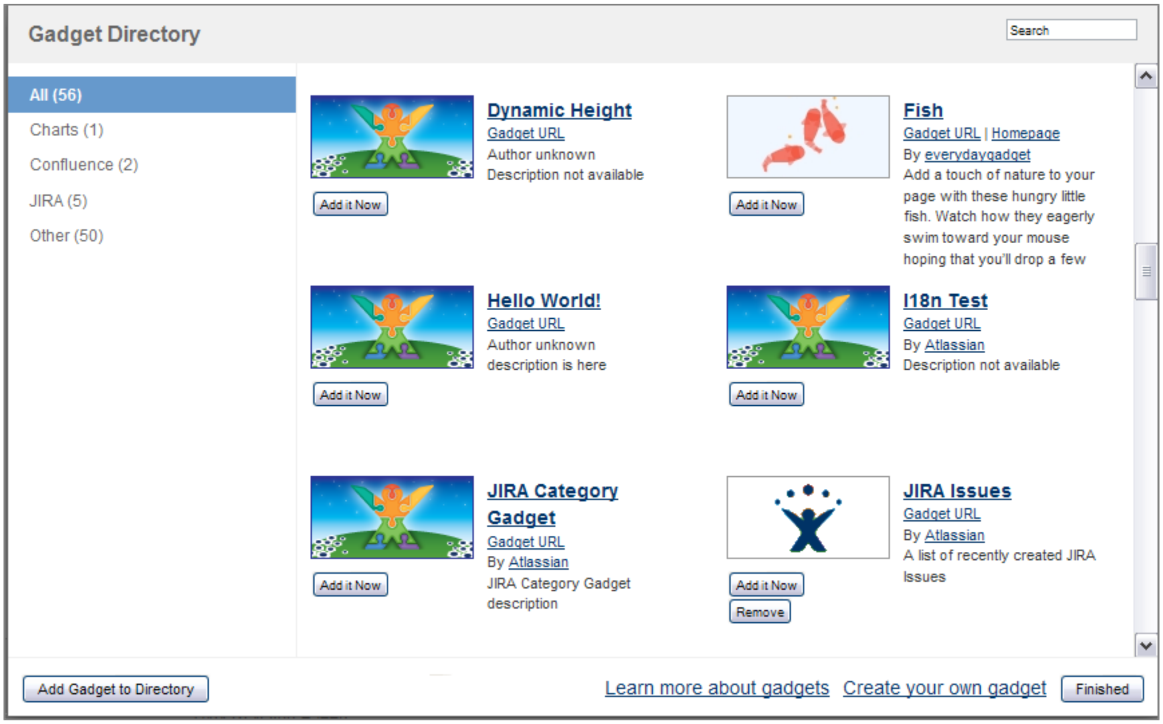
There are two options for finding the gadget's URL:

- The gadget URLs are shown on the '**Add Gadget**' screen, as described below.
- Alternatively, if your gadget is packaged as an Atlassian plugin, the format of the gadget URL is described here.

To find an Atlassian gadget's URL,

1. Go to the dashboard by clicking the '**Dashboard**' link or the '**Home**' link at the top left of the screen.
2. The dashboard will appear. Click '**Add Gadget**'.
3. The '**Add Gadget**' screen will display a list of available gadgets in your gadget directory, as shown in the screenshot below. Find the gadget you want, using one or more of the following tools:
   - Use the scroll bar on the right to move up and down the list of gadgets.
   - Select a category in the left-hand panel to display only gadgets in that category.
   - Start typing a key word for your gadget in the '**Search**' textbox. The list of gadgets will change as you type, showing only gadgets that match your search term.
4. When you have found the gadget you want, right-click the '**Gadget URL**' link for that gadget and copy the gadget's URL into your clipboard.

Screenshot: Finding a gadget's URL


Gadget Directory

Dynamic Height
Gadget URL
Author unknown
Description not available

Fish
Gadget URL | Homepage
By everydaygadget
Add a touch of nature to your page with these hungry little fish. Watch how they eagerly swim toward your mouse hoping that you'll drop a few

Hello World!
Gadget URL
Author unknown
description is here

I18n Test
Gadget URL
By Atlassian
Description not available

JIRA Category Gadget
Gadget URL
By Atlassian
JIRA Category Gadget description

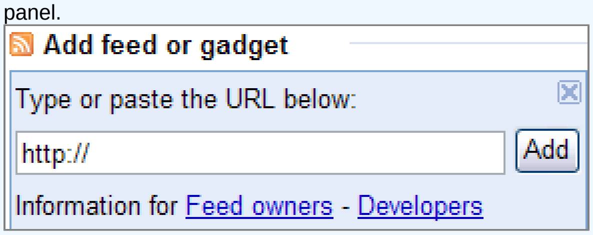
JIRA Issues
Gadget URL
By Atlassian
A list of recently created JIRA Issues

Adding an Atlassian Gadget to iGoogle

You can customise your iGoogle home page by adding gadgets and moving them around on the page.

To add an Atlassian gadget to your iGoogle page,

1. First find the gadget's URL as described above.
2. Go to iGoogle and log in if you have a username and password.
3. Click '**Add stuff**' near the top right of the iGoogle page.
4. The Google gadget directory will appear, showing a list of available gadgets. Click '**Add feed or gadget**' in the right-hand panel.


Add feed or gadget

Type or paste the URL below:

http://

Information for Feed owners - Developers

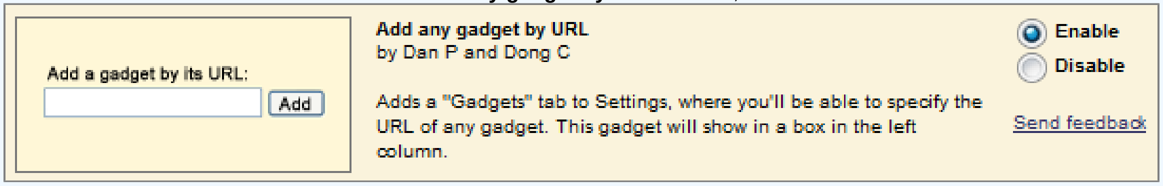
5. A text box will open, as shown above. Enter or paste the gadget's URL from your clipboard into the textbox and click '**Add**'.
6. Go back to your iGoogle home page. The gadget will appear on your iGoogle page.

Adding an Atlassian Gadget to Gmail

You can add gadgets to the left-hand panel of your Gmail page.

To add an Atlassian gadget to your Gmail page,

1. First find the gadget's URL as described above.
2. Log in to Gmail.
3. Click '**Settings**' near the top right of the Gmail page.
4. The Gmail settings page will appear. Click the '**Labs**' tab.
5. The Gmail Labs page will appear. This is a laboratory area or testing ground where Google allows you to use experimental features in Gmail. Scroll down to find the feature called '**Add any gadget by URL**'.
6. Select the '**Enable**' radio button next to the '**Add any gadget by URL**' feature, as shown here:



Add a gadget by its URL: [ ] Add

Add any gadget by URL
by Dan P and Dong C

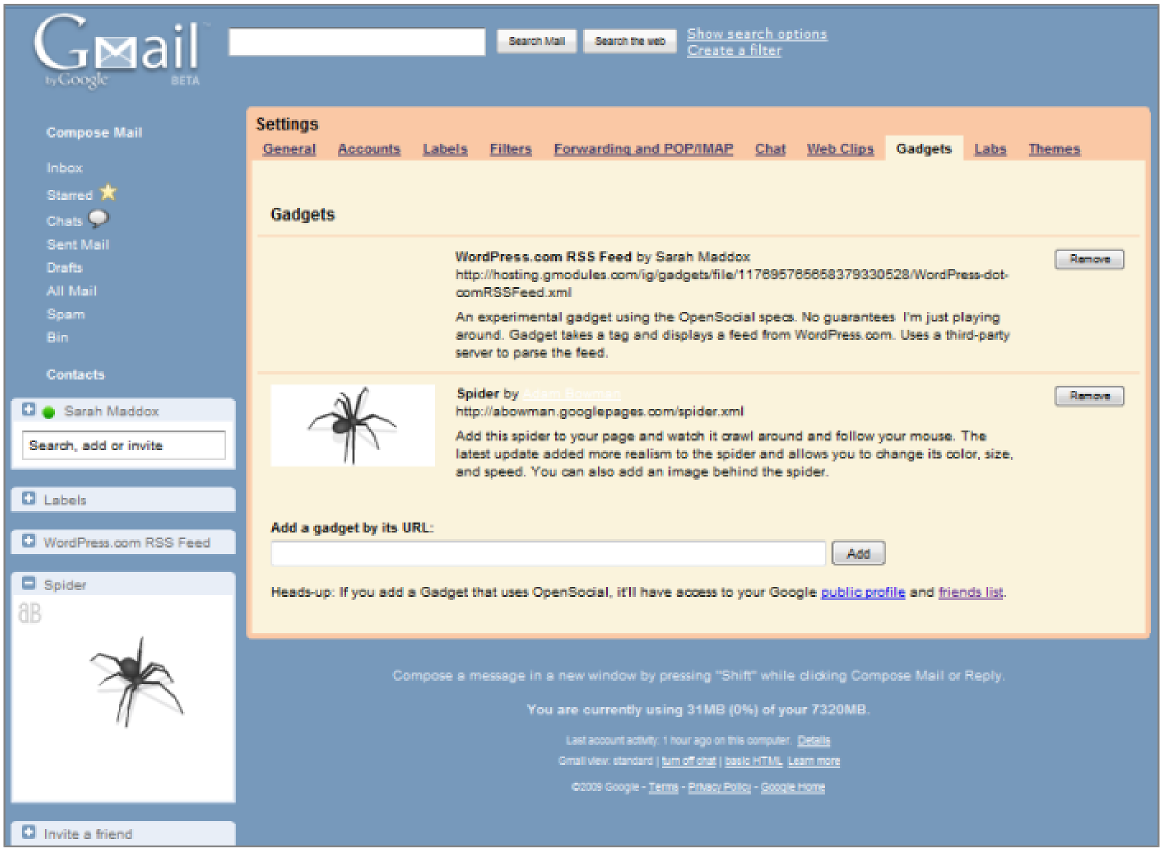
Adds a "Gadgets" tab to Settings, where you'll be able to specify the URL of any gadget. This gadget will show in a box in the left column.

● Enable
○ Disable

Send feedback

7. Click '**Save Changes**' to enable the new feature.
8. A new '**Gadgets**' tab will appear on your 'Settings' page. Click the 'Gadgets' tab.
9. The 'Gadgets' page will appear, as shown in the screenshot below. Enter or paste your gadget's URL into the '**Add a gadget by its URL**' textbox then click the '**Add**' button.
10. The gadget will appear in the left-hand panel of your Gmail page, as shown in the screenshot below.

Screenshot: Adding a gadget to Gmail



Gadgets

WordPress.com RSS Feed by Sarah Maddox
http://hosting.gmodules.com/ig/gadgets/file/117695765858379330528/WordPress-dot-comRSSFeed.xml
An experimental gadget using the OpenSocial specs. No guarantees I'm just playing around. Gadget takes a tag and displays a feed from WordPress.com. Uses a third-party server to parse the feed.

Spider by Adam Bowman
http://abowman.googlepages.com/spider.xml
Add this spider to your page and watch it crawl around and follow your mouse. The latest update added more realism to the spider and allows you to change its color, size, and speed. You can also add an image behind the spider.

Add a gadget by its URL:

Heads-up: If you add a Gadget that uses OpenSocial, it'll have access to your Google public profile and friends list.

Compose a message in a new window by pressing "Shift" while clicking Compose Mail or Reply.

You are currently using 31MB (0%) of your 7320MB.

Last account activity: 1 hour ago on this computer. Details
Gmail view: standard | turn off chat | basic HTML Learn more
©2009 Google - Terms - Privacy Policy - Google Home

Limitations and Support

The external integrations described above should work, but they are experimental at this stage. Have fun playing around with them.

⚠ Atlassian does not support the use of Atlassian gadgets on iGoogle or other external web sites.

Please note the following limitations:

1. Gadgets used on third-party websites must draw their content from a server that is accessible on the public internet. If you run your Atlassian applications behind a firewall, the third-party website will not be able to access the data and display it in the gadget. For example, if you put a JIRA gadget onto Gmail and your JIRA server is behind a firewall, the Google servers will not be able to retrieve the gadget specification and the JIRA gadget will display an error message on Gmail.
2. We have had difficulty adding gadgets from servers that use SSL, because Google's servers do not appear to accept our SSL certificate. These problems can be difficult to debug without help from Google.
3. Individual gadgets may or may not work well in Google's containers.
4. The production version of iGoogle does not support all of the newer gadget and OpenSocial features. The newest features may require signing up for the iGoogle Developer Sandbox. See Hints for Developing in iGoogle Sandbox.

RELATED TOPICS

Gadgets and Dashboards User Guide

- Allowing Other Applications to Access Data on Your Behalf
- Adding a Gadget to your Dashboard
- Removing a Gadget from your Dashboard
- Changing the Layout of your Dashboard
- Changing the Look and Behaviour of a Gadget
- Adding an Atlassian Gadget to iGoogle and Other Web Sites

Gadgets and Dashboards Development Hub

About the Gadgets and Dashboards Development Hub

The Development Hub is for people who want to write gadgets, create plugins, use the remote APIs or find out more about developing for the Atlassian products.

Resources and Links

Other resources for developers:

- Atlassian Developer Network
- Plugin Framework
- Javadoc

News and forums:

- Atlassian Forums
- Atlassian Developer blog

Table of Contents

RELATED TOPICS

Gadgets and Dashboards Documentation
Atlassian Product Integration

Gadget Development

At heart, Atlassian gadgets are Google gadgets. Atlassian gadgets use the new Google `gadgets.*` API defined by the OpenSocial specification. In addition, you will probably want to use some of the Atlassian extensions to the gadget specification. You will also want to package your gadget as an Atlassian plugin, along with plugin modules that allow your gadget to interact with the Atlassian applications such as JIRA and Confluence.

Gadget Development Table of Contents

RELATED TOPICS

Reference Documents
Gadgets and Dashboards Documentation

Getting Started with Gadget Development

At heart, Atlassian gadgets are Google gadgets. Atlassian gadgets use the new Google `gadgets.*` API defined by the OpenSocial specification. In addition, you will probably want to use some of the Atlassian extensions to the gadget specification. You will also want to package your gadget as an Atlassian plugin, along with plugin modules that allow your gadget to interact with the Atlassian applications such as JIRA and Confluence.

Introducing Gadgets, Containers, Plugins and Applications

A gadget is essentially an XML file containing the gadget specification. You will use recognised XML elements to define the following:

1. Gadget characteristics, such as the author's name (your name), the gadget title and description, preferred sizing, etc.
2. A screenshot and/or thumbnail image that containers can display to show users what your gadget looks like.
3. Required features that containers must provide for your gadget.
4. User preferences, where your gadget can allow its users to customise certain aspects of the gadget display.
5. The content section, where you use define the content that your gadget will display. This is where you add the HTML and JavaScript functions that produce your gadget's output.

As the gadget developer, you create the static XML file and make it available on a server. The dashboard or other container will pull the XML file from the server or plugin where it resides, and render it. The container may, for example, display the gadget inside an iframe on a dashboard. Or the container may display the gadget on a web page via a wiki macro.

Here is an overview of how gadgets, containers, plugins and applications interact with each other:

- Google gadgets that support the OpenSocial specification will run on OpenSocial containers.
- A simple Google gadget will run on an Atlassian container, including the dashboard of any Atlassian application that supports gadgets.
- In principle, an Atlassian gadget will run on an OpenSocial container.
- You can customise your Atlassian gadget to interface in funky ways with your Atlassian applications, via the Atlassian Plugin Framework. See Packaging your Gadget as an Atlassian Plugin.

Writing a Simple Gadget

If you have never written a gadget before, you can start by writing a simple Google gadget:

1. Become familiar with Google gadgets. There are two Google gadget developer guides, one for the earlier 'legacy' version of the Google gadget API and one for the new version which supports OpenSocial. Make sure you use the new guide.
2. Take a look at our hints for developing in iGoogle sandbox, compiled from our own experiences with Google gadgets.
3. Read the guide to getting started with OpenSocial.

Writing an Atlassian Gadget

Now move on to write an Atlassian Gadget.

RELATED TOPICS

Gadgets and Dashboards Development Hub

Writing an Atlassian Gadget

To write an Atlassian gadget, you can use the Atlassian extensions to the gadget specification:

1. Take a look at our example gadgets.
2. See the Atlassian gadget XML specification
3. Write your gadget, referring to the advanced topics below.
4. Use the Atlassian gadget reference implementation or deploy your gadget on another container.

More Advanced Topics

- Creating your Gadget XML Specification
- Using Substitution Variables and Directives in your Gadget
- Allowing Configuration Options in your Gadget
- Including Features into your Gadget

- Packaging your Gadget as an Atlassian Plugin
- Internationalising your Gadget
- Using Web Resources in your Gadget
- Using Atlassian REST APIs in your Gadget
- Providing User Authentication for Gadgets
- Using the Atlassian Gadgets JavaScript Framework
- Running your Gadget in the Atlassian Reference Implementation
- Managing Caching for your Gadget

RELATED TOPICS

Gadgets and Dashboards Development Hub

Creating your Gadget XML Specification

This is the reference guide for the XML file you will write to define your Atlassian gadget.

On this page:

Overview

A gadget is essentially an XML file containing the gadget specification. You will use recognised XML elements to define the following:

1. Gadget characteristics, such as the author's name (your name), the gadget title and description, preferred sizing, etc.
2. A screenshot and/or thumbnail image that containers can display to show users what your gadget looks like.
3. Required features that containers must provide for your gadget.
4. User preferences, where your gadget can allow its users to customise certain aspects of the gadget display.
5. The content section, where you use define the content that your gadget will display. This is where you add the HTML and JavaScript functions that produce your gadget's output.

Atlassian Gadgets and Google Gadgets

When writing an Atlassian gadget you may use additional XML elements, attributes and features that extend the Google gadgets XML specification.

On the page below, we give an overview of elements in the standard Google gadget specification that are supported by Atlassian gadgets. In cases where the Atlassian gadget specification differs from Google, we have added a comment like this:

> *(Comment about the difference in Atlassian gadgets.)*

Example of an Atlassian Gadget XML Specification

Below is a truncated example of a gadget specification. You can also take a look at a longer example of a real gadget.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Module>
   <ModulePrefs title="JIRA Issues" author_email="adent@example.com" directory_title="JIRA Issues"
         screenshot="
           http://labs.atlassian.com/svn/GADGETS/trunk/jira-issues/basic/jira-issues-screenshot.png"
         thumbnail="
           http://labs.atlassian.com/svn/GADGETS/trunk/jira-issues/basic/jira-issues-thumbnail.png">
      <Require feature="minimessage" />
      <Optional feature="dynamic-height" />
   </ModulePrefs>
   <UserPref name="show_date" display_name="Show Dates?" datatype="bool" default_value="true"/>

   <UserPref name="show_summ" display_name="Show Summaries?" datatype="bool" default_value="true"/>
   <UserPref name="num_entries" display_name="Number of Entries:" default_value="5"/>
   <Content type="html">
       <![CDATA[
          Hello, world!
       ]]>
    </Content>
</Module>
```

`Module` Element

This element indicates that the XML file contains a gadget. The element contains the entire gadget definition.

`ModulePrefs` Element

The `<ModulePrefs>` section specifies the gadget's characteristics, such as title, author, preferred sizing, and so on. Users who are viewing the gadget cannot change the values in this section.

You can use substitution variables, like `__UP_myuserpref__`, in the attributes in the `<ModulePrefs>` and `<UserPref>` sections, where `myuserpref` matches the `name` attribute of a user preference. See the Google documentation on user preference substitution variables.

Parent Element: `Module`

Example:

```
<ModulePrefs title="JIRA Issues" author_email="adent@example.com" directory_title="JIRA Issues"
      screenshot="
        http://labs.atlassian.com/svn/GADGETS/trunk/jira-issues/basic/jira-issues-screenshot.png"
      thumbnail="
        http://labs.atlassian.com/svn/GADGETS/trunk/jira-issues/basic/jira-issues-thumbnail.png">
   <Require feature="minimessage" />
   <Optional feature="dynamic-height" />
</ModulePrefs>
```

Attributes:

Attribute Name	Required /Optional	Description
title	Optional	The title of the gadget. This is a standard Google gadget attribute. See the Google Gadgets XML Reference. Refer also to the `directory_title` attribute below.
title_url	Optional	If you specify this value, the gadget title will link to this URL. This is a standard Google gadget attribute. See the Google Gadgets XML Reference.
description	Optional	A description of the gadget. This is a standard Google gadget attribute. See the Google Gadgets XML Reference.
author	Optional	Your name, as creator of the gadget. This is a standard Google gadget attribute. See the Google Gadgets XML Reference.
author_email	Optional	Your email address, as creator of the gadget. This is a standard Google gadget attribute. See the Google Gadgets XML Reference.

screenshot	Optional	The address of an image that a container or directory can use to illustrate the gadget. This is a standard Google gadget attribute. See the Google Gadgets XML Reference. *This attribute is not referenced by Atlassian Gadgets. Atlassian Gadgets does not make use of this feature, so it will have no effect on an Atlassian container. You may include the feature in your gadget if you want it to be used in other containers.*
thumbnail	Optional	The address of a thumbnail image (120x60 pixels) that a container or directory can use to illustrate the gadget. This is a standard Google gadget attribute. See the Google Gadgets XML Reference.
directory_title	Optional	The name of the gadget as it should appear in the gadget directory. If the value of your `title` attribute contains user preference substitution variables (like `__UP_userpref__`), then you will supply a plain text value in the `directory_title` attribute. See the Google documentation on user preference substitution variables.

`Require` Element

The `<Require>` element instructs the container that the gadget needs a specific feature to be provided by the container.

Parent Element: `Module/ModulePrefs`

Example:

```
<Require feature="minimessage" />
<Require feature="dynamic-height" />
```

Attributes:

Attribute Name	Required /Optional	Description
feature	Required	The name of the feature that the gadget requires. See our guide to including features into your gadget.

`Optional` Element

The `<Optional>` element instructs the container that the gadget can make use of a specific feature, if the container is able to provide it.

Parent Element: `Module/ModulePrefs`

Example:

```
<Optional feature="dynamic-height" />
```

Attributes:

Attribute Name	Required /Optional	Description
feature	Required	The name of the feature that the gadget will use if provided by the container. See our guide to including features into your gadget.

`Param` Element

The `<Param>` element allows you to supply parameters for features requested via the `<Require>` or `<Optional>` elements.

Parent Element: `Module/ModulePrefs/Optional` and `Module/ModulePrefs/Require`

Example:

```
<Optional feature="gadget-directory">
   <Param name="categories">
      JIRA
      Charts
   </Param>
</Optional>
```

Attributes:

Attribute Name	Required /Optional	Description
name	Required	A name-value pair giving the parameter and its value. You can read more about the `<Param>` element in the Google Gadgets XML Reference.

Preload Element

The `<Preload>` element instructs the container to fetch data from a remote source during the gadget rendering process.

Parent Element: `Module/ModulePrefs`

The Atlassian usage of this element is identical to the usage described in the Google Gadgets XML Reference.

Icon Element

> *This attribute is not referenced by Atlassian Gadgets. Atlassian Gadgets does not make use of this feature, so it will have no effect on an Atlassian container. You may include the feature in your gadget if you want it to be used in other containers.*

The `<Icon>` element specifies a 16x16 pixel image that containers can associate with a particular gadget.

Parent Element: `Module/ModulePrefs`

Locale Element

The `<Locale>` element specifies the locales supported by your gadget. For more information, please refer to the page on internationalising your gadget.

Parent Element: `Module/ModulePrefs`

Link Element

> *This attribute is not referenced by Atlassian Gadgets. Atlassian Gadgets does not make use of this feature, so it will have no effect on an Atlassian container. You may include the feature in your gadget if you want it to be used in other containers.*

The `<Link>` element specifies a container-specific link. For example, a container may link to your gadget to provide online help.

Parent Element: `Module/ModulePrefs`

OAuth Element and its Children

The `<OAuth>` element specifies your gadget's use of the OAuth protocol for authentication. For more information on this element and its child elements, please refer to the page on user authentication for gadgets.

Parent Element: `Module/ModulePrefs`

Child Elements: `Service`, `Request`, `Access` and `Authorization` — Please refer to the page on user authentication for gadgets.

UserPref Element

The `<UserPref>` section allows your gadget to give users a way of supplying their preferences and other user-specific information, called 'user preferences'. These user input fields are turned into user interface controls when the gadget runs. When the gadget is displayed on iGoogle, the user preferences are accessible from the 'Edit settings' dropdown menu. In Atlassian gadgets, the menu option is 'Edit'.

As a gadget developer, you are entirely free to choose the user preference name.

The container handles generation of the configuration UI, saves the settings and provides an API to access the settings in JavaScript.

For example, a gadget that displays a list of JIRA issues may allow the user to choose whether to display the issue summaries and to specify how many issues should be displayed.

You can use substitution variables, like `__UP_myuserpref__`, in the attributes in the `<ModulePrefs>` and `<UserPref>` sections, where `myuserpref` matches the `name` attribute of a user preference. See the Google documentation on user preference substitution variables.

See the discussion of gadget configuration options.

Parent Element: `Module`

Example:

```
<UserPref name="show_summ" display_name="Show Summaries?" datatype="bool" default_value="true"/>
<UserPref name="num_entries" display_name="Number of Entries:" default_value="5"/>
```

Attributes:

Attribute Name	Required /Optional	Description
name	Required	The name of the user preference. This is a standard Google gadget attribute. See the Google Gadgets XML Reference.
display_name	Optional	The text displayed next to the preference input field. This is a standard Google gadget attribute. See the Google Gadgets XML Reference.
urlparam	Optional	The string to pass as the parameter name for content type="url". This is a standard Google gadget attribute. See the Google Gadgets XML Reference.
datatype	Optional	The data type of this attribute. This is a standard Google gadget attribute. See the Google Gadgets XML Reference .
required	Optional	Boolean argument (true or false) indicating whether this user preference is required. The default is false. This is a standard Google gadget attribute. See the Google Gadgets XML Reference.
default_value	Optional	A default value for the user preference. This is a standard Google gadget attribute. See the Google Gadgets XML Reference.

`Enum` Element

The `<Enum>` element allows you to specify enumerated values for your user preference, when you have given a `datatype` of `enum`.

Parent Element: `Module/UserPref`

The Atlassian usage of this element is identical to the usage described in the Google Gadgets XML Reference.

`Content` Element

The `<Content>` section is where it all happens!

In this section, you will define the type of content, and either include the content itself or provide a URL to pull in the content from an external source. You will combine the attributes and user preferences with programming logic and formatting information to make the gadget do what you want it to do.

There are two main types of gadget content, distinguished by the `type` attribute of the `<Content>` element.

Type `url` gadgets include content from a remote site, but do not have full access to features of the container. This type of gadget is supported for compatibility with existing legacy iGoogle gadgets, but is considered deprecated and should generally be avoided when creating new gadgets.

Type `html` gadgets include the content directly inside the `<Content>` element. This includes all of the HTML, CSS, and JavaScript code that the container will render in the gadget's iframe. This content must be enclosed in a `CDATA` section to avoid having the gadget specification parser interpret the embedded HTML markup as XML. For a guide to the JavaScript functionality available, please refer to our JavaScript API reference guide and the Gadgets API Reference.

Parent Element: `Module`

Example:

```
<Content type="html">
   <![CDATA[
      <strong>Hello, world!</strong>
   ]]>
</Content>
```

Attributes:

Attribute Name	Required /Optional	Description
type	Optional	The type of content. The possible values are `html` and `url`. The default is `html`.
href	Required for `type="url"`, and not allowed for other content types.	String that provides a destination URL.
view	Optional	The container view in which this content should be displayed. Gadgets can define multiple content sections, each with a different view. The views supported by Atlassian containers are `default` for the standard view of a gadget on a page alongside other gadgets and content, and `canvas` for the maximised view of a single gadget on a page by itself. Other containers may support additional view types. If the same content section should be displayed in more than one view, you can specify multiple views, separated by commas.

RELATED TOPICS

Writing an Atlassian Gadget
Gadgets and Dashboards Development Hub

Example of Gadget XML Specification

Below is the code from the JIRA Introduction gadget.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Module>
   <ModulePrefs
         title="__MSG_gadget.introduction.title__"
         directory_title="__MSG_gadget.introduction.title__"
         description="__MSG_gadget.introduction.description__"
         author="Atlassian"
         author_affiliation="Atlassian" author_location="Sydney, Australia"
         title_url="http://www.atlassian.com/"
         screenshot="#staticResourceUrl("com.atlassian.jira.gadgets:average-age-chart-gadget",
           "intro-screenshot.png")"
         thumbnail="#staticResourceUrl("com.atlassian.jira.gadgets:average-age-chart-gadget",
           "intro-thumb.png")">
   <Optional feature="gadget-directory">
   <Param name="categories">JIRA</Param>
   </Optional>
   <Require feature="dynamic-height" />
   <Require feature="settitle" />
   #supportedLocales("gadget.common,gadget.introduction")
   </ModulePrefs>
   <Content type="html">
   <![CDATA[
            #requireResource("com.atlassian.jira.gadgets:common")
            #includeResources()
            <div id="intro-content"><div class="fullyCentered loading"></div></div>
            <script type="text/javascript">
                  jQuery.namespace("jira.app.intro");
                  jira.app.intro = function(){
                        var response = function(){
                              var resize = function(){gadgets.window.adjustHeight();};

                              return function(response){
                                    if (response.rc != 200){
                                          AJS.log("Error on server call
                                             '__ATLASSIAN_BASE_URL__/rest/gadget/1.0/intro'.  Return
                                             code: " + response.rc);
                                          var fragment =
                                             response.text.match(/<body[^>]*>([\S\s]*)<\/body[^>]*>/);
                                          if (fragment && fragment.length > 0) {
                                                jQuery("body").html("<div style=\"padding:0 20px\">" +
                                                   fragment[1] + "</div>");
                                                resize();
                                          }
                                          return;
                                    }
                                    if (response.data && response.data.html){
                                          jQuery("#intro-content").removeClass("loading").html(response.data
                                    }
                                    else {
                                          jQuery("#intro-content").removeClass("loading").html("__MSG_gadget
                                    }
                                    resize();
                              };
                        }();

                        var setTitle = function(){
                              var isLocal = function(){return
                                 window.location.href.indexOf("__ATLASSIAN_BASE_URL__/plugins/servlet/gadget
                                 == 0;};
                              var instanceNameResponse = function(response){
                                    if (response.data && response.data.instanceName){
                                          gadgets.window.setTitle(response.data.instanceName + ":
                                             __MSG_gadget.introduction.title__");
                                    }
                              };
                              return function(){
                                    if (!isLocal()){
                                          var params = {};
                                          params[gadgets.io.RequestParameters.CONTENT_TYPE] =
                                             gadgets.io.ContentType.JSON;
                                          params[gadgets.io.RequestParameters.METHOD] =
                                             gadgets.io.MethodType.GET;
                                          gadgets.io.makeRequest("__ATLASSIAN_BASE_URL__/rest/gadget/1.0/ins
                                             instanceNameResponse, params);
                                    }
                              };
                        }();
                        return function(){
                              setTitle();
                              var params = {};
                              params[gadgets.io.RequestParameters.CONTENT_TYPE] =
                                 gadgets.io.ContentType.JSON;
                              params[gadgets.io.RequestParameters.METHOD] = gadgets.io.MethodType.GET;
                              gadgets.io.makeRequest("__ATLASSIAN_BASE_URL__/rest/gadget/1.0/intro",
                                 response, params);

                        };
                  }();
                  gadgets.util.registerOnLoadHandler(jira.app.intro);
            </script>


   ]]>
   </Content>
</Module>
```

RELATED TOPICS

Creating your Gadget XML Specification

Using Substitution Variables and Directives in your Gadget

The Atlassian Gadgets framework allows you to specify substitution variables and #-directives in your gadget specification XML file that will be replaced by generated content at run time.

On this page:

Substitution Variables

The Atlassian Gadgets framework supports the standard substitution variables as defined in the Google documentation on user preference substitution variables. These are variables like `__UP_myuserpref__`, in the attributes in the `<ModulePrefs>` and `<UserPref>` sections, where `myuserpref` matches the `name` attribute of a user preference.

In addition, the following Atlassian-specific substitution variables are available.

Atlassian Base URL Variable

At run time, this variable will be replaced with the base URL of the host application (e.g. JIRA) that is serving the URL.

Note that this only works with plugin gadgets, since the translation is done by the gadget *publisher*, not the renderer. (The renderer does the standard user pref and msg substitutions.)

Format:

```
__ATLASSIAN_BASE_URL__
```

Note: There is a double underscore at the beginning and at the end of the variable.

Example 1:
Here is an HTML snippet from the `<Content>` section of your gadget XML specification using the `__ATLASSIAN_BASE_URL__` variable:

```
<img src="__ATLASSIAN_BASE_URL__/download/resources/com.atlassian.jira.gadgets/loading.gif" height=
   "35" width="35"/>
```

At run time, the above section of the gadget XML will be replaced with the following:

```
<img src="http://myhost.com:port/myapp/download/resources/com.atlassian.jira.gadgets/loading.gif"
   height="35" width="35"/>
```

Example 2:
Here is a script snippet from the `<Content>` section of your gadget XML specification:

```
gadgets.io.makeRequest("__ATLASSIAN_BASE_URL__/rest/gadget/1.0/intro", response, params);
```

At run time, the above section of the gadget XML will be replaced with the following:

```
gadgets.io.makeRequest("http://myhost.com:port/myapp/rest/gadget/1.0/intro", response, params);
```

Directives

Our software (JIRA, FishEye etc) is packaged software that can be installed on customers' servers. We want our gadget developers to be able to write gadgets that can retrieve information from any server, rather than from a specific web service or URL. So we have developed a templating layer that sits on top of the gadget specification.

The Atlassian Gadgets framework allows you to specify #-directives in your gadget specification XML file that will be replaced by generated content at run time. These #-directives are provided by the Atlassian Gadget Publisher plugin. They work for any gadget that is provided as a plugin in an Atlassian application. The #-directives do **not** work for gadgets that are served from an external web server.

#-directives are sometimes also called 'macros' or 'pseudo-macros'.

Specific uses of #-directives:

- Internationalising your Gadget
- Using Web Resources in your Gadget

RELATED TOPICS

Creating your Gadget XML Specification
Writing an Atlassian Gadget
Gadgets and Dashboards Development Hub

Allowing Configuration Options in your Gadget

This page gives an overview of the options available for allowing users to configure their instance of your gadget.

> **Security Note**
> Keep in mind that user preferences are stored in the container server, and are passed as URL parameters to the gadget rendering server. You may want to be careful about what you put in there. In particular, a shipping gadget would **never** want to store a username and password in user preferences, because then anyone with read access to a shared dashboard with that gadget would be able to get the owner's login information. See user authentication guidelines.

On this page:

- Setting User Preferences via the `UserPref` Element in the Gadget XML
- Handling Configuration Inside your Gadget

Setting User Preferences via the `UserPref` Element in the Gadget XML

You can make use of the user preferences section of your gadget XML specification, where you can declaratively specify a set of configurable options. See our guide to the gadget XML specification.

Each user preference has a name, a display name, a data type and a default value. You can also flag a user preference as required or optional.

To access the user preferences from your gadget, you can use one or more of the following:

- The user preferences JavaScript API, as described in the OpenSocial JavaScript API.
- Specially-formatted tokens in the gadget XML file that are automatically substituted by the gadget renderer. These tokens look like this: `__UP_pref_name__`.
- JavaScript that sets the user preferences via the `setprefs` feature. See our guide to including features into your gadget.

Advantages of this Configuration Method

- Development is easy. Just declare what you want in your gadget XML specification.
- The container handles generation of the configuration UI, saves the settings and provides an API to access the settings in JavaScript.
- You can make use of automatic substitution in the gadget specification content, including substitution in the gadget title and other metadata as well as the body content.
- It is easy for users to figure out how to change the settings, because the container handles the configuration consistently for all gadgets.

Disadvantages of this Configuration Method

- It is inflexible. You are limited to a small number of data types, have no control over the UI, and the only validation you can do is to set fields as required or optional.
- The user preferences are static. There is no way to change the number or type of configuration settings programatically. `Enum` type preferencess (rendered as a radio button or select list) can only have values that are hardcoded in the specification file. This means that they are not useful for something like choosing a project from JIRA.
- Both iGoogle and Atlassian Gadgets currently refresh the whole page when you save a user preference form.
- Both iGoogle and Atlassian Gadgets (via Shindig) automatically HTML-escape values returned from `prefs.getString()`. (See SHINDIG-89.) This means:
  - If you're using the value as literal HTML, you must not double-escape it.
  - If you're using the value in some other way (in a URL for example) then you must unescape it.

Handling Configuration Inside your Gadget

The other option is to handle configuration yourself, inside the gadget. This is the approach used by the Remember the Milk gadget, for example. In this case, you can store the settings in a user preference of type `hidden`, using the `setprefs` API as described in the Google developer guide , or you can come up with some other way to store the settings. You might be interested in looking at the json API for serialising data.

Advantages of this Configuration Method

- You can do whatever you want in your configuration UI: Load dynamic data, provide fancy UI controls, use Flash or Java, hide and show sections depending on what users enter. And so on.
- You can store the data however you want: Post it back to the application server, keep it in a cookie, use a Google Gears database.
- If you want to, you can still use user preferences to store the values and save yourself the effort of figuring out where to put the data.
- Because everything happens inside the gadget, you can avoid reloads and dynamically refresh the parts of your gadget that are affected when configuration settings change.

Disadvantages of this Configuration Method

- This method demands a lot more work. You have to write a lot of code to do things that the container handles automatically for `UserPref`. For example, if you want your gadget's title to be based on a user preference, you will have to use the settitle feature to do that yourself.
- There are no established conventions or guidelines for this yet.

- You will need to provide your own UI for entering the configuration mode, since the edit command provided by the container only works for `UserPref`.

RELATED TOPICS

Creating your Gadget XML Specification
Writing an Atlassian Gadget
Gadgets and Dashboards Development Hub

Including Features into your Gadget

A 'feature' is a JavaScript library that provides specific functionality for inclusion into your gadget. This page gives an overview of how you can use a feature in your gadget and a list of available features.

Using a Feature in your Gadget

To create a gadget that uses a particular feature, such as dynamic height adjustment, you will do the following:

1. Include the feature library via a `<Require>` element or an `<Optional>` element in your gadget XML specification. The format is:

```
<ModulePrefs
   title="My Gadget">
   <Require feature="my-feature-name"/>
</ModulePrefs>
```

   Here is a simple example:

```
<ModulePrefs
   title="My Height-Adjusting Gadget">
   <Require feature="dynamic-height"/>
</ModulePrefs>
```

2. Write some JavaScript code to use the feature. The JavaScript code goes in the `<Content>` element of your gadget XML specification. The JavaScript APIs are described in the OpenSocial JavaScript API and in our JavaScript API reference guide.

List of Features

An unadorned list of features is available from our code repository. Below we provide more detail about the features.

Supported in Atlassian Gadgets?	Feature Name	Description	Specified by OpenSocial or Atlassian Gadgets?

<table>
<tr>
<td>✓ Fully supported</td>
<td>atlassian.util</td>
<td>

Allows a gadget to retrieve the base URL for the renderer server. Example:

```
<Module>
   <ModulePrefs title="My Gadget">
      <Optional feature="atlassian.util"/>
   </ModulePrefs>
   <Content type="html">
   <![CDATA[
      <div id="main">
         <script language="javascript">
            function showRendererBaseUrl() {
               document.getElementById("main").innerHTML =
                     atlassian.util.getRendererBaseUrl();
            };
            gadgets.util.registerOnLoadHandler(showRendererBaseUrl);
         </script>
      </div>
   ]]>
   </Content>
</Module>
```

⚠ Because this feature is specific to Atlassian Gadgets, we recommend that you use `<Optional>` rather than `<Required>` when specifying this feature in your gadget. Otherwise the gadget will not work in other containers.

</td>
<td><b>Atlassian Gadgets</b></td>
</tr>
<tr>
<td>✗ Not supported</td>
<td>analytics</td>
<td>Google Analytics</td>
<td>OpenSocial</td>
</tr>
<tr>
<td>✗ Not supported</td>
<td>caja</td>
<td>Support for Caja, a Google project aiming to allow web applications to provide active content safely, simply, and flexibly. The basis of the project is that a subset of Javascript provides an object-capability language.</td>
<td>OpenSocial</td>
</tr>
<tr>
<td>✗ Not referenced</td>
<td>content-rewrite</td>
<td>

The content-rewrite feature defines a set of rewriting operations that a container can perform on rendered and proxied content. It also defines rules to allow developers to control which content the rewriter can operate on.
Tip: During development, you may find this feature useful to disable the automatic caching provided by Shindig, so that your gadget resources will be re-loaded when you re-load the page. Otherwise you will have to restart the container in order to see the updated values in your gadget. Format:

```
<ModulePrefs
   title="My Gadget">
   <Optional feature="content-rewrite">
        <Param name="expires">86400</Param>
        <Param name="include-url"></Param>
        <Param name="exclude-url">excluded</Param>
        <Param name="exclude-url">moreexcluded</Param>
        <Param name="minify-css">true</Param>
        <Param name="minify-js">true</Param>
        <Param name="minify-html">true</Param>
  </Optional>
</ModulePrefs>
```

See the OpenSocial Gadgets API Specification.

</td>
<td>OpenSocial</td>
</tr>
<tr>
<td>✓ Fully supported</td>
<td>core.io</td>
<td>

Provides remote content retrieval functions. See the OpenSocial `gadgets.io` API reference.

ℹ Note that you do not need to explicitly request this feature with a `<Required>` or `<Optional>` element. This feature is provided automatically to all gadgets.

</td>
<td>OpenSocial</td>
</tr>
</table>

✓ Fully supported	core	Provides core gadget support, including JavaScript APIs for manipulating JSON data, escaping and unescaping strings, and accessing user preferences. See the OpenSocial `gadgets.json` API reference, the OpenSocial `gadgets.Prefs` API reference, and the OpenSocial `gadgets.util` API reference. ℹ Note that you do not need to explicitly request this feature with a `<Required>` or `<Optional>` element. This feature is provided automatically to all gadgets.	OpenSocial

✓ Fully supported	dynamic-height	Gives a gadget the ability to resize itself. Format:
		`<ModulePrefs`
		`    title="My Gadget">`
		`    <Require feature="dynamic-height"/>`
		`</ModulePrefs>`
		You will call the JavaScript function `gadgets.window.adjustHeight()` whenever there is a change in content, or another event occurs that requires the gadget to resize itself. See the OpenSocial `gadgets.window` API reference. You may also find useful information in the Google documentation on creating a user interface.
			OpenSocial

✓ Fully supported	flash	Allows you to embed a Flash movie into your gadget. Format:
		`<ModulePrefs`
		`    title="My Gadget">`
		`    <Require feature="flash"/>`
		`</ModulePrefs>`
		You will call the JavaScript function `gadgets.flash.embedFlash()` to embed a `.swf` file in your gadget and display it in a designated location. Note that to use this feature, all resources must be bundled in a `.swf` file. See the OpenSocial `gadgets.flash` API reference. You may also find useful information in the Google documentation on creating a user interface.
			OpenSocial

✓ Fully supported	gadget-directory	Allows a gadget to specify details for its directory listing, such as the category in which the gadget should be listed. Format:
		`<ModulePrefs`
		`    title="My Gadget">`
		`    <Optional feature="gadget-directory">`
		`        <Param name="categories">`
		`            JIRA`
		`            Charts`
		`        </Param>`
		`    </Optional>`
		`</ModulePrefs>`
		The only recognized parameter is `categories` — the list of directory categories that this gadget should appear in, one per line. Valid values are "JIRA", "Confluence", "FishEye", "Crucible", "Crowd", "Clover", "Bamboo", "Admin", "Charts", "External Content", and "Other". Gadgets that don't specify a directory category, or that specify only invalid categories, will be listed in "Other".
		⚠ Because this feature is specific to Atlassian Gadgets, we recommend that you use `<Optional>` rather than `<Required>` when specifying this feature in your gadget. Otherwise the gadget will not work in other containers.
			Atlassian Gadgets

✗ Not supported	locked-domain		OpenSocial

✓ Fully supported	minimessage	A MiniMessage is a temporary message displayed to users from within a gadget. Format:
		`<ModulePrefs`
		`   title="My Gadget">`
		`   <Require feature="minimessage"/>`
		`</ModulePrefs>`
		See the OpenSocial `gadgets.MiniMessage` API reference. You may also find useful information in the Google documentation on creating a user interface.
			OpenSocial

✓ Fully supported	oauthpopup	Manages popup windows for OAuth. See our detailed instructions on OAuth and the OpenSocial `gadgets.oauth.Popup` API reference.	OpenSocial
✗ Not supported	opensocial-0.6		OpenSocial
✗ Not supported	opensocial-0.7		OpenSocial
✗ Not supported	opensocial-current		OpenSocial
✗ Not supported	opensocial-reference		OpenSocial
✗ Not supported	opensocial-templates		OpenSocial

✓ Fully supported	rpc	Provides operations for making remote procedure calls for gadget-to-container, container-to-gadget, and gadget-to-gadget communication. Format:
		`<ModulePrefs`
		`   title="My Gadget">`
		`   <Require feature="rpc"/>`
		`</ModulePrefs>`
		You need to specify "rpc" as a required feature if you are using the `gadgets.rpc` JavaScript API directly. If you are using only features that happen to rely upon `gadgets.rpc`, such as `settitle`, `dynamic-height`, then the `rpc` feature will be implicitly included and you do not need to include it explicitly. But you should not rely on this auto-inclusion if you are using the API directly, as this behaviour may differ in other containers. See the OpenSocial `gadgets.rpc` API reference.
			OpenSocial

✓ Fully supported	setprefs	Allows you to set the values for user preferences programmatically, without the user's direct participation. Format:
		`<ModulePrefs`
		`   title="My Gadget">`
		`   <Require feature="setprefs"/>`
		`</ModulePrefs>`
		See the OpenSocial `gadgets.Prefs` API reference. You may also find useful information in the Google documentation on saving state.
			OpenSocial

✓ Fully supported	settitle	Allows you to set your gadget's title programmatically. Format:
		`<ModulePrefs`
		`    title="My Gadget">`
		`    <Require feature="settitle"/>`
		`</ModulePrefs>`
		See the OpenSocial `gadgets.window` API reference. You may also find useful information in the Google documentation on creating a user interface.
			OpenSocial

☒ Not referenced	skins	Allows you to manage the skin of your gadget programmatically. Format:
		`<ModulePrefs`
		`    title="My Gadget">`
		`    <Require feature="skins"/>`
		`</ModulePrefs>`
		See the OpenSocial `gadgets.skins` API reference.
			OpenSocial

✓ Fully supported	tabs	Allows you to add a tabbed user interface to your gadget. Format:
		`<ModulePrefs`
		`    title="My Gadget">`
		`    <Require feature="tabs"/>`
		`</ModulePrefs>`
		See the OpenSocial `gadgets.TabSet` API reference. You may also find useful information in the Google documentation on creating a user interface.
			OpenSocial

💡 Partially supported	views	A view is a location in a container where a gadget is displayed. Different views have different characteristics. For example, a container might have a view that shows gadgets in a small format, and a view that shows gadgets in full page format. Atlassian Gadgets supports the following views:
		`default` — The standard view of a gadget, displayed in a small box on the page, possibly with other gadgets. You can also use the aliases '`DEFAULT`', '`DASHBOARD`', '`profile`', or '`home`'.
		`canvas` — The maximised view of a gadget when displayed by itself on the page.
		
		You need to specify "views" as a required feature if you are using the `gadgets.views` JavaScript API directly. If you just need to use the canvas view, it is enough to include a content section with `view="canvas"` and you do not need to specify the feature. Format:
		`<ModulePrefs`
		`    title="My Gadget">`
		`    <Require feature="views"/>`
		`</ModulePrefs>`
		See the OpenSocial `gadgets.views` API reference. You may also find useful information in the Google documentation on creating a user interface.
		Partial support
		Atlassian Gadgets provides only partial support of the "views" feature. Only the `getSupportedViews` function is known to work. The `requestNavigateTo` function is known *not* to work. The other functions have not yet been tested, so may or may not work.
		Restricting edit permissions on your gadget preferences
		There is an Atlassian-specific extension to the `views` feature that allows you to restrict who can edit your gadget preferences. See Restricting Edit Permissions on your Gadget Preferences.
			OpenSocial

❌ Not supported	pubsub	Allows your gadget to publish and subscrib to message channels. See the OpenSocial `gadgets.pubsub` API reference.	OpenSocial

In the above table we show the level of support provided in Atlassian Gadgets for each feature:

- **Fully supported** — Atlassian Gadgets provides full support for this feature. It will work on an Atlassian container.
- **Partially supported** — Atlassian Gadgets supports some of the functionality provided by this feature. The table above tells you which aspects are supported.
- **Not referenced** — Atlassian Gadgets does not make use of this feature, so it will have no effect on an Atlassian container. You may include the feature in your gadget if you want it to be used in other containers.
- **Not supported** — Atlassian Gadgets does not support this feature and your gadget will not work on an Atlassian container if you `<Require>` the feature. To use the feature in containers where it is available while still remaining compatible with Atlassian Gadgets, you must use the `<Optional>` element to specify the feature, then test for the presence of the feature using the `gadgets.util.hasFeature` function.

In the above table we have also categorised the features as follows:

- **Specified by OpenSocial** — Most features are described in the OpenSocial JavaScript API.
- **Specified by Atlassian Gadgets** — Additional features are provided specifically by the Atlassian Gadgets framework. These features will work on an Atlassian Gadgets container but probably not on another OpenSocial container.

RELATED TOPICS

Creating your Gadget XML Specification
Using the Atlassian Gadgets JavaScript Framework
Writing an Atlassian Gadget
Gadgets and Dashboards Development Hub

Packaging your Gadget as an Atlassian Plugin

This page tells you how to add your gadget to an Atlassian application (JIRA, Confluence, etc) as a plugin. In short, you will add a `<gadget>` module type to your `atlassian-plugin.xml` file.

On this page:

- Prerequisites
- Purpose of the Gadget Module Type
- Configuration
    - Attributes
- Example
- URL for Published Gadgets

Prerequisites

> **This list of prerequisites is incomplete**
> TO DO: We need to add instructions for configuring specific Atlassian applications to support AGSL-1.

- Your Atlassian application must be configured to support Atlassian Gadgets.
- Your Atlassian application must support the Atlassian Plugin Framework version 2.2 or later (see the version compatibility matrix).
- Your plugin must be an OSGi-based plugin, as supported by the Atlassian Plugin Framework.
- You will need to install the REST plugin module type.

Purpose of the Gadget Module Type

Gadget plugin modules enable you to add your gadget to an Atlassian application (JIRA, Confluence, etc) as a plugin. Your gadget can then make use of the application's remote API to fetch data and interact with the application.

Configuration

The element for the Gadget plugin module is **`gadget`**. It allows the following attributes for configuration:

Attributes

Name	Required	Description	Default

key	✓	The `key` attribute is a standard module key, so it is required and must be unique within the plugin across **all** module types. Atlassian Gadgets does not use this key for anything special, so you can choose any key you like.	
location	✓	The `location` attribute can be either the relative path to a resource within the plugin, or the absolute URL of an externally-hosted gadget.	

Example

The syntax of the module type is:

```
<atlassian-plugin name="Hello World" key="example.plugin.helloworld" plugins-version="2">
   <plugin-info>
      <description>A basic gadget module</description>
      <vendor name="Atlassian Software Systems" url="http://www.atlassian.com"/>
      <version>1.0</version>
   </plugin-info>

   <gadget key="unique-gadget-key" location="path/to/gadget.xml"/>

</atlassian-plugin>
```

URL for Published Gadgets

Gadgets published by an Atlassian container (such as JIRA or Confluence) are provided by the REST plugin module built into the Atlassian Gadgets framework. The URL of published gadgets has the following format — with context:

```
http://my-server.com:port/my-context/rest/gadgets/1.0/g/my-plugin.key:my-gadget/my-path/my-gadget.xml
```

Or without context:

```
http://my-app.my-server.com:port/rest/gadgets/1.0/g/my-plugin.key:my-gadget/my-path/my-gadget.xml
```

Example:

```
http://mycompany.com/jira/rest/gadgets/1.0/g/com.atlassian.streams.streams-jira-plugin:activitystream-gad
```

RELATED TOPICS

Creating your Gadget XML Specification
Writing an Atlassian Gadget
Gadgets and Dashboards Development Hub

Internationalising your Gadget

This page gives an overview of the standard gadget API for internationalisation, and details of the features provided specifically by the Atlassian Gadgets framework.

On this page:

- Overview of the Google Gadgets API for Internationalisation and Localisation
- Additional Features Provided by Atlassian Gadgets
    - `#supportedLocales` Directive

Overview of the Google Gadgets API for Internationalisation and Localisation

Atlassian gadgets support internationalisation and localisation as defined in the Google gadgets API.

Message Bundles

Any text that is visible to users, and therefore needs translating, will live in external XML files called message bundles. You will have one message bundle for the original language, plus a number of additional message bundles for the other supported languages.

Message bundles are XML files with a top element of `<messagebundle>`. Each message bundle contains the translated strings for a given locale. Each string is identified by a unique name that is the same across all your message bundles.

Message bundles can be hosted at any URL and can be shared between applications.

Using the Message Bundles in your Gadget

In your gadget XML specification you will have a list of `<Locale>` elements that specify the message bundles used by your gadget. The `<Locale>` tag lives in the `<ModulePrefs>` section of the gadget XML specification and links the language to the relevant message bundle.

Please refer to the Google documentation for more details.

Additional Features Provided by Atlassian Gadgets

The Atlassian Gadgets framework allows you to specify #-directives in your gadget specification XML file that will be replaced by generated content at run time. These #-directives are provided by the Atlassian Gadget Publisher plugin. They work for any gadget that is provided as a plugin in an Atlassian application. The #-directives do **not** work for gadgets that are served from an external web server.

ℹ #-directives are sometimes also called 'macros' or 'pseudo-macros'.

You can use the #-directive described below with any Atlassian application that supports language packs

Alternatively, you can choose to create your own message bundles and code your own `<Locale>` elements instead of using the #-directive described below.

`#supportedLocales` Directive

The Atlassian Gadgets framework scans your gadget specification XML file. If it encounters a `#supportedLocales` directive, it will:

- Retrieve the language and country information for all of the installed language packs.
- Replace the `#supportedLocales` directive with a `<Locale>` element for each combination of language and country, as required in the gadget XML specification.

The `messages` attribute of each `<Locale>` element will point to a location in the application that publishes the gadget. This location provides a dynamically-generated message bundle created by finding all of the internationalisation property keys that begin with one of the prefixes specified in the `#supportedLocales` directive.

Parent Element: `<ModulePrefs>`

Format:
`#supportedLocales("mygadget.prefix")`
where `mygadget.prefix` is a prefix common to the message keys defined in your plugin's internationalisation file. It is possible to specify multiple prefixes by separating them with commas, for example `#supportedLocales("gadget.common,gadget.introduction")`.

Example:
Here is a snippet of the gadget XML using the `#supportedLocales` directive:

```
<ModulePrefs
      title="__MSG_gadget.introduction.title__"
      directory_title="__MSG_gadget.introduction.title__"
      description="__MSG_gadget.introduction.description__">
   <Require feature="dynamic-height"/>
   <Require feature="settitle"/>

   #supportedLocales("gadget.introduction")
</ModulePrefs>
```

At run time, the above section of the gadget XML will be replaced with the following:

```
<ModulePrefs
      title="__MSG_gadget.introduction.title__"
      directory_title="__MSG_gadget.introduction.title__"
      description="__MSG_gadget.introduction.description__">
   <Require feature="dynamic-height"/>
   <Require feature="settitle"/>

   <Locale messages="http://myhost.com
     :port/myapp/rest/gadgets/1.0/msg/gadget.introduction/ALL_ALL.xml"/>
   <Locale lang="fr" country="FR" messages="http://myhost.com
     :port/myapp/rest/gadgets/1.0/msg/gadget.introduction/fr_FR.xml"/>
   <Locale lang="de" country="DE" messages="http://myhost.com
     :port/myapp/rest/gadgets/1.0/msg/gadget.introduction/de_DE.xml"/>
   <Locale lang="pt" country="BR" messages="http://myhost.com
     :port/myapp/rest/gadgets/1.0/msg/gadget.introduction/pt_BR.xml"/>
   <Locale lang="it" country="IT" messages="http://myhost.com
     :port/myapp/rest/gadgets/1.0/msg/gadget.introduction/it_IT.xml"/>
   <Locale lang="de" country="CH" messages="http://myhost.com
     :port/myapp/rest/gadgets/1.0/msg/gadget.introduction/de_CH.xml"/>
   <Locale lang="en" country="US" messages="http://myhost.com
     :port/myapp/rest/gadgets/1.0/msg/gadget.introduction/en_US.xml"/>
</ModulePrefs>
```

RELATED TOPICS

Creating your Gadget XML Specification
Writing an Atlassian Gadget
Gadgets and Dashboards Development Hub

Using Web Resources in your Gadget

The Atlassian Gadgets framework allows you to specify #-directives in your gadget specification XML file that will be replaced by generated content at run time. These #-directives are provided by the Atlassian Gadget Publisher plugin. They work for any gadget that is provided as a plugin in an Atlassian application. The #-directives do **not** work for gadgets that are served from an external web server.

#-directives are sometimes also called 'macros' or 'pseudo-macros'.

You can use these #-directives to specify web resources needed by your gadget.

On this page:

- `#requireResource` and `#includeResources` Directives

`#requireResource` and `#includeResources` Directives

The `#requireResource` and `#includeResources` directives work with the `WebResourceManager` as used in JIRA (see JIRA web resources), Confluence (see Confluence web resources) and other Atlassian applications. In turn, the `WebResourceManager` uses the Web Resource Plugin Module type defined in the Atlassian Plugin Framework.

The #-directives correspond directly to the `requireResource` and `includeResources` methods in `WebResourceManager`.

You can include multiple `#requireResource` directives.

The gadget processor proceeds as follows:

- When the gadget processor finds a `#requireResource` directive, it adds the specified resource to a list.
- When the gadget processor finds a `#includeResources` directive, it processes the list of web resources:
  - First it resolves dependencies transitively (since web resources can depend on other web resource modules, potentially in another plugin, which can themselves depend on other web resource modules, etc).
  - Then it eliminates any duplicates from the fully resolved list.

For example, if my gadget has both `#requireResource("com.atlassian.gadgets:common-resources")` and `#requireResource("com.atlassian.gadgets.mygadget:mygadget-resources"`, and both of those have a dependency on `com.atlassian.auiplugin:ajs`, the gadget processor will only include AJS once.

Parent Element: `<Content>`

Format of `#requireResource`:
`#requireResource("`*`plugin.key`*`:`*`module-key`*`")`
Each `#requireResource` directive names the complete module key (*`plugin.key`*`:`*`module-key`*) for a `web-resource` plugin module.

Format of `#includeResources`:
`#includeResources()`

Example:
Here is a snippet of gadget XML using the `#requireResource` directive:

```
<Content type="html">
<![CDATA[
   #requireResource("com.atlassian.jira.gadgets:common")
   #includeResources()

   <div id="intro-content">
      Hello, world
   </div>
]]>
</Content>
```

At run time, the above section of the gadget XML will be replaced with the following:

```
<Content type="html">
<![CDATA[
   <link type="text/css" rel="stylesheet"
      href="http://myhost.com
        :port/myapp/s/448/1/1.1.7/_/download/batch/com.atlassian.auiplugin:ajs/com.atlassian.auiplugin:ajs.
      media="all">
   <!--[if IE]>
   <link type="text/css" rel="stylesheet"
      href="http://myhost.com
        :port/myapp/s/448/1/1.1.7/_/download/batch/com.atlassian.auiplugin:ajs/com.atlassian.auiplugin:ajs.
      media="all">
   <![endif]-->
   <script type="text/javascript"
      src="http://myhost.com
        :port/myapp/s/448/1/1.1.7/_/download/batch/com.atlassian.auiplugin:ajs/com.atlassian.auiplugin:ajs.
        ></script>
   <script type="text/javascript"
      src="http://myhost.com
        :port/myapp/s/448/1/4.0.0/_/download/batch/com.atlassian.jira.gadgets:common/com.atlassian.jira.gad
        ></script>
   <link type="text/css" rel="stylesheet"
      href="http://myhost.com
        :port/myapp/s/448/1/4.0.0/_/download/batch/com.atlassian.jira.gadgets:common/com.atlassian.jira.gad
      media="all"/>

   <div id="intro-content">
      Hello, world
   </div>
]]>
</Content>
```

RELATED TOPICS

Creating your Gadget XML Specification
Writing an Atlassian Gadget
Gadgets and Dashboards Development Hub

Using Atlassian REST APIs in your Gadget

Documentation under development

In the gadgets world, the UI is rendered entirely using HTML and JavaScript (on the dashboard server). To get dynamic data into the gadget, you will make Ajax calls using `makeRequest` back to the originating server.

Why a special API for making requests? When we render gadgets, even if the gadget specification is coming from your server out there, it is pulled into the dashboard server and parsed and rendered into HTML on the dashboard server. So your JavaScript can only talk to the server where the dashboard is running. The `makeRequest` call will proxy back (in a process that is called 'phoning home') to the originating server to request data, using one of the following:

- The REST APIs that the application already supports.
- A custom servlet that you build in a plugin for that product.
- A custom REST plugin module that is built into the Atlassian Gadgets framework. This is the recommended option.

Providing User Authentication for Gadgets

ⓘ **Documentation under development**

This page is an overview of user authentication in gadgets. There is another topic on using the authentication mechanism provided by the Atlassian Gadgets JavaScript Framework.

To cover:

- OAuth
- Trusted apps
- Other forms

Introduction to OAuth

The central principle behind the OAuth protocol is:
If you want a client program to access a server somewhere, there should be a more secure way than asking the user to enter their username and password into the client program which then sends it to the server. OAuth provides another way to authenticate, using secure tokens passed between two servers.

For example, if a user puts a JIRA Issues gadget onto their iGoogle page, they should not need to enter their JIRA username and password into iGoogle.

RELATED TOPICS

Using Authentication in your Gadget
Using the Atlassian Gadgets JavaScript Framework

Using the Atlassian Gadgets JavaScript Framework

This document assumes that you are familiar with (or have the documentation available for) writing gadgets and you have read Writing an Atlassian Gadget.

Using the Atlassian Gadgets JavaScript Framework is the preferred and recommended way of developing gadgets for Atlassian applications.

ⓘ **JavaScript Framework is currently in JIRA only**
The Atlassian Gadgets JavaScript Framework is currently part of the JIRA project. It will eventually be moved into Atlassian Gadgets. See AG-622. Until that merge is completed, you will only be able to use the framework if you are developing gadgets for JIRA.

On this page:

- Introduction
- Using the Framework

Introduction

During development of our own gadgets, we realised there were a lot of common requirements and functionality between gadgets. This led to the development of the Atlassian Gadgets JavaScript Framework. This framework is the basis of most of the gadgets developed at Atlassian.

Terminology: On this page and its child pages,

- when we refer to the '**framework**', we mean the Atlassian Gadgets JavaScript Framework.
- when we refer to a '**gadget**', we mean a gadget JavaScript object.

Feature overview:

- Authentication
  - Brokers a Trusted Applications connection if available and required
  - Brokers an OAuth connection if available and required
- View Helpers
  - Automatic reloading of a gadget based on a time interval
  - Automatic reloading of a gadget on browser window resize
  - Automatic gadget resizing on browser window resize
  - Parallel Ajax resource loading for the view
- Configuration
  - Permission-based gadget configuration
  - Configuration screen display on initial gadget load
  - Complex configuration form building

    - Validation of configuration
    - Inline error display
    - Saving of parameters
    - Parallel Ajax resource loading for configuration form
- jQuery-style remoting
    - A wrapper over `gadgets.io.makeRequest` supplying a cleaner and more common interface for Ajax calls
- Cookie storage
    - Cookie storage and retrieval on a gadget by gadget basis
- Performance
    - Conversion of proxied remote calls into direct calls if possible
- Common styling
    - Loading of screens, standard icons, common CSS

Using the Framework

These examples assume that you are familiar with the gadget XML format and Atlassian's customisations to it.

Here is a basic stub for using the framework:

```
<?xml version="1.0" encoding="UTF-8" ?>
<Module>
   <ModulePrefs title="__MSG_gadget.title__" directory_title="__MSG_gadget.title__" description=
     "__MSG_gadget.description__">
      <Require feature="dynamic-height" />
      <Require feature="oauthpopup" />
      <Require feature="setprefs" />
      <Require feature="settitle" />
      <Require feature="views" />
      <Optional feature="atlassian.util" />
      #oauth
      #supportedLocales("gadget.common,your.gadget.resource.prefix")
   </ModulePrefs>
   <Content type="html">
   <![CDATA[
      /* (1) This resource is required for using the gadget framework */
      #requireResource("com.atlassian.jira.gadgets:common")
      #includeResources()

      <script type="text/javascript">
         (function () {
            /* (2) Construct and initialise the gadget */
            var gadget = AJS.Gadget({
               baseUrl: "__ATLASSIAN_BASE_URL__", /* (3) Used to make base url available to gadget */
               view: {...view-options...} /* (4) Defines the view logic */
            });
         })();
      </script>
   ]]>
   </Content>
</Module>
```

The lines of interest are:

1. This includes all required JavaScript and CSS resources needed to use the framework. See Using Web Resources in your Gadget.
2. This constructs the gadget object and initialises it with the passed in options.
3. The framework needs access to the base URL, so we pass it in. See Using Substitution Variables and Directives in your Gadget.
4. This initialises the view.

> ℹ The `requireResource` #-directive will change once the framework is extracted from JIRA.

RELATED TOPICS

- Creating a Gadget JavaScript Object
    - Field Definitions
- Gadget Object API
- Gadget Developers' JavaScript Cookbook
    - Adding a Chart to the Issue Navigator
    - Adding a Reload Option to your Gadget
    - Adding Something to your Gadget Footer
    - Adjusting the Gadget Height when the Window is Resized
    - Making Ajax Calls
    - Making your Gadget Reload when Resized
    - Restricting Edit Permissions on your Gadget Preferences

    - Showing the Config Screen on Initial Load Only
    - Specifying Required Features for the Framework
    - Specifying Required Resources for the Framework
    - Theming your Gadget Dynamically
    - Using Authentication in your Gadget
    - Using Cookies in your Gadget
    - Using Special UserPrefs for the Framework

Including Features into your Gadget
Using Web Resources in your Gadget
Writing an Atlassian Gadget
Gadgets and Dashboards Development Hub

Creating a Gadget JavaScript Object

This page tells you how to create a gadget using the Atlassian Gadgets JavaScript Framework.

> **JavaScript Framework is currently in JIRA only**
> The Atlassian Gadgets JavaScript Framework is currently part of the JIRA project. It will eventually be moved into Atlassian Gadgets. See AG-622. Until that merge is completed, you will only be able to use the framework if you are developing gadgets for JIRA.

On this page:

High-Level Format for Creating a Gadget

The framework uses a declarative approach in terms of configuration. View, configuration and authorisation parameters are passed into the constructor and used to configure the gadget. See Gadget Object API for methods available on the gadget object.

The top-level options are:

```
var gadget = AJS.Gadget({
      baseUrl: "__ATLASSIAN_BASE_URL__",
      useOauth: ...,
      config: ...,
      view:...
});
```

Where:

- **baseUrl** — A required option used to pass the base URL to the framework. The framework uses this URL for prefixing relative Ajax requests and also makes it available from `gadget.getBaseUrl()` and `AJS.gadget.getBaseUrl()`.
- **useOauth** — An optional parameter that is used to configure the type of authentication used. Below is a summary of the possible values. There is more detail in the authentication section below:
    - **"always"** — Forces all requests to use OAuth. Please note that this removes the ability to provide data to anonymous users (REST resources with the annotation `@AnonymousAllowed`).
    - ***url*** — Here you must provide a URL that requires authentication. Typically we use `"/rest/gadget/1.0/currentUser"`. The framework will then try to broker an authenticated connection in the following order:
        1. Current session (a gadget being served locally, e.g. JIRA displaying a JIRA gadget), then
        2. A Trusted Applications connection, then
        3. OAuth. A gadget being served in a non-Atlassian container will typically use OAuth.
- **config** — You can use this optional parameter to define the configuration form. If this parameter is not defined, it is assumed that the gadget has no configuration options. Please note that this is not the built-in Opensocial Gadget configuration screen. The latter did not meet our needs so we decided to implement our own. See below.
- **view** — This object defines the view to display. See below.

Authentication

The framework will automatically try to broker an authenticated connection to the server, based on the URL provided in the `useOauth`

configuration option described above. If the value passed in is `"always"`, the framework will always try and use OAuth. If the value passed in is a URL, the framework will determine the best method of invoking that URL. The URL must not allow anonymous access. JIRA currently provides the following REST resource:
`/rest/gadget/1.0/currentUser`

The framework tries to connect to the server in the following order:

1. **Local** — We can determine whether the gadget is being served locally (e.g. JIRA serving out a JIRA gadget) and then we can just sue the local browser to talking to the server. This is the most efficient method of communication as no proxies are used.
2. **Trusted Applications** — If a successful request returns from the supplied resource when no OAuth token is passed, it must be a trusted application. We rely on the container server to use its Trusted Applications connection to the remote server for authentication.
3. **OAuth** - If the previous two methods fail, we fall back to OAuth.

More: Using Authentication in your Gadget

Configuration Form

Early on in our Gadget development, we realised that the OpenSocial gadgets' mechanism for creating configuration forms was not going to meet our needs. The fields must all be known beforehand, options are static and fields cannot be manipulated programatically. OpenSocial also only supports very basic fields. Our framework offers all the options available in the Opensocial Gadget framework for creating a configuration form. In addition, our framework allows for Ajax populated lists, custom fields and even dynamically included fields.

Typically, when loaded, a gadget will show a config screen first, then the normal gadget view. The user can choose to view it in canvas view if defined.

In order to take full advantage of the features in the framework, you should include the following in your gadget XML:

```
<Require feature="setprefs" />
<Require feature="views" />
<Require feature="dynamic-height" />
```

Where:

- **setprefs** — This is required to persist user preferences.
- **views** — This is used to determine whether the current user can edit the preferences or not. Please note that this is an Atlassian-specific extension to the feature and will not work in other containers. The edit button will always be shown in other containers.
- **dynamic-height** — Provides dynamic height adjustment in the config screen, view creation and on resize.

The `config` object consists of two parts: Ajax option definitions and a descriptor. When the form is generated, the Ajax options are retrieved and then fed into the descriptor which then returns an object fully defining the configuration form. The framework will then render the form.
When the form is submitted, the form will use the passed in validation URL to validate the form and if successful, will then save the preferences to the server. If validation fails, the form will display the errors inline.

ℹ All fields on the configuration form will be persisted as UserPrefs against the gadget.

```
...
config: {
      descriptor: function(){...},
      args: {Function, Array}
},
...
```

Where:

- **descriptor** — This function returns a new Configuration Descriptor
- **args** — Either an array of objects or a function that returns one. The objects in this array have two keys:
  - (String) key — Name of key that will be used to access the data from within the template.
  - (Object) ajaxOptions — Set of request options to retrieve data from remote resource. Options are available here. If a String is used, it will just retrieve the REST resource assuming it is JSON.

Ajax Options

This can either be an array of options to retrieve, or a function that returns a similar array.

```
...
args: [
      {
            key: "key to access options by",
            ajaxOptions:  "/rest/gadget/1.0/projects" /* this can also be a more complex JQuery style remote call */
      },
      ... more options to retrieve
]
...
```

Where:

- **key** — The descriptor will be able to access the results of the Ajax call via this key, e.g. `args.key`.
- **ajaxOptions** — If this is a string, the framework assumes it is the URL to a REST resource returning JSON. However, this can be a more complex jQuery-style remote call definition. Options are available here.

jQuery style remote Ajax options

```
...
args: [
      {
            key: "project",
            ajaxOptions:  {
                  url: "/rest/gadget/1.0/projects",
                  data: {
                        restrictToPermission: "create"
                  },
                  type: "POST"
            }
      },
      ... more options to retrieve
]
...
```

Note: This is just a fictitious REST endpoint. The example will not work.

Configuration Descriptor

The configuration descriptor is a function that returns an object defining the configuration form. After the Ajax options have been retrieved, they are passed into the function to get the object that defines the form.

The function is called within the scope of the gadget. Hence `this` refers to the current gadget object.

```
...
descriptor: function(args){
      /* This is called within the scope of the current gadget.  "this" refers to the current gadget */
      return {
            action: "validation url", /* A url to validate the form against */
            theme: "", /* The layout of the form - "top-label" or "long-label" */
            fields:[
                  {
                        /* Field descriptor */
                  },
                  ... more fields
            ]
      };
},
...
```

Where:

- **args** — This is an object populated by the results of the Ajax options defined in the `args` parameter of the `config` object. If you retrieve a list of projects via Ajax, this list can be accessed via: `args.projects`
- **action** — This is a URL pointing to a REST resource that will validate the form and return appropriate errors if the form contains invalid data. Please see [expected REST endpoint formats] for the structure and return codes of the response.
- **theme** — This tells the form how to lay itself out. There are two possible values:

    - **"top-label"** — This displays the field label above the field. Useful for narrow forms.
    - **"long-label"** — This displays the label on the same line as the field. Better suited to forms with a lot of width available.
- **fields** — This is an array of field objects.

More: Theming your Gadget Dynamically

Configuration Fields

Configuration forms are made up of list of fields. Typically, these fields relate to UserPref fields defined in the gadget XML. We usually define our UserPrefs in the XML as hidden UserPrefs. This will ensure that the field will not appear in the OpenSocial gadget configuration form.
E.g.

```
<UserPref name="project" datatype="hidden" />
```

We support the following types of fields (details are on the Field Definitions page):

- Hidden fields — A hidden input area on the form whose value is passed back to the server during validation and preference saving.
- Text input — A single line text input.
- Text area — A text area for accepting larger text inputs.
- Select fields — A simple selection list that displays a list of options. It is possible to display a list of options retrieved from a REST resource.
- Multi-select fields — A multi-select field where it is possible to select multiple values. It is possible to display a list of options retrieved from a REST resource.
- Radio button group — A set of radio buttons for selecting unique values. It is possible to display a list of options retrieved from a REST resource.
- Check box group — A set of check boxes to select multiple values. It is possible to display a list of options retrieved from a REST resource.
- Custom fields — A user defined field. This allows users to insert arbitrary HTML into the form.
- Callback builder — An empty `<div/>` is inserted into the form. After form creation, a callback handler is called passing in the jQuery wrapped `div`. This is the most powerful field creation technique and can be used to build highly interactive fields.

More:

- Field Definitions
- Showing the Config Screen on Initial Load Only
- Specifying Required Resources for the Framework

View

This object defines the view to display, as shown in the high level overview above.

```
config: {...},
view: {
      enableReload: true,
      onResizeReload: true,
      onResizeAdjustHeight: false,
      template: function (args) {},
      args: [{
            key: "chart",
            ajaxOptions: {...}
      }]
}
```

Where:

- **enableReload** — (Optional.) Adds a reload icon to the bottom of the gadget and adds a reload period option to the configuration form. The gadget will redraw the view every X minutes.
- **onResizeReload** — (Optional.) When the browser window resizes or the layout changes, the gadget redraws the view. This is used in our chart gadgets so that the chart image can be regenerated to size and granularity.
- **onResizeAdjustHeight** — (Optional.) When the browser window resizes or the layout changes, `gadget.adjustHeight()` is called to ensure that if any lines changed their wrapping the gadget height will adjust accordingly. You must include the `dynamic-height` feature for this to work.
- **template** — A function that is responsible for creating the view. See below.
- **args** — Either an array of objects or function that returns one. The results of these remote calls are passed in to the template during rendering.
    - **key** — Name of the key that will be used to access the data from within the template.
    - **ajaxOptions** — Set of request options to retrieve data from a remote resource. Options are jQuery Ajax options.

View Template Function

The `template` function is responsible for rendering the gadget's view. The function is run within the scope of the gadget so '`this`' refers to the gadget. See the Gadget Object API for the methods available.

The template creates HTML using jQuery. It does not include styling. Styling is done later via CSS.

The function takes one argument – `args` – a map containing the response from the Ajax calls. This map is keyed by the specified key.

The function must be able to be called multiple times for reloading and resizing.

Example:

```
view: {
      enableReload: true,
      onResizeAdjustHeight: true,
      template: function (args) {
            var gadget = this;
            var filters = args.favFilters.filters;

            if (!filters){
                  gadget.getView().removeClass("loading").html(
                     "<p>__MSG_gadget.favourite.filters.no.favourites__</p>");
            } else {
                  var list = AJS.$("<ul/>").attr("id", "filter-list");

                  AJS.$(filters).each(function(){
                        list.append(
                              AJS.$("<li/>").append(
                                    AJS.$("<div/>").addClass("filter-name").append(
                                          AJS.$("<a/>").attr({
                                                target: "_parent",
                                                title: gadgets.util.escapeString(this.description),
                                                href:
                                                   "__ATLASSIAN_BASE_URL__/secure/IssueNavigator.jspa?mode=h
                                                   + this.value
                                          }).text(this.label)
                                    )
                              ).append(
                                    AJS.$("<div/>").addClass("filter-count").text(this.count)
                              ).click(function () {
                                    if (window.parent){
                                          window.parent.location =
                                             "__ATLASSIAN_BASE_URL__/secure/IssueNavigator.jspa?mode=hide&re
                                             + this.value;
                                    } else {
                                          window.location =
                                             "__ATLASSIAN_BASE_URL__/secure/IssueNavigator.jspa?mode=hide&re
                                             + this.value;
                                    }
                              })
                        );
                  });
                  gadget.getView().html(list);
            }
      },
      args: [{
            key: "favFilters",
            ajaxOptions: function () {
                  return {
                        url: "/rest/gadget/1.0/favfilters",
                        data:  {
                              showCounts : this.getPref("showCounts")
                        }
                  };
            }
      }]
}
```

RELATED TOPICS

Using the Atlassian Gadgets JavaScript Framework
Writing an Atlassian Gadget
Gadgets and Dashboards Development Hub

Field Definitions

This page gives more detail about the field types that were introduced in Creating a Gadget JavaScript Object.

On this page:

Hidden Fields

Sometimes you want to pass values back to the server from a form submission, but you do not want the user to be able to change those values. You can use hidden fields to achieve this.

The following code defines a hidden field:

```
fields: [
      {
            userpref: "isConfigured",
            type: "hidden",
            value: "true"  /* This example is hard coded but it could be dynamically
                                       generated or retrieved from the gadget prefs.*/
      },
      ... /* Other fields */
]
```

Where:

- **userpref** — The name of the field. Should be the name of the UserPref to save.
- **type** — Use `"hidden"` to specify that this field is hidden.
- **value** — The value that will be passed back to the server for validation and will then be saved.

Text Input

This field type will be displayed as a simple text field. If the the `type` attribute is not defined, the field will be of this type by default.

```
fields: [
      {
            id: "numToDisplay",
            userpref: "numToDisplay",
            class: "numField"
            label: gadget.getMsg("gadget.common.num.label"),
            description: gadget.getMsg("gadget.common.num.description"),
            type: "text",
            value: gadget.getPref("numToDisplay")
      },
      ... /* Other fields */
]
```

Where:

- **id** — (Optional.) The HTML `id` to give this field. Defaults to the value of the `userpref` field.
- **userpref** — The name of the field. Should be the name of the UserPref to save.
- **class** — (Optional.) The HTML class to apply to this field. If omitted, will default to **type**.
- **label** — The label to display next to the field.
- **description** — (Optional.) The description of the field.
- **type** — (Optional.) If the value is `"text"` or omitted, a text field (i.e. the type discussed in this paragraph) is rendered.
- **value** — (Optional.) The initial value of the field. If this is a UserPref, we usually populate it with the UserPref value.

Text Area

This field type creates a text input area in the form, used for large text inputs.

```
fields: [
      {
            id: "textToDisplay",
            userpref: "textToDisplay",
            class: "textBox"
            label: gadget.getMsg("gadget.common.display.text.label"),
            description: gadget.getMsg("gadget.common.display.text.description"),
            type: "textarea",
            value: gadget.getPref("textToDisplay")
      },
      ... /* Other fields */
]
```

Where:

- **id** — (Optional.) The HTML `id` to give this field. Defaults to the value of the `userpref` field.
- **userpref** — The name of the field. Should be the name of the UserPref to save.
- **class** — (Optional.) The HTML class to apply to this field.
- **label** — The label to display next to the field.
- **description** — (Optional.) The description of the field.
- **type** — Value must be `"textarea"` for this type of field.
- **value** — (Optional.) The initial value of the field. If this is a UserPref, we usually populate it with the UserPref value.

Select

This field type is a simple selection list. As the AJAX options have been retrieved and are available during the creation of the field decscriptor, it is trivial to use them as options for the selection list. The results can be used raw or transformed.

```
fields: [
      {
            id: "cumulative-field",
            class: "cumulative-select-list",
            userpref: "isCumlative",
            label: gadget.getMsg("gadget.common.cumulative.label"),
            description:gadget.getMsg("gadget.common.cumulative.description"),
            type: "select",
            selected: gadget.getPref("isCumlative"),
            options:[
                  {  /* These options are hard-coded but could easily be the results of an AJAX call
                           (E.g. args.projects) or be calculated at run time (E.g. the next three
                             days) /*
                        label:gadget.getMsg("gadget.common.yes"),
                        value:"true"
                  },
                  {
                        label:gadget.getMsg("gadget.common.no"),
                        value:"false"
                  }
            ]
      },
      ... /* Other fields */
]
```

Where:

- **id** — (Optional.) The HTML `id` to give this field. Defaults to the value of the `userpref` field.
- **userpref** — The name of the field. Should be the name of the UserPref to save.
- **class** — (Optional.) The HTML class to apply to this field. Defaults to `"select"`.
- **label** — The label to display next to the field.
- **description** — (Optional.) The description of the field.
- **type** — Value must be `"select"` for this type of field.
- **selected** — (Optional.) The value of initially selected option. If this is a UserPref, we usually populate it with the UserPref value.
- **options** – An array of `option` pairs or groups.

An option is defined as:

```
{
      id : "option-id",
      label: "A human readable label",
      value: "option-value",
      selected: true
}
```

Where:

- **id** — (Optional.) The `id` of the option.
- **label** — The displayable label for the option.
- **value** — The value to save if this option is selected.
- **selected** — (Optional.) Specifies whether or not to initially select this option. If only one option can be selected (selection list or radio group) it is more convenient to specify the selected value on the field definition.

Option groups are used in the following example:

```
fields: [
      {
            userpref: "groupedOptions",
            label: gadget.getMsg("gadget.common.grouped.label"),
            type: "select",
            selected: gadget.getPref("groupedOptions"),
            options:[
                  {
                        group: {
                              label: "First Option Group",
                              options: [
                                    {
                                          label: "First Group -> First Option",
                                          value: "group-1_opt-1",
                                    },
                                    {
                                          label: "First Group -> Second Option",
                                          value: "group-1_opt-2",
                                    }
                              ]
                        }
                  },
                  { /* You can mix groups and non-group options */
                        label:"No Group Options",
                        value:"group-N/A_opt-1"
                  },
                  {
                        group: {
                              label: "Second Option Group",
                              options: [
                                    {
                                          label: "Second Group -> First Option",
                                          value: "group-2_opt-1",
                                    }
                              ]
                        }
                  }
            ]
      },
      ... /* Other fields */
]
```

Where:

- **group** — An object containing the sub-options.
- **group.label** — The label for the option group.
- **group.options** — The options to display under the option group.

Multi Select

This field type produces a multi-select input value.

```
fields: [
      {
            id: "version-field",
            class: "multiVersionField",
            userpref: "selectedVersions",
            label: gadget.getMsg("gadget.common.versions.label"),
            description:gadget.getMsg("gadget.common.versions.description"),
            type: "multiselect",
            selected: gadget.getPref("selectedVersions"), /* Only use this if you only want value
              selected.
                                                                                     Otherwise specify
                                                                                       selection
                                                                                       state on the
                                                                                       option itself
                                                                                       */
            options:[ /* See Select Field for valid options */]
      },
      ... /* Other fields */
]
```

Where:

- **id** — (Optional.) The HTML `id` to give this field. Defaults to the value of the `userpref` field.
- **userpref** — The name of the field. Should be the name of the UserPref to save.
- **class** — (Optional.) The HTML class to apply to this field. Defaults to `"multi-select"`.
- **label** — The label to display next to the field.
- **description** — (Optional.) The description of the field.
- **type** — Value must be `"multiselect"` for this type of field.
- **selected** — (Optional.) The value of initially selected option. Only use this if you only want a single value selected. Otherwise specify selection state on the option itself.
- **options** — An array of `options` pairs or groups. See the Select field type (above) for valid options.

Radio Button Group

This field type produces a group of radio buttons for selecting a unique value from a list of choices.

```
fields: [
      {
            id: "security-level",
            class: "security-radio",
            userpref: "securityLevel",
            label: gadget.getMsg("gadget.common.security.label"),
            description:gadget.getMsg("gadget.common.security.description"),
            type: "radio",
            selected: gadget.getPref("securityLevel"),
            options:[
                  {  /* See Select Field for valid options */
                        label: "Publicly Available",
                        value:"public"
                  },
                  {
                        label:"Private",
                        value:"private"
                  }
            ]
      },
      ... /* Other fields */
]
```

Where:

- **id** — (Optional.) The HTML `id` to give this fieldset. Defaults to the value of the `userpref` field.
- **userpref** — The name of the field. Should be the name of the UserPref to save.
- **class** — (Optional.) The HTML class to apply to this fieldset.
- **label** — The label to display next to the fieldset.
- **description** — (Optional.) The description of the field.
- **type** — Value must be `"radio"` for this type of field.
- **selected** — (Optional.) The value of the option to be selected initially. If this is a UserPref, we usually populate it with the UserPref value.
- **options** — An Array of `options` pairs or groups. See the Select field type (above) for valid options.

Checkbox Group

This field type produces a group of checkboxes for selecting multiple values.

```
fields: [
      {
            id: "group-select",
            class: "group-selector-checkboxes",
            userpref: "selectedGroups",
            label: gadget.getMsg("gadget.common.groups.label"),
            description:gadget.getMsg("gadget.common.groups.description"),
            type: "checkbox",
            options:[
                  {  /* See Select Field for valid options */
                        id: "group-users",
                        label: "Users",
                        value: "users"
                  },
                  {
                        id: "group-devs",
                        label: "Developers",
                        value: "developers",
                        selected : true
                  },
                  {
                        id: "group-admins",
                        label: "Administrators",
                        value: "admins",
                        selected : true
                  }
            ]
      },
      ... /* Other fields */
]
```

Where:

- **id** — (Optional.) The HTML `id` to give this fieldset. Defaults to the value of the `userpref` field.
- **userpref** — The name of the field. Should be the name of the UserPref to save.
- **class** — (Optional.) The HTML class to apply to this fieldset.
- **label** — The label to display next to the fieldset.
- **description** — (Optional.) The description of the field.
- **type** — Value must be `"checkbox"` for this type of field.
- **selected** — (Optional.) The value of the option to be selected initially. Only use this if you only want a single value selected. Otherwise specify the selection state on the option itself.
- **options** — An Array of `options` pairs or groups. See the Select field type (above) for valid options.

Custom Field

This field type simply injects the string into the form as HTML.

```
fields: [
      {
            label: gadget.getMsg("gadget.common.groups.label"),
            description:gadget.getMsg("gadget.common.groups.description"),
            type: "custom",
            template: function(){
                  /* Returned string is injected into form */
                  return "<div>An example of a custom field</div>";
            }
      },
      ... /* Other fields */
]
```

Where:

- **label** — The label to display next to the field.
- **description** — (Optional.) The description of the field.
- **type** — Value must be `"custom"` for this type of field.
- **template** – A function that returns a string to inject into the form.

Callback Builder

This is the most powerful and flexible type of field for configuration. The callback enables you to construct any HTML and attach event listeners to it.

```
fields: [
      {
            id: "my-callback-field",
            label: gadget.getMsg("gadget.common.builder.label"),
            description:gadget.getMsg("gadget.common.builder.description"),
            type: "callbackBuilder",
            callback: function(parentDiv){
                  parentDiv.append(
                        AJS.$("<input/>").attr({
                              id: "call-back-hidden-field",
                              type: "hidden",
                              name: userpref
                        }).val(gadget.getPref("myCallback))
                  );
            }
      },
      ... /* Other fields */
]
```

Where:

- **id** — The `id` given to the `div` that is passed into the builder.
- **label** — The label to display next to the field.
- **description** — (Optional.) The description of the field.
- **type** — Value must be `"callbackBuilder"` for this type of field.
- **callback** — A function that takes in a JQuery wrapped `div`. You can then manipulate the contents of that `div` and attach event handlers, e.g. click handlers.

RELATED TOPICS

Using the Atlassian Gadgets JavaScript Framework
Writing an Atlassian Gadget
Gadgets and Dashboards Development Hub

Gadget Object API

This page describes the methods available in each type of gadget object. This page is part of the documentation on the Atlassian Gadget Library.

JavaScript Framework is currently in JIRA only
The Atlassian Gadgets JavaScript Framework is currently part of the JIRA project. It will eventually be moved into Atlassian Gadgets. See AG-622. Until that merge is completed, you will only be able to use the framework if you are developing gadgets for JIRA.

On this page:

- Overview
- Standard Gadget
- Configured Gadget
- Configurable Gadget

Overview

Please refer to Creating a Gadget JavaScript Object for details on constructing a gadget object. The methods provided on this page can be called not only on the constructed object, but also from any method provided in the construction configs. All methods passed in as config parameters (e.g. the view template, the config descriptor, ...) are run in the scope of the gadget itself. Therefore, `this` refers to the gadget and any of the following methods can be called on `this`.

Under the hood, the constructor method `AJS.Gadget(...)` is a factory method that constructs a specific type of gadget depending on the config parameters passed in. The three kinds of gadgets are:

- Standard
- Configured (inherits all of the methods from Standard Gadget)
- Configurable (inherits all of the methods from Configured Gadget)

Each type is described below.

Standard Gadget

A Standard Gadget is constructed when a `view` parameter is passed in but no `config` parameter. This is useful when no configuration is needed for the gadget. An example is the Quick Issue Create gadget in JIRA.

All other gadget types extend the Standard Gadget type.

```
return {
      showMessage: function (type, msg, dismissible){}, /* Displays a message in dialogue box. */
      savePref: function (name, value){},               /* Saves user preferences locally and to the
        database. */
      setViewMode: function (){},                       /* Toggles class of gadget to the specified
        view. */
      getViewMode: function (){},                       /* Returns the current view mode as a string.
        For example "Canvas". */
      getBaseUrl: function (){},                        /* Helper function to get the context path
        for jira. */
      getPrefs: function (){},                          /* Gets user preference object. */
      getPref: function (name){},                       /* Some sugar for getting a preference by
        name */
      getPrefArray: function (name){},                  /* Retrieves a user pref array */
      getMsg: function (key){},                         /* Gets the i18n String */
      getGadget: function (){},                         /* Gets the gadget object, wrapper div for
        all gadget html (jQuery Object) */
      resize: function (){},                            /* Resizes iframe to fit content */
      showLoading: function (){},                       /* Shows loading indicator */
      hideLoading: function (){},                       /* Hides loading indicator */
      createCookie: function (name, value, days){},     /* Stores a value into a cookie, unique to
        this gadget. */
      readCookie: function (name){},                    /* Retrieves a previously stored cookie value
        */
      eraseCookie: function (name){}                    /* Removes a cookie value */
};
```

showMessage

Displays a message in a dialogue box.

```
showMessage: function (type, msg, dismissible) {}
```

Where:

- **type** — (String.) Style of message. Options include "error, info".
- **msg** — (String, Object.) An HTML string or jQuery object containing message.
- **dismissible** — (Boolean.) If set to false, no cancel button will be available.

savePref

Saves user preferences locally and to the database. In order to persist these values and have them available when gadget is reloaded, the `setprefs` feature must be declared as required in the gadget XML specification.

```
savePref: function (name, value) {}
```

Where:

- **name** — (String.) Name of preference to save.
- **value** - (String, Array.) Value (or values) to save to the database.

setViewMode

Toggles the class of the gadget to the specified view. This class is used to style the view accordingly.

```
setViewMode: function (mode) {}
```

Where:

- **mode** — The class to toggle on the gadget.

getViewMode

Returns the current view mode as a string. For example `"Canvas"`.

```
getViewMode: function () {}
```

getBaseUrl

Helper function to get the context path for JIRA. Necessary for remote requests.

```
getBaseUrl: function () {}
```

getPrefs

Gets user preference object.

```
getPrefs: function () {}
```

getPref

Gets a preference by name.

```
getPref: function (name) {}
```

Where:

- **name** — The name of the preference to retrieve.

getPrefArray

Retrieves a user preference array.

```
getPrefArray: function (name){}
```

Where:

- **name** — The name of the preference array to retrieve.

getMsg

Gets the i18n string from the included language bundles. Returns the key if it does not exist.

```
getMsg: function (key){}
```

Where:

- **key** — The key of the message to retrieve.

getGadget

Gets the gadget object, wrapper `div` for all gadget HTML (jQuery object).

```
getGadget: function (){}
```

resize

Resizes the iframe to fit the content.

```
resize: function (){}
```

showLoading

Shows an indicator that the gadget is loading.

```
showLoading: function (){}
```

hideLoading

Hides the loading indicator.

```
hideLoading: function (){}
```

createCookie

Stores a value in a cookie, unique to this gadget.

Use cookies with caution, so that the browser does not create too many cookies. They are necessary if you need to store a value for the current user rather than for the gadget. Where possible, use UserPrefs instead. UserPrefs will store values for the gadget, not the user.

```
createCookie: function (name, value, days){}
```

Where:

- **name** — The name (key) of the cookie to store.
- **value** — The value to store in the cookie.
- **days** — The number of days to keep the cookie active.

readCookie

Retrieve a previously stored cookie value.

```
readCookie: function (name){}
```

Where:

- **name** — The name of the cookie value to retrieve.

eraseCookie

Removes a cookie value.

```
eraseCookie: function (name){}
```

Where:

- **name** — The name of the cookie value to erase.

Configured Gadget

A Configured Gadget is constructed when `view` and `config` parameters are passed in but the current user does not have permission to edit the gadget's preferences. The gadget contains a view and footer.

This gadget has all of the same methods as a Standard Gadget plus the following:

```
return AJS.$.extend(getStandardInterface(), {
      getView: function(){},            /* Gets the view object, wrapper div for all view html
        (jQuery Object)  */
      showView: function(refresh){},    /* Display the view */
      getFooter: function(){}           /* Gets the footer object, wrapper div for all footer html
        (jQuery Object) */
});
```

getView

Gets the view object, wrapper `div` for all view HTML (jQuery object). This object is a `div` with the class of `"view"` and is contained within the object returned from `getGadget()`.

```
getView: function(){}
```

showView

Displays the view. When refreshing content, the view template is called. If not refreshing content, this method simply displays the currently rendered view.

```
showView: function(refresh){}
```

Where:

- **refresh** — Specifies whether or not to refresh the view content.

getFooter

Gets the footer object, wrapper `div` for all footer HTML (jQuery Object). This object is a JQuery wrapped `div` with the class of `"footer"`. It is contained within the object returned from `getGadget()` and is displayed underneath the view.

```
getFooter: function(){}
```

Configurable Gadget

A Configurable Gadget is constructed when `view` and `config` parameters are passed in and the current user has permission to edit the gadget's preferences. The gadget contains a view, a footer and a configuration screen.

This gadget inherits all of the methods from Configured Gadget plus the following:

```
return AJS.$.extend(getConfiguredInterface(), {
      showConfig: function(){},    /* Displays the configuration screen */
      getConfig: function(){}      /* Gets the config object, wrapper div for all config html (jQuery
        Object) */
});
```

showConfig

Displays the configuration screen with all fields defined during construction.

```
showConfig: function(){}
```

getConfig

Gets the config form object, wrapper `div` for all config HTML (jQuery Object). It is contained within the object returned from `getGadget()`.

```
getConfig: function(){}
```

RELATED TOPICS

Using the Atlassian Gadgets JavaScript Framework
Writing an Atlassian Gadget
Gadgets and Dashboards Development Hub

Gadget Developers' JavaScript Cookbook

Terminology: On this page and its child pages,

- when we refer to the '**framework**', we mean the Atlassian Gadgets JavaScript Framework.
- when we refer to a '**gadget**', we mean a gadget JavaScript object.

JavaScript Framework is currently in JIRA only
The Atlassian Gadgets JavaScript Framework is currently part of the JIRA project. It will eventually be moved into Atlassian Gadgets. See AG-622. Until that merge is completed, you will only be able to use the framework if you are developing gadgets for JIRA.

Here is a list of all entries in the cookbook, plus the first few lines of content. Click a link to see the full text for each entry.

Adding a Chart to the Issue Navigator

{info:title=Documentation under development}

Adding a Reload Option to your Gadget

This will add a reload icon to the footer of the gadget and an auto-reload option to the config screen:

Adding Something to your Gadget Footer

Use the following code in your view template function:

Adjusting the Gadget Height when the Window is Resized

Enable the `onResizeAdjustHeight` option on the view object:

Making Ajax Calls

The framework wraps the OpenSocial `gadgets.io.makeRequest` method with the standard jQuery interface. You should write code as if you were just using the jQuery interface. For example:

Making your Gadget Reload when Resized

Enable the `onResizeReload` option on the view object:

Restricting Edit Permissions on your Gadget Preferences

Use the framework and include the `views` feature:

Showing the Config Screen on Initial Load Only

# Include the `isConfigured` userpref defaulted to 'false':

Specifying Required Features for the Framework

The framework uses the following features (it is safest just to include all of them):

Specifying Required Resources for the Framework

The framework requires the inclusion of the following resources:

Theming your Gadget Dynamically

It is possible to have the theme dynamically adapt to the space available with some JavaScript like this:

Using Authentication in your Gadget

Specify the appropriate features as 'required' in your gadget XML, include the necessary resources in the CDATA section, and use the `useOauth` parameter when constructing your gadget:

Using Cookies in your Gadget

Use the gadget object API for `createCookie`, `readCookie` and `eraseCookie`:

Using Special UserPrefs for the Framework

There are a couple of `UserPref` values that the framework can use to add functionality to the gadget:

RELATED TOPICS

Using the Atlassian Gadgets JavaScript Framework
Gadgets and Dashboards Documentation

Adding a Chart to the Issue Navigator

This page is part of the Gadget Developers' JavaScript Cookbook.

How do I add my chart to the issue navigator view?

Documentation under development

1. You must place the gadget in both categories: JIRA and Charts

```
<Optional feature="gadget-directory">
      <Param name="categories">
            JIRA
            Charts
      </Param>
</Optional>
```

2. You must include the following 2 UserPrefs:

```
<UserPref name="isPopup" datatype="hidden" default_value="false"/>
<UserPref name="projectOrFilterId" datatype="hidden" />
```

3. Dynamically change the Configuration field for your gadget's Configuration form:

```
config: {
      descriptor: function ()
      {
            var gadget = this;
            var filterField ;
            if (/^jql-/.test(gadget.getPref("projectOrFilterId")) || gadget.getPref("isPopup")
              === "true"){
                  // JQL has been passed in and we should place this in a hidden field
                  filterField =
                  {
                        userpref: "projectOrFilterId",
                        type: "hidden",
                        value: gadgets.util.unescapeString(gadget.getPref("projectOrFilterId"))
                  };
            } else{
                  // Lets display the filter picker
                  filterField = AJS.gadget.fields.filterPicker(gadget, "projectOrFilterId");
            }
            return  {
                  action: "/rest/gadget/1.0/chart/validate",
                  theme : "top-label",
                  fields: [
                        filterField,
                        { other field }
                  ]
            };
      }
}
```

   The popup sets the "projectOrFilterId" UserPref to the value `jql-<JQL-string>`. You don't want to display the filter picker.

4. Your resource should accept the UserPref `projectOrFilterId` and be able to deal with values of both formats - `filter-<filterId>` and `jql-<JQL string`

Refer to the PieChart Gadget for an example of this.

RELATED TOPICS

Gadget Developers' JavaScript Cookbook
Using the Atlassian Gadgets JavaScript Framework

Adding a Reload Option to your Gadget

This page is part of the Gadget Developers' JavaScript Cookbook.

How do I add an automatic reload to my gadget?

This will add a reload icon to the footer of the gadget and an auto-reload option to the config screen:

1. Include the `refresh` userpref defaulted to false:

```
<UserPref name="refresh" datatype="hidden" default_value="false" />
```

2. Enable reloading in the `view` config object:

```
view: {
   enableReload: true,
   template: ...
}
```

RELATED TOPICS

Gadget Developers' JavaScript Cookbook
Using the Atlassian Gadgets JavaScript Framework

Adding Something to your Gadget Footer

This page is part of the Gadget Developers' JavaScript Cookbook.

How do I add something to the footer?

Use the following code in your view template function:

```
view:{
   template: function(args) {
      var gadget = this;
      gadget.getFooter().append(
         AJS.$("<div/>").text("Text for the footer");
      );
   }
}
```

RELATED TOPICS

Gadget Developers' JavaScript Cookbook
Using the Atlassian Gadgets JavaScript Framework

Adjusting the Gadget Height when the Window is Resized

This page is part of the Gadget Developers' JavaScript Cookbook.

How do I make `gadgets.window.adjustHeight()` be called when the window is resized?

Enable the `onResizeAdjustHeight` option on the view object:

```
view: {
   onResizeAdjustHeight: true,
   template: ...
}
```

RELATED TOPICS

Gadget Developers' JavaScript Cookbook
Using the Atlassian Gadgets JavaScript Framework

Making Ajax Calls

This page is part of the Gadget Developers' JavaScript Cookbook.

How do I make Ajax calls?

The framework wraps the OpenSocial `gadgets.io.makeRequest` method with the standard jQuery interface. You should write code as if you were just using the jQuery interface. For example:

```
AJS.$.ajax({
   url: "/rest/gadget/1.0/filtersAndProjects",
   type: "GET",
   data: ({projectsOnly : "true"}),
   dataType: "json",
   success: function(msg){
      alert(msg);
   }
});
```

RELATED TOPICS

Gadget Developers' JavaScript Cookbook
Using the Atlassian Gadgets JavaScript Framework

Making your Gadget Reload when Resized

This page is part of the Gadget Developers' JavaScript Cookbook.

How do I make the gadget reload when it is resized?

Enable the `onResizeReload` option on the view object:

```
view: {
   onResizeReload: true,
   template: ...
}
```

RELATED TOPICS

Gadget Developers' JavaScript Cookbook
Using the Atlassian Gadgets JavaScript Framework

Restricting Edit Permissions on your Gadget Preferences

This page is part of the Gadget Developers' JavaScript Cookbook.

How do I restrict who can edit my gadget preferences?

Use the framework and include the `views` feature:

```
<Require feature="views" />
```

Note: This is an Atlassian-specific extension to the `views` feature.

RELATED TOPICS

Gadget Developers' JavaScript Cookbook
Using the Atlassian Gadgets JavaScript Framework

Showing the Config Screen on Initial Load Only

This page is part of the Gadget Developers' JavaScript Cookbook.

How do I show the config screen on initial load only?

Include the `isConfigured` userpref defaulted to 'false' and include the `nowConfigured` field in your config descriptor. On first save, the userpref will be set to true and from then on, the view screen will be shown on gadget load:

1. Include the `isConfigured` userpref defaulted to 'false':

   ```
   <UserPref name="isConfigured" datatype="hidden" default_value="false" />
   ```

2. Include the following field in your config descriptor:

```
AJS.gadget.fields.nowConfigured()
```

3. Or, if that is not available:

```
{
   userpref: "isConfigured",
   type: "hidden",
   value: "true"
}
```

RELATED TOPICS

Gadget Developers' JavaScript Cookbook
Using the Atlassian Gadgets JavaScript Framework

Specifying Required Features for the Framework

This page is part of the Gadget Developers' JavaScript Cookbook.

What features are required for the framework?

The framework uses the following features (it is safest just to include all of them):

Feature	Where Used
setprefs	To store gadget preferences
views	To detect whether the user can edit the gadget preferences
settitle	To automatically include the instance name in the title
dynamic-height	Config screen, View creation, onResizeAdjustHeight
oauthpopup	Authentication
atlassian.util	To detect whether the gadget is rendered by the same server it fetches data from, allowing retrieval optimizations. **Note:** This feature should be declared using an `<Optional>` element, or else your gadget will not work in non-Atlassian containers.
#oauth	Authentication

RELATED TOPICS

Gadget Developers' JavaScript Cookbook
Using the Atlassian Gadgets JavaScript Framework

Specifying Required Resources for the Framework

This page is part of the Gadget Developers' JavaScript Cookbook.

What resources do I need to specify as required in order to use the framework?

The framework requires the inclusion of the following resources:

```
#requireResource("com.atlassian.jira.gadgets:common")
#includeResources()
```

RELATED TOPICS

Gadget Developers' JavaScript Cookbook
Using the Atlassian Gadgets JavaScript Framework

Theming your Gadget Dynamically

This page is part of the Gadget Developers' JavaScript Cookbook.

How do I dynamically theme my gadget?

It is possible to have the theme dynamically adapt to the space available with some JavaScript like this:

```
action: ...
theme : function () {
   if (gadgets.window.getViewportDimensions().width < 500){
      return "top-label";
   } else{
      return "long-label";
   }
}(),
fields: ...
```

RELATED TOPICS

Gadget Developers' JavaScript Cookbook
Using the Atlassian Gadgets JavaScript Framework

Using Authentication in your Gadget

This page is part of the Gadget Developers' JavaScript Cookbook.

How do I use authentication in my gadget?

Specify the appropriate features as 'required' in your gadget XML, include the necessary resources in the CDATA section, and use the `useOauth` parameter when constructing your gadget:

1. Specify the appropriate features as 'required' in the gadget XML:

```
<Require feature="oauthpopup" />
#oauth
```

2. Include the following resources at the beginning of the CDATA section:

```
#requireResource("com.atlassian.jira.gadgets:common")
#includeResources()
```

3. Construct a gadget passing the URL in the `useOauth` param:

```
var gadget = AJS.Gadget({
   baseUrl: "__ATLASSIAN_BASE_URL__",
   useOauth: "/rest/gadget/1.0/currentUser",
   ...
```

4. Any Ajax call will now go through the authentication steps if required.For example:

```
AJS.$.ajax({
   url: "/rest/gadget/1.0/filtersAndProjects",
   type: "GET",
   data: ({projectsOnly : "true"}),
   dataType: "json",
   success: function(msg) {
      alert(msg);
   }
});
```

RELATED TOPICS

Gadget Developers' JavaScript Cookbook
Using the Atlassian Gadgets JavaScript Framework

Using Cookies in your Gadget

This page is part of the Gadget Developers' JavaScript Cookbook.

How do I use cookies in my gadget?

Use the gadget object API for `createCookie`, `readCookie` and `eraseCookie`:

```
view: {
   template: function(args) {
      var gadget = this;
      gadget.createCookie("columnOrder", "assignee", 30);

      var columnOrder = gadget.readCookie("columnOrder");

      gadget.eraseCookie("columnOrder");
   }
}
```

RELATED TOPICS

Gadget Developers' JavaScript Cookbook
Using the Atlassian Gadgets JavaScript Framework

Using Special UserPrefs for the Framework

This page is part of the Gadget Developers' JavaScript Cookbook.

What are the special userprefs used for?

There are a couple of `UserPref` values that the framework can use to add functionality to the gadget:

```
<UserPref name="isConfigured" datatype="hidden" default_value="false" />
<UserPref name="refresh" datatype="hidden" default_value="false"/>
<UserPref name="isPopup" datatype="hidden" default_value="false"/>
```

Where:

- **isConfigured** — Specifies whether the gadget has been configured. If it has not been configured, the config screen is shown. If it has been configured, the view screen is shown.
- **refresh** — Stores how often the gadget should automatically refresh.
- **isPopup** — Is set by the server to determine whether the gadget is being displayed as a popup window, as opposed to inside a dashboard.

RELATED TOPICS

Gadget Developers' JavaScript Cookbook
Using the Atlassian Gadgets JavaScript Framework

Running your Gadget in the Atlassian Reference Implementation

Documentation under development
Here we'll tell you how to use the gadgets reference implementation.

Sources of information for this document:

- Installing the Atlassian Gadgets Reference Implementation on Tomcat — This internal guide contains some useful information, but we'll come up with an easier way.
- Running your Plugin in the Reference Implementation — This existing guide is useful for people who are writing plugins. We will need to modify it to describe the Gadgets refimpl instead of the Plugins refimpl. Or, even better, we can modify the internal document RefApp Plugin to describe the procedure for Gadgets.
- Plugins Relevant to Gadget Development

None of the above is much use to standalone (non-plugin) gadget authors, so we'll need something more for them.

We should also provide a download of the AG RefImpl webapp.

Managing Caching for your Gadget

Caching is on by default in most applications.

You can control caching as follows:

- Change the global default by setting the system property with key `com.atlassian.gadgets.dashboard.ignoreCache` to `true` or

`false`.
- Change the caching setting on a single request, regardless of the system property setting, by adding the `ignoreCache` parameter to the URL, e.g. `http://jira.example.com/jira/secure/DashboardV2.jspa?ignoreCache=false`.

If caching is on (e.g. `ignoreCache` is false), your gadget specs and locale files will not update right away when they change.

RELATED TOPICS

Writing an Atlassian Gadget
Gadget and Dashboards Development Hub

Gadget Containers

To display your gadget, you will need a dashboard or other container which supports Atlassian gadgets. These come in different flavours:

- Developers can use the Atlassian Gadget Reference Implementation.
- Eventually, you will use an Atlassian application that supports gadgets. At present, there are no Atlassian application production releases that support gadgets. Under current planning, JIRA will be first off the block, closely followed by Confluence.
- Gmail Labs.
- iGoogle.
- Google Apps. See blog post introducing private apps. This means that you can add gadgets to Google Apps from your behind-the-firewall Atlassian application, such as Confluence or JIRA.
- Try adding your gadget to one of the other OpenSocial containers.
- The Google OpenSocial Development Environment provides an Eclipse plugin that lets you develop either OpenSocial JavaScript (gadget) client applications or OpenSocial Java client applications using OpenSocial 0.8's REST/RPC protocols, entirely within the Eclipse development environment. See the introductory blog post.

RELATED TOPICS

Gadgets and Dashboards Documentation

Plugins Relevant to Gadget Development

Documentation under development

This page acts as a central place for listing all plugins that a gadget developer will find useful, assuming that these plugins are not installed by default in the relevant applications.

For now, we will list all the plugins here. Later we may split them into separate pages by category. For example, we must integrate this information with Gadget Version Matrix and Running your Gadget in an Atlassian Application.

The following plugins are used by host applications that provide various AGSLs:

- Gadget Publisher plugin
- Gadget Renderer plugin
- Dashboard plugin
- Directory plugin
- Embedded Gadgets plugin
- Plugin Exchange Manager plugin

The following plugins are used by the Reference Implementation:

- RefImpl Dashboard plugin
- RefImpl Dashboard UI plugin

Other plugins:

- Sample Gadgets plugin (Examples used for testing; may be useful for developers.)
- Atlassian Gadgets Shared Libraries plugin (Internal utility classes used by two or more Atlassian Gadget plugins; may not be useful externally.)
- Atlassian Gadgets Test Framework (Classes used in Atlassian Gadgets for unit and functional tests. Not really intended for outside use, but might be handy for people writing their own tests.)

RELATED TOPICS

Gadgets and Dashboards Development Hub

Examples of Gadgets

Beta Documentation and Support
This documentation is under development and currently applies to a **beta** version of Atlassian Gadgets. Atlassian will provide support for gadgets that are part of supported plugins, when those become available. Please refer to Atlassian supported plugins.

See the sample gadgets module in the AG project. The list of gadgets is in the atlassian-plugin.xml.

Atlassian Gadgets

These gadgets have been developed for use on the dashboards of Atlassian applications. Typically they allow you view and interact with information from an Atlassian application such as JIRA and Confluence.

- Crucible Gadget Plugin — This is a plugin that you will need to install into Crucible 2.0 or later.
- JIRA Issues Gadget — This comes in two varieties:
  - The basic version is designed to run standalone. It fetches anonymously-viewable issues from `jira.atlassian.com`. You can use the Subversion XML spec file URL directly to serve the gadget, or check it out and copy the containing directory to your own web server.
  - The plugin version can be checked out and built with Maven 2 (by running `mvn package`). The resulting JAR file at `target/jira-issues-gadget-plugin-1.0-SNAPSHOT.jar` can be installed into JIRA 4.0 or later.

Other Gadgets that Work on Atlassian Dashboards

These are gadgets that we have tried on our test Dashboards and that seem to work. We do not guarantee them in any way, but they are interesting and provide a good starting point for experimentation.

- Tree Frog — This is the gadget URL that you can add directly to your dashboard.

RELATED TOPICS

Gadgets and Dashboards Development Hub
Gadgets and Dashboards Documentation

Tutorials on Writing a Gadget

Below are some useful tutorials on how to write a gadget:

- Creating a Plugin Gadget for JIRA

RELATED TOPICS

Gadget Development

Gadgets and JIRA Portlets

Documentation under development

This page compares JIRA portlets with gadgets.

On this page:

- What's Changed
- JIRA Portlet-to-Gadget Bridge
- Converting a JIRA Portlet to a Gadget

What's Changed

From a user's perspective...

In JIRA versions prior to 4.0, you could configure your dashboard by adding portlets. In JIRA 4.0 and later, portlets have been converted to industry-standard gadgets.

From a developer's perspective...

In the portlets world, you do everything server-side (on the server where the data is held) and use your portlet class and your Velocity templates to render the UI.

In the gadgets world, the UI is rendered entirely using HTML and JavaScript (on the dashboard server). To get dynamic data into the gadget, you will make Ajax calls using `makeRequest` back to the originating server.

Why a special API for making requests? When we render gadgets, even if the gadget specification is coming from your server out there, it is pulled into the dashboard server and parsed and rendered into HTML on the dashboard server. So your JavaScript can only talk to the server where the dashboard is running. The `makeRequest` call will proxy back (in a process that is called 'phoning home') to the originating server to request data, using one of the following:

- The REST APIs that the application already supports.
- A custom servlet that you build in a plugin for that product.
- A custom REST plugin module. This is the recommended option.

Here is a summary of the difference between gadgets and portlets, from the perspective of a JIRA portlet developer:

- With gadgets you will not have to write any Java, unless you need to create a new REST API.
- All you need to do is write the gadget specification file and deploy it into JIRA. That's it.
- If you need to access internal data for reporting, have customised APIs, etc, you will be able to access those as well.
- The container handles a lot of stuff for you, such as:
  - HTTP caching.
  - Rendering the content to the browser.
- You will write JavaScript to make a request to the remote APIs, to grab the data and populate the content. This replaces the Java code you need to write for a portlet, that populates a Velocity template, context etc.

JIRA Portlet-to-Gadget Bridge

We have built a bridge so that you do not have to convert all the JIRA portlets to gadgets right away. Instead, you can use the bridge to display your existing portlet's output on your JIRA gadgets dashboard.

The bridge will not work for all the portlets. Some portlets contain advanced Ajax functionality that will not be pulled into the gadget properly. But you should be able to make backwards-compatible changes to most portlets that will allow them to work in the bridge as well as in older versions of JIRA.

Note: The bridge is applicable to existing portlets only and is an interim solution. Such legacy portlets can run only on:

- JIRA dashboards, not iGoogle or Confluence or other OpenSocial containers.
- The gadget's local instance of JIRA, i.e. the portlet cannot pull information from another JIRA installation, as a gadget can

> **Test before deploying to JIRA 4 in production**
> Be sure to test your portlets in a JIRA 4 dashboard before deploying to a production instance of JIRA 4.

Converting a JIRA Portlet to a Gadget

We will have more information from this page after the Atlassian Summit. Tim is doing a presentation that we can adapt for this page.

There are three options for converting a portlet to a gadget:

1. Use the legacy bridge described above.
   - Pros: Easy (should just work automatically); can deploy one plugin to older JIRA versions as well as JIRA 4.
   - Cons: Cannot take advantage of gadget features; may not be the fastest; legacy portlet API is deprecated and may be removed in the future; no way to make this kind of gadget work cross-application on the server-side. (The portlets from JIRA can still be *displayed* in other gadget containers, so this is not a big con for that many people.)

2. Convert the portlet to a servlet or webwork plugin module that renders the content pretty much exactly the same way the portlet did. Then write a gadget that uses `gadgets.io.makeRequest` to get those contents, and replaces the body of the gadget with the retrieved contents.
   - Pros: Easier than rewriting entirely as a gadget; can make use of gadget APIs; may be more familiar technology for people used to writing portlets (server-side Java servlets and Velocity templates, versus JavaScript/AJAX).
   - Cons: Not ideal from a performance standpoint; does not take full advantage of dynamic JavaScript UI capabilities; using JavaScript and CSS might be awkward; difficult to make backwards-compatible with older JIRA versions.

3. Rewrite the portlet as a proper gadget, using a REST API to retrieve the data you need from JIRA using `gadgets.io.makeRequest`, and DOM manipulation in JavaScript to handle the UI.
   - Pros: Allows you to take full advantage of modern AJAX UI techniques; may have performance benefits; best way to take advantage of caching.
   - Cons: May require a substantial rewrite; may require learning a lot of new technology; difficult or impossible to make backwards-compatible with older JIRA versions.

RELATED TOPICS

Using Atlassian REST APIs in your Gadget
Writing an Atlassian Gadget
Gadgets and Dashboards Development Hub

Writing a Plugin Upgrade Task for JIRA 4.0 (in the JIRA documentation)

Gadget Version Matrix

This page describes the level of support that each Atlassian application provides for the Atlassian Gadgets framework.

Version Matrix

The matrix below shows the Atlassian applications which provide at least some support for Atlassian gadgets. The applications are listed horizontally across the top and the gadget support levels are listed vertically on the left.

- Version numbers next to a tick ✓ show the **earliest** release of the application which provides the relevant level of gadget support.
- Version numbers in brackets show a future release of the application expected to provide the relevant level of gadget support.

Atlassian Gadget Support Level	Bamboo	Confluence	Crowd	Crucible	FishEye	JIRA
(AGSL-0) Prerequisites	✓ Bamboo 2.3	✓ Confluence 3.0	✓ Crowd 2.0	✓ Crucible 2.0	✓ FishEye 2.0	(JIRA 4.0)
(AGSL-1) Gadget Publisher	✓ Bamboo 2.3			✓ Crucible 2.0	✓ FishEye 2.0	(JIRA 4.0)
(AGSL-2) Embedded Gadgets						(JIRA 4.0)
(AGSL-3) Static Dashboard						(JIRA 4.0)
(AGSL-4) Configurable Dashboard with Reference SPI Implementation						(JIRA 4.0)
(AGSL-5) Configurable Dashboard with Custom SPI Implementation						(JIRA 4.0)

Explanation of the Atlassian Gadget Support Levels

Atlassian Gadget Support Level	Description
(AGSL-0) Prerequisites	AGSL-0 means that a host application has not yet integrated Atlassian Gadgets, but has satisfied all of the prerequisite dependencies required for Atlassian Gadgets integration.
(AGSL-1) Gadget Publisher	AGSL-1 means that a host application supports serving gadget spec files that are bundled into installed plugins for consumption by other Atlassian or non-Atlassian gadget containers.
(AGSL-2) Embedded Gadgets	AGSL-2 means that a host application supports some mechanism for embedding individual gadgets in a page and rendering the gadget contents via the Atlassian Gadgets Renderer and Shindig.
(AGSL-3) Static Dashboard	AGSL-3 means that a host application supports displaying programmatically-generated, read-only dashboards in one or more pages as defined by the host application.
(AGSL-4) Configurable Dashboard with Reference SPI Implementation	AGSL-4 means that a host application supports displaying read/write dashboards in one or more pages as defined by the host application, as well as a directory of known gadgets that can be added to dashboards. Persistence and permission checking is provided by a reference implementation of the Atlassian Gadgets SPI (based on SAL).
(AGSL-5) Configurable Dashboard with Custom SPI Implementation	AGSL-5 means that a host application supports displaying read/write dashboards in one or more pages as defined by the host application, as well as a directory of known gadgets that can be added to dashboards. Persistence and permission checking is provided by a custom, host-specific implementation of the Atlassian Gadgets SPI.

RELATED TOPICS

Gadgets and Dashboards Documentation

Reference Documents

Documentation under development

Below are links to some useful articles and blog posts on gadget development, OpenSocial and related topics.

On this page:

- Google Gadgets
- OpenSocial
- OAuth
- Shindig

Google Gadgets

- Google Code — Home of the official gadget specifications and documentation.
- Gadgets Developer Guide — Reference for gadget authors.
- Google Gadgets Tutorial — From an independent (non-Google) developer.

OpenSocial

- OpenSocial.org — Official home of the OpenSocial Foundation.
- OpenSocial Specs — Official technical specifications.
- OpenSocial - Google Code — Home of Google's documentation on building OpenSocial applications and containers.
- Building a Container
- OpenSocial API Documentation
- OpenSocial Community Wiki — Where new specification proposals are developed
- OpenSocial Directory
- OpenSocial Java Client Library — A Java library for using the OpenSocial v0.8 REST API (not needed for most simple gadgets, only for server-side components).

OAuth

- OAuth
- Google OAuth & Federated Login Research
- Google Online Security Blog
- Validating Signed Requests
- OAuth for gadgets
- OAuth Core 1.0 "Editor's Cut"
- OAuth Service Providers
- OAuth Session Fixation Attack (Shindig wiki)

Shindig

- Shindig
- Shindig Community Wiki
- Shindig Website Staging Wiki
- Shindig Architecture Articles
- Shindig Architectural Overview Nov 2008
- Shindig Architecture: Java Gadget Classes
- Shindig Java internals diagram
- (Answers to) Questions about implementing Shindig
- Shindig Java Style Guide
- Locking Down Shindig
- Shindig Git Clone
- Google Gadgets & Shindig RPC Enlightment

Gadgets and Dashboards FAQ

Gadgets and Dashboards FAQ

Known issues, hints and tips and answers to commonly raised questions about using, configuring and developing Atlassian gadgets and dashboards:

Finding Known Issues and Workarounds

Please refer to the open and recently-resolved issues for the Atlassian Gadgets project. This project includes development of both gadgets and dashboards.

Below is a list of the open issues.

- Before reporting a bug or requesting a new feature, please take a look to see if it has already been reported.
- If you find a likely-looking issue, click the link to find more information and possible workarounds.

JIRA Issues (25 issues)

Type	Key	Summary	Reporter	Status
	AG-805	Possible timing issue remaining in MakeRequestTest	Tim Moore	Open
	AG-311	QA Blitz Test:Gadget spec is not checked until the entire page content is obtained	Veenu Bharara	Open
	AG-1107	dashboard hardcodes content-type charset to UTF-8 which is bad and breaks some browsers	Justus Pendleton	Open
	AG-1103	User can drag and drop gadgets in between tabs in canvas mode	Veenu Bharara	Open
	AG-931	After adding a gadget, exisitng gadget fields overlay the gadget browser on IE6	Andreas Knecht	Open
	AG-1099	ajs-gadgets does not handle OAuth properly when the requested URL doesn't match the base URL	Tim Moore	Open
	AG-976	Intermittent failure in UserPrefsTest	Tim Moore	Open
	AG-1096	ajs-gadgets does not properly translate jQuery dataType values to makeRequest content type values	Tim Moore	Open
	AG-234	Calling setprefs on a list type preference does not update the new dynamic UI	Tim Moore	Open
	AG-1032	500 page containing '%' causes error	Tom Davies	Open
	AG-1005	Improper escaping in dashboard.vm	Tim Moore	Open
	AG-661	Gadget security tokens expire after an hour, breaking AJAX in gadgets that have been left open in a browser	Tim Moore	Open
	AG-935	Sometimes adjustHeight() doesn't work properly in IE. This may be performance related	Andreas Knecht	Open
	AG-966	Gadget dragged to another tab doesn't display immediately in that tab	Tim Moore	Open
	AG-1060	gadget doesn't resize when error message is expanded making it impossible to actually read the error	Justus Pendleton	Open

●	AG-1011	Scrollbar not being displayed in IE8	Hamish Barney	Open
●	AG-1065	Make dashboard javascript operations queue	Jonathan Nolen	Open
●	AG-1041	Add Gadget To Directory form doesn't validate URIs	Andreas Knecht	Open
●	AG-959	GadgetSpecUrlBuilder throwing errors unnecessarily	Brydie McCoy	Open
●	AG-963	Gadget move operation has atomicity problem	John Kodumal	Open
●	AG-954	Cannot re-install plugin (sometimes)	Scott Harwood	Open
●	AG-946	Right-clicking gadget header in Safari adds shadow to the gadget	Shaoting Cai	Open
●	AG-888	IE6: Operations Tools item has no hoverstate	Lachlan Hardy	Open
●	AG-862	IE6: Column spacing is too wide	Lachlan Hardy	Open
●	AG-922	Viewing the gadget preference in canvas mode hides part of the gadget	Ryan Talusan	Open

Usage and Administration FAQ

Here is a list of all entries in the usage and administration FAQ, plus the first few lines of content. Click a link to see the full text for each entry.

Development FAQ

Here is a list of all entries in the development FAQ, plus the first few lines of content. Click a link to see the full text for each entry.

- Hints for Developing in iGoogle Sandbox

Hints for Developing in iGoogle Sandbox

If you decide to play around with developing your own Google Gadgets, you may find these notes useful. They are gathered from the experiences of other developers using iGoogle sandbox for the first time.

Hints:

1. There are two Google gadget developer guides, one for the earlier 'legacy' version of the Google gadget API and one for the new version which supports OpenSocial. Make sure you use the new one.
2. Here is the guide for the iGoogle sandbox, a development environment provided by Google where you can develop gadgets for iGoogle.
3. To sign up for the sandbox, you need a Gmail account. Then go to this URL: http://www.google.com/ig/sandbox. Sign up using your Gmail email address. **But** there is a catch: If you go to the above URL again, you will be signed out of the sandbox! Then you will need to sign up again at the same URL. Just give the same information again, and you will get your sandbox access back with all your gadgets intact.
4. You will know you are in the sandbox when the following words appear near the top left of your iGoogle page: '**Welcome to the iGoogle Developer sandbox**'.
5. Your iGoogle sandbox appears as a new tab on your iGoogle page. After signing up, just examine your iGoogle page to see what has magically appeared on it.
6. The editor in which you create the gadgets is actually a gadget itself. So to use it, you need to add it to your iGoogle page (i.e. the sandbox) in the same way as you would add any other gadget. For some reason, it is not auto-added to your sandbox even though a number of other gadgets do automatically appear there. Your first experience of the editor is just as a text box in the middle of the developer's guide, here. But you don't have to keep going back to that page to edit your gadgets.
7. In your iGoogle sandbox, you will have a list of gadgets displayed within another gadget called '**My Gadgets**'. This is where you add gadgets to your sandbox.
8. For the gadgets you are busy developing, it is a good idea to uncheck the '**Cached**' checkbox in the 'My Gadgets' list, so that you can see the results of your changes immediately.
9. When you create a new gadget and save it in the editor, copy the new URL. It is at the top right of the editor box. Then add the gadget to the 'My Gadgets' list immediately. Otherwise you may have difficulty finding your gadget again.
10. Save your gadget code somewhere else as well as in the gadget editor, e.g. on your own machine using a text editor like Notebook. The

Google gadget editor sometimes does some odd things, especially when you are copying code from one gadget to another.

11. If your gadget does not display the output you are expecting, look for the basic misplaced bracket or whatever, rather than assuming it is some weird gadgety thing that is causing the problem.

RELATED TOPICS

Writing an Atlassian Gadget

- Creating your Gadget XML Specification
- Using Substitution Variables and Directives in your Gadget
- Allowing Configuration Options in your Gadget
- Including Features into your Gadget
- Packaging your Gadget as an Atlassian Plugin
- Internationalising your Gadget
- Using Web Resources in your Gadget
- Using Atlassian REST APIs in your Gadget
- Providing User Authentication for Gadgets
- Using the Atlassian Gadgets JavaScript Framework
- Running your Gadget in the Atlassian Reference Implementation
- Managing Caching for your Gadget

Gadgets and Dashboards Glossary

Here is a list of all entries in the glossary, plus the first few lines of content. Click a link to see the full text for each entry.

- Application Programming Interface or API (Glossary Entry) — An application programming interface or API is an interface defined and implemented by the Atlassian Gadgets framework that is used by host applications to invoke operations provided by Atlassian Gadgets. For example, there are APIs for rendering gadgets and dashboards, loading and saving dashboards, and fetching and parsing gadget XML specifications.
- Atlassian Gadgets (Glossary Entry) — The term 'Atlassian gadgets' has two meanings. Firstly, Atlassian gadgets are similar to Google gadgets and provide additional options allowing interaction with Atlassian applications such as JIRA and Confluence. Secondly, the term 'Atlassian Gadgets' (usually with initial capital letters) means the development framework that allows you to develop your own Atlassian gadget.
- Atlassian Gadgets Support Level or AGSL (Glossary Entry) — An Atlassian Gadgets Support Level (AGSL) is a number (0, 1, 2, etc) indicating the level of support an application provides for Atlassian gadgets. An AGSL is one of several stages of integration between Atlassian gadgets and a host application. See the application support levels.
- Container (Glossary Entry) — A container is an application or web page that embeds and displays gadgets, either on a dashboard or individually on a page. The application may offer a configurable dashboard where the user can add gadgets. Or the application may offer another means of displaying a gadget, such as a macro which embeds the gadget into a wiki page.
- Dashboard (Glossary Entry) — A dashboard is typically the landing page of an application such as JIRA or Confluence, providing an entry point into the different levels of functionality within the application. For applications that support Atlassian gadgets, the dashboard is a container where users can add gadgets and personalise their dashboard display.
- Directive (Glossary Entry) — The Atlassian Gadgets framework allows you to specify #-directives in your gadget specification XML file that will be replaced by generated content at run time. These #-directives are provided by the Atlassian Gadget Publisher plugin. They work for any gadget that is provided as a plugin in an Atlassian application. The #-directives do **not** work for gadgets that are served from an external web server.
- Directory (Glossary Entry) — The gadgets directory displays a list of available gadgets, allowing users to select a gadget for installation onto their dashboard. In the future we will extend the functionality offered by the directory, so that it also allows users to comment on and rate gadgets. Administrators can add external gadgets to the directory, making the gadgets available for users to add to their dashboards.
- Gadget (Glossary Entry) — A gadget is a small object (i.e. a piece of functionality) offering dynamic content that can be displayed in a container. Google gadgets are one type of gadget. Atlassian gadgets are similar, providing additional options allowing interaction with Atlassian applications such as JIRA and Confluence.
- Gadget Publisher Plugin (Glossary Entry) — The Gadget Publisher is a plugin to be installed into an Atlassian application, giving the application the ability to produce gadgets for display in other applications.
- Gadget Renderer Plugin (Glossary Entry) — The Gadget Renderer is a plugin to be installed into an Atlassian application, giving the application the ability to render (display) the contents of gadgets. Host applications rarely interact with this plugin directly. It is mostly a wrapper around Shindig that integrates Shindig into an Atlassian plugin framework container.
- Host Application (Glossary Entry) — The host application (or host) is an application that integrates with Atlassian gadgets by using the application programming interface (API) and provides implementations of the service provider interface (SPI).
- OAuth (Glossary Entry) — OAuth is an open protocol allowing web applications to authorize other applications to access their data and services on the behalf of their users. OAuth is the recommended mechanism for an Atlassian gadget that needs to authenticate a user before displaying information on a web page.
- OpenSocial (Glossary Entry) — The main aim of OpenSocial is to define a common API for social applications across multiple websites. Using standard JavaScript and HTML, developers can create applications that access a social network's friends and feeds. Applications that use the OpenSocial APIs can be embedded within a social network itself or access a site's social data from anywhere on the web. OpenSocial applications are based on Google gadgets technology, and gadgets have been standardised as part of the OpenSocial effort. The Atlassian Gadgets framework is based on version 0.8 of the gadget specification from OpenSocial, but does not currently support other parts of OpenSocial, such as the social data or web service APIs.
- Plugin Exchange Manager (Glossary Entry) — The Plugin Exchange Manager (PEM) is a plugin to be installed in Atlassian applications. (*This plugin has not yet been released and its documentation has not yet been published*.) It provides an interface for plugin management

within an application. The PEM itself is a plugin which is an implementation of the Atlassian REST plugin module type. The PEM also interfaces with the Atlassian Plugin Exchange. In past lives, the PEM has been called the Unified Plugin Manager (UPM) and the REST Plugin Manager (RPM).
- Plugin Framework 2 (Glossary Entry) — The Atlassian Plugin Framework 2 is the latest development framework allowing developers to build plugins for Atlassian applications. A plugin is a bundle of code, resources and configuration files that can be dropped into an Atlassian product to add new functionality or change the behaviour of existing features. Refer to the Atlassian Plugin Framework documentation.
- Reference Implementation (Glossary Entry) — The reference implementation (or refimpl) is an application with no features and no functionality. Instead, it represents the lowest common denominator of all the Atlassian applications. In this way the reference implementation provides a way of codifying the shared framework in all the applications. The reference implementation of Atlassian Gadgets allows gadget authors to test gadgets in a lightweight dashboard container without having to install and launch a full product such as JIRA.
- REST Plugin Manager (Glossary Entry) — RPM is an earlier name for the Plugin Exchange Manager.
- Service Provider Interface or SPI (Glossary Entry) — A service provider interface or SPI is an interface defined by the Atlassian Gadgets framework that is implemented by host applications. An application or container will implement a number of SPIs in order to support Atlassian Gadgets at the various support levels.
- Shared Access Layer or SAL (Glossary Entry) — The Atlassian Shared Access Layer (SAL) provides a consistent, cohesive API to common plugin tasks, regardless of the Atlassian application into which your plugin is deployed. Refer to the SAL documentation.
- Shindig (Glossary Entry) — Apache Shindig is the reference implementation of the OpenSocial API specifications that provides the code to render gadgets, proxy requests, and handle REST and RPC requests.
- Trusted Application Authentication (Glossary Entry) — Trusted application authentication (or 'trusted apps') is an Atlassian-developed mechanism allowing two applications to exchange information on behalf of a logged-in user. Atlassian Gadgets can use trusted application authentication to allow transparent authorization of gadgets running in an Atlassian gadget container application to access data and services from other Atlassian applications that have been configured to trust it. For example, an administrator can configure JIRA and Confluence to communicate in a trusted way, so that Confluence can request information from JIRA on behalf of the currently logged-in user. JIRA will not ask the user to log in again or to supply a password.
- Unified Plugin Manager (Glossary Entry) — UPM is an earlier name for the Plugin Exchange Manager.

RELATED TOPICS

Gadgets and Dashboards Documentation

Application Programming Interface or API (Glossary Entry)

An application programming interface or API is an interface defined and implemented by the Atlassian Gadgets framework that is used by host applications to invoke operations provided by Atlassian Gadgets. For example, there are APIs for rendering gadgets and dashboards, loading and saving dashboards, and fetching and parsing gadget XML specifications.

Atlassian Gadgets (Glossary Entry)

The term 'Atlassian gadgets' has two meanings. Firstly, Atlassian gadgets are similar to Google gadgets and provide additional options allowing interaction with Atlassian applications such as JIRA and Confluence. Secondly, the term 'Atlassian Gadgets' (usually with initial capital letters) means the development framework that allows you to develop your own Atlassian gadget.

Atlassian Gadgets Support Level or AGSL (Glossary Entry)

An Atlassian Gadgets Support Level (AGSL) is a number (0, 1, 2, etc) indicating the level of support an application provides for Atlassian gadgets. An AGSL is one of several stages of integration between Atlassian gadgets and a host application. See the application support levels.

RELATED TOPICS

Gadget Version Matrix

Container (Glossary Entry)

A container is an application or web page that embeds and displays gadgets, either on a dashboard or individually on a page. The application may offer a configurable dashboard where the user can add gadgets. Or the application may offer another means of displaying a gadget, such as a macro which embeds the gadget into a wiki page.

RELATED TOPICS

Gadget Containers

Dashboard (Glossary Entry)

A dashboard is typically the landing page of an application such as JIRA or Confluence, providing an entry point into the different levels of functionality within the application. For applications that support Atlassian gadgets, the dashboard is a container where users can add gadgets and personalise their dashboard display.

RELATED TOPICS

Gadgets and Dashboards Administration Guide
Gadgets and Dashboards User Guide

Directive (Glossary Entry)

The Atlassian Gadgets framework allows you to specify #-directives in your gadget specification XML file that will be replaced by generated content at run time. These #-directives are provided by the Atlassian Gadget Publisher plugin. They work for any gadget that is provided as a plugin in an Atlassian application. The #-directives do **not** work for gadgets that are served from an external web server.

ⓘ #-directives are sometimes also called 'macros' or 'pseudo-macros'.

Directory (Glossary Entry)

The gadgets directory displays a list of available gadgets, allowing users to select a gadget for installation onto their dashboard. In the future we will extend the functionality offered by the directory, so that it also allows users to comment on and rate gadgets. Administrators can add external gadgets to the directory, making the gadgets available for users to add to their dashboards.

Gadget (Glossary Entry)

A gadget is a small object (i.e. a piece of functionality) offering dynamic content that can be displayed in a container. Google gadgets are one type of gadget. Atlassian gadgets are similar, providing additional options allowing interaction with Atlassian applications such as JIRA and Confluence .

Gadget Publisher Plugin (Glossary Entry)

The Gadget Publisher is a plugin to be installed into an Atlassian application, giving the application the ability to produce gadgets for display in other applications.

Gadget Renderer Plugin (Glossary Entry)

The Gadget Renderer is a plugin to be installed into an Atlassian application, giving the application the ability to render (display) the contents of gadgets. Host applications rarely interact with this plugin directly. It is mostly a wrapper around Shindig that integrates Shindig into an Atlassian plugin framework container.

Host Application (Glossary Entry)

The host application (or host) is an application that integrates with Atlassian gadgets by using the application programming interface (API) and provides implementations of the service provider interface (SPI).

OAuth (Glossary Entry)

OAuth is an open protocol allowing web applications to authorize other applications to access their data and services on the behalf of their users. OAuth is the recommended mechanism for an Atlassian gadget that needs to authenticate a user before displaying information on a web page.

RELATED TOPICS

Providing User Authentication for Gadgets

OpenSocial (Glossary Entry)

The main aim of OpenSocial is to define a common API for social applications across multiple websites. Using standard JavaScript and HTML, developers can create applications that access a social network's friends and feeds. Applications that use the OpenSocial APIs can be embedded within a social network itself or access a site's social data from anywhere on the web. OpenSocial applications are based on Google gadgets technology, and gadgets have been standardised as part of the OpenSocial effort. The Atlassian Gadgets framework is based on version 0.8 of the gadget specification from OpenSocial, but does not currently support other parts of OpenSocial, such as the social data or web service APIs.

Plugin Exchange Manager (Glossary Entry)

The Plugin Exchange Manager (PEM) is a plugin to be installed in Atlassian applications. (*This plugin has not yet been released and its documentation has not yet been published*.) It provides an interface for plugin management within an application. The PEM itself is a plugin which is an implementation of the Atlassian REST plugin module type. The PEM also interfaces with the Atlassian Plugin Exchange. In past lives, the PEM has been called the Unified Plugin Manager (UPM) and the REST Plugin Manager (RPM).

Plugin Framework 2 (Glossary Entry)

The Atlassian Plugin Framework 2 is the latest development framework allowing developers to build plugins for Atlassian applications. A plugin is a bundle of code, resources and configuration files that can be dropped into an Atlassian product to add new functionality or change the behaviour of existing features. Refer to the Atlassian Plugin Framework documentation.

Reference Implementation (Glossary Entry)

The reference implementation (or refimpl) is an application with no features and no functionality. Instead, it represents the lowest common denominator of all the Atlassian applications. In this way the reference implementation provides a way of codifying the shared framework in all the applications. The reference implementation of Atlassian Gadgets allows gadget authors to test gadgets in a lightweight dashboard container without having to install and launch a full product such as JIRA.

RELATED TOPICS

Running your Gadget in the Atlassian Reference Implementation

REST Plugin Manager (Glossary Entry)

RPM is an earlier name for the Plugin Exchange Manager.

Service Provider Interface or SPI (Glossary Entry)

A service provider interface or SPI is an interface defined by the Atlassian Gadgets framework that is implemented by host applications. An application or container will implement a number of SPIs in order to support Atlassian Gadgets at the various support levels.

Shared Access Layer or SAL (Glossary Entry)

The Atlassian Shared Access Layer (SAL) provides a consistent, cohesive API to common plugin tasks, regardless of the Atlassian application into which your plugin is deployed. Refer to the SAL documentation.

Shindig (Glossary Entry)

Apache Shindig is the reference implementation of the OpenSocial API specifications that provides the code to render gadgets, proxy requests, and handle REST and RPC requests.

Trusted Application Authentication (Glossary Entry)

Trusted application authentication (or 'trusted apps') is an Atlassian-developed mechanism allowing two applications to exchange information on behalf of a logged-in user. Atlassian Gadgets can use trusted application authentication to allow transparent authorization of gadgets running in an Atlassian gadget container application to access data and services from other Atlassian applications that have been configured to trust it. For example, an administrator can configure JIRA and Confluence to communicate in a trusted way, so that Confluence can request information from JIRA on behalf of the currently logged-in user. JIRA will not ask the user to log in again or to supply a password.

Unified Plugin Manager (Glossary Entry)

UPM is an earlier name for the Plugin Exchange Manager.

Atlassian Gadgets Release Notes

Current released version
Atlassian Gadgets **1.0.3** has now been released — see the Atlassian Gadgets 1.0.3 Release Notes.

Atlassian Gadgets Release Notes

- Atlassian Gadgets 1.0.3 Release Notes
- Atlassian Gadgets 1.0.2 Release Notes
- Atlassian Gadgets 1.0.1 Release Notes
- Atlassian Gadgets 1.0 Release Notes

Atlassian Gadgets 1.0.3 Release Notes

30 September 2009

With pleasure, Atlassian presents **Atlassian Gadgets 1.0.3**. This release includes updated German, French, US English and Japanese translations.

Want to develop an Atlassian gadget?
Take a look at the gadget developer guide. Then see how to use the Atlassian Plugin SDK and wrap your gadget as a plugin.

Comments, Requests and Feedback
We would love your feedback. Please log your requests, bug reports and comments in our issue tracker.

Complete List of Fixes in this Release

JIRA Issues (1 issues)

Key	Summary	Priority	Status
AG-998	Update translations for German, French (maybe Japanese) when available	⇧	Resolved

Atlassian Gadgets 1.0.2 Release Notes

11 September 2009

With pleasure, Atlassian presents **Atlassian Gadgets 1.0.2**. This release fixes a bug that caused a failure when fetching gadget specifications via an HTTP proxy.

Want to develop an Atlassian gadget?
Take a look at the gadget developer guide. Then see how to use the Atlassian Plugin SDK and wrap your gadget as a plugin.

Comments, Requests and Feedback
We would love your feedback. Please log your requests, bug reports and comments in our issue tracker.

Complete List of Fixes in this Release

JIRA Issues (1 issues)

Key	Summary	Priority	Status
AG-1044	HttpClientFetcher doesn't use standard JVM proxyHost & port params	⇧	Resolved

Atlassian Gadgets 1.0.1 Release Notes

11 September 2009

With pleasure, Atlassian presents **Atlassian Gadgets 1.0.1**. This release includes improved logging of SSL errors and fixes a problem with over-aggressive caching of HTTP responses. This is a recommended upgrade for anyone using Atlassian Gadgets 1.0.0.

Want to develop an Atlassian gadget?
Take a look at the gadget developer guide. Then see how to use the Atlassian Plugin SDK and wrap your gadget as a plugin.

Comments, Requests and Feedback
We would love your feedback. Please log your requests, bug reports and comments in our issue tracker.

Complete List of Fixes in this Release

JIRA Issues (3 issues)			
Key	**Summary**	**Priority**	**Status**
AG-1031	HttpClientFetcher is caching all responses (including ones with no cache directives)	⇧	Resolved
AG-1009	Improve logging of SSL exceptions in HttpClientFetcher	⇧	Resolved
AG-666	Caching of OAuth responses	⇧	Resolved

Atlassian Gadgets 1.0 Release Notes

10 September 2009

With pleasure, Atlassian presents **Atlassian Gadgets 1.0**.

This is the first production release of the Atlassian Gadgets and Dashboards framework. JIRA 4.0 (currently in beta) will be the first application to use Atlassian Gadgets. In the new JIRA dashboard, you can add gadgets and personalise the dashboard display by moving the gadgets around, changing their colour, and changing other preferences to suit you.

The great thing about gadgets is that they're easy to write. Even a simple Google Gadget should work on an Atlassian dashboard. For closer integration with an Atlassian application, you can package your gadget as an Atlassian plugin. The Atlassian Gadgets framework uses OpenSocial to help our applications exchange data and components. This opens the door to new types of interoperability between enterprise and consumer systems.

Highlights of this Release:

Want to develop an Atlassian gadget?
Take a look at the gadget developer guide. Then see how to use the Atlassian Plugin SDK and wrap your gadget as a plugin.

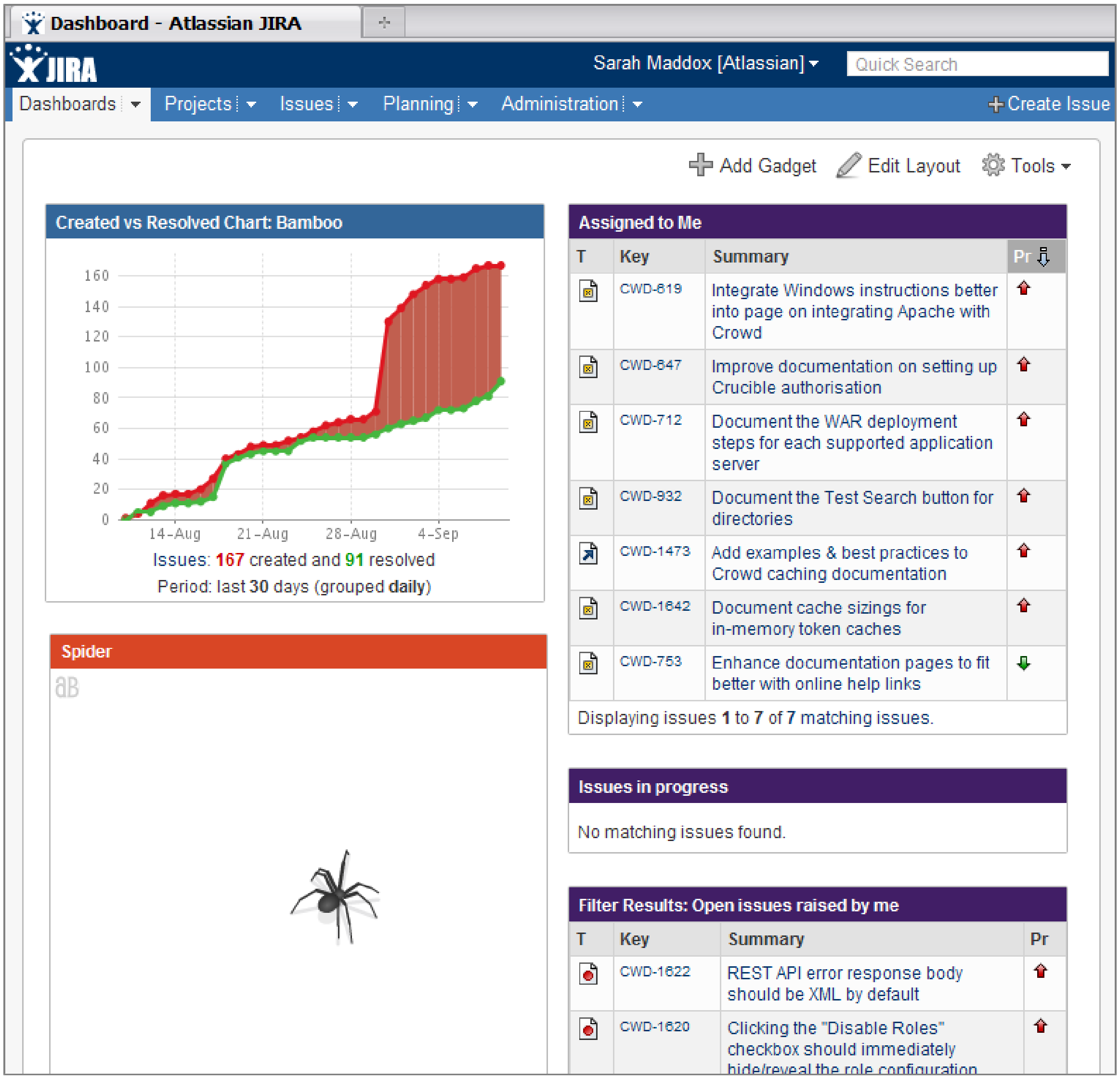
Comments, Requests and Feedback
We would love your feedback. Please log your requests, bug reports and comments in our issue tracker.

Highlights of this Release

1

Drag 'n Drop Gadgets on your Dashboard

JIRA 4.0 (currently in beta) will be the first application to use Atlassian Gadgets. In the new JIRA dashboard, you can add gadgets and personalise the dashboard display by moving the gadgets around, changing their colour, and changing other preferences to suit you. Take a look at our ideas on what gadgets and dashboards do for you and how to use them.



Created vs Resolved Chart: Bamboo

Issues: 167 created and 91 resolved

Period: last 30 days (grouped daily)

Spider

Assigned to Me

T	Key	Summary	Pr
	CWD-619	Integrate Windows instructions better into page on integrating Apache with Crowd	↑
	CWD-647	Improve documentation on setting up Crucible authorisation	↑
	CWD-712	Document the WAR deployment steps for each supported application server	↑
	CWD-932	Document the Test Search button for directories	↑
	CWD-1473	Add examples & best practices to Crowd caching documentation	↑
	CWD-1642	Document cache sizings for in-memory token caches	↑
	CWD-753	Enhance documentation pages to fit better with online help links	↓

Displaying issues **1** to **7** of **7** matching issues.

Issues in progress

No matching issues found.

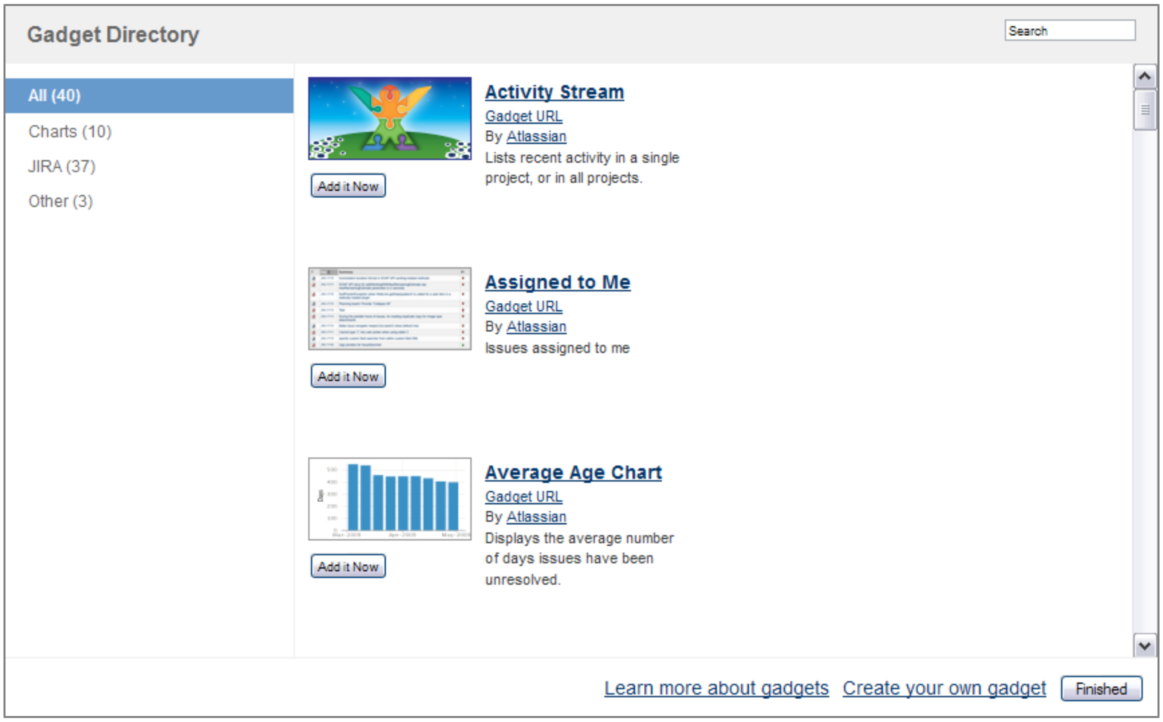
Filter Results: Open issues raised by me

T	Key	Summary	Pr
	CWD-1622	REST API error response body should be XML by default	↑
	CWD-1620	Clicking the "Disable Roles" checkbox should immediately hide/reveal the role configuration	↑

2

Add New Gadgets to your Dashboard

Add gadgets to your dashboard from your gadget directory.



Gadget Directory

Search

- All (40)
- Charts (10)
- JIRA (37)
- Other (3)

Activity Stream
Gadget URL
By Atlassian
Lists recent activity in a single project, or in all projects.
Add it Now

Assigned to Me
Gadget URL
By Atlassian
Issues assigned to me
Add it Now

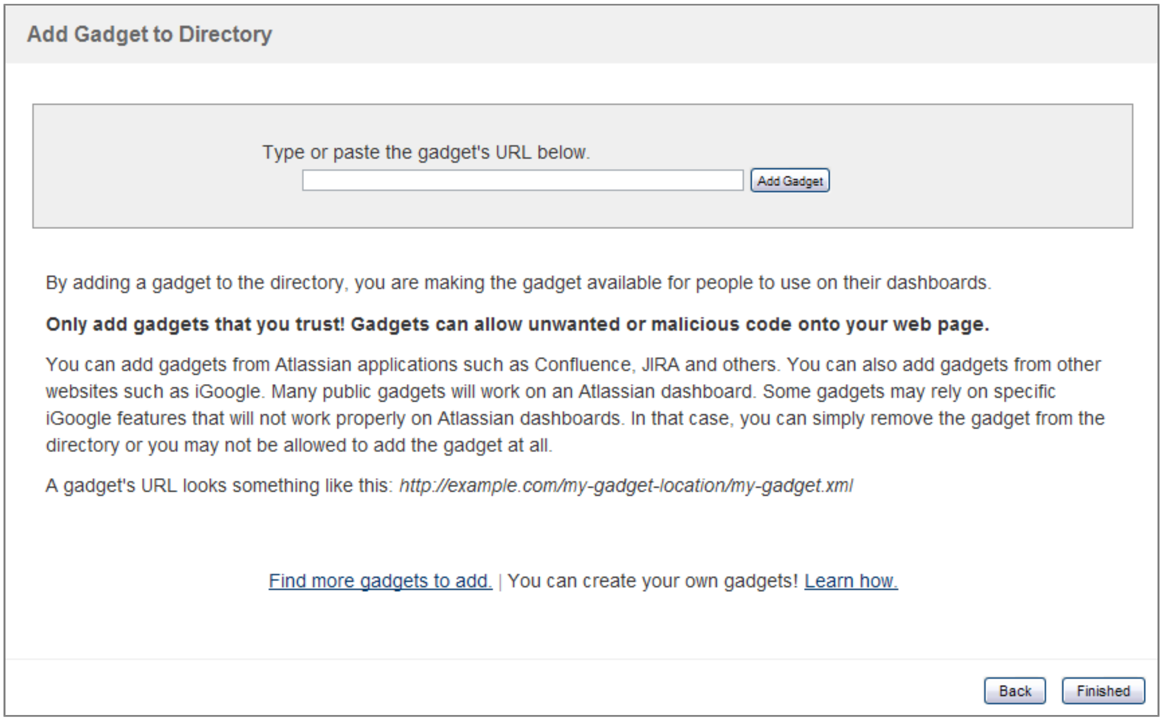
Average Age Chart
Gadget URL
By Atlassian
Displays the average number of days issues have been unresolved.
Add it Now

Learn more about gadgets Create your own gadget Finished

3

Add Gadgets to your Directory

The gadget directory displays all the gadgets that are available to users. These are the gadgets that users can add to their dashboard. If you have administrator privileges, you can add gadgets from Atlassian applications such as Confluence, JIRA and others. You can also add gadgets from other websites such as iGoogle. Many public gadgets will work on an Atlassian dashboard.



Add Gadget to Directory

Type or paste the gadget's URL below.

Add Gadget

By adding a gadget to the directory, you are making the gadget available for people to use on their dashboards.

Only add gadgets that you trust! Gadgets can allow unwanted or malicious code onto your web page.

You can add gadgets from Atlassian applications such as Confluence, JIRA and others. You can also add gadgets from other websites such as iGoogle. Many public gadgets will work on an Atlassian dashboard. Some gadgets may rely on specific iGoogle features that will not work properly on Atlassian dashboards. In that case, you can simply remove the gadget from the directory or you may not be allowed to add the gadget at all.

A gadget's URL looks something like this: *http://example.com/my-gadget-location/my-gadget.xml*

Find more gadgets to add. | You can create your own gadgets! Learn how.

Back Finished

4

Write a Gadget

The great thing about gadgets is that they're easy to write. Take a look at our development hub. Even a simple Google Gadget should work on an Atlassian dashboard. For closer integration with an Atlassian application, you can package your gadget as an Atlassian plugin. See how to use the Atlassian Plugin SDK.

5

Use OpenSocial in your Gadgets

Atlassian is taking OpenSocial behind the firewall and into the enterprise. The Atlassian Gadgets framework uses OpenSocial to help our applications exchange data and components. This opens the door to new types of interoperability between enterprise and consumer systems.