Jive Anywhere – Cartridges SDK 2.1

Overview

Jive Anywhere allows extending its behavior by creating cartridges which can be deployed using the Admin console. For instance, a cartridge can change the search logic for relevant discussions, change UI elements and generate previews for created discussions. Each cartridge is **only** activated for a list of predefined URL patterns and only one cartridge can be activated at a time for a given web page. **Note:** A cartridge is not intended for branding or theming purposes and does not allow that.

You can develop a cartridge for your company needs, such as extending a web app or download a cartridge created by others. Cartridges are developed using industry standards and easy to learn technologies including JavaScript, jQuery, HTML and CSS.

Minimum requirements

Client requirements:

- Jive Anywhere 2.1
- Internet Explorer 8, Chrome 22, Firefox 16 or Safari 5.1

Server requirements:

- Extended APIs 2.1 (v3 recommended)
- Jive 5.x

---

*Cartridges SDK 2.1 introduces some new features as the **minimumRequiredVersion** property and APIs for supporting multiple communities. These changes lead to **breaking changes**. Please read the "What's new" section for more information.*

When using multiple communities, cartridges are only loaded from the server of the default community but apply to all connected communities. The default community can be set using the settings page of Jive Anywhere.

What's new in Cartridges SDK 2.1

The following changes that are **marked in red**, must be applied to existing cartridges otherwise they will fail to run on the new version of Jive Anywhere.

Multiple communities APIs

Previously a cartridge could only access the default community which made it automatically disabled for communities other than the default one.

You can now use **activeConnection**, **defaultConnection** and **connections[]** of the **connectionContexts** property to build cartridges with logic that is enriched from several communities. The **JCommon.asyncRunForArray()** helper method can greatly help with initiating multiple asynchronous requests to multiple servers and process the results at once.
The **createDiscussionViewFrame()** has also been enhanced to allow setting the community of the discussion in view.

- `ModuleContext.clientFacade` ➔ `ModuleContext.connectionContexts.activeConnection.clientFacade`
- `ModuleContext.getConnectionSettings()` ➔ `ModuleContext.connectionContexts.activeConnection.getConnectionSettings()`

More ways to control the UI of the sidebar and the new discussion dialogue

You can now use **reloadSidebar()** and **showSidebar()** to dynamically load the discussion list and re-invoke onGetModuleUi when necessary. ModuleUiInfo properties such as **postButtonText**, **hideDefaultUi**, **hideFilters** and the **newDiscussionCustomCheckboxes** array allow further customization for the look and feel of the sidebar.

The new discussion dialogue can be customized by adding additional checkboxes. Each checkbox must have a name, default value (boolean) and text for its label. When a new discussion is being created the discussion object that is passed to **onCreateDiscussion()** will have a property named **customValues** with a dictionary of the values of these custom checkboxes. You can also display a progress bar while a discussion is being created, in case of large binary file uploads using the **uploadProgressCallback** of onCreateDiscussion().

By using the **refreshDiscussionPreviewOnChange** property of a checkbox you may ask to trigger the **onGetPreviewData()** method whenever a checkbox value changes to re-generate the HTML preview for the discussion using the new values.

- `onGetPreviewData (openGraphMetadata, callback)` ➔ `onGetPreviewData (openGraphMetadata, isFinal, customValues, callback)`
- `onCreatePublicDiscussion (discussion, callback)` ➔ `onCreateDiscussion (discussion, isPublic, callbackCompleted, uploadProgressCallback)`
- `onCreatePrivateDiscussion (discussion, callback)` ➔ `onCreateDiscussion (discussion, isPublic, callbackCompleted, uploadProgressCallback)`

New server APIs

ClientFacade now contains additional methods such as **createPublicConvertedDiscussion()**, **createPrivateConvertedDiscussion()**, **getConvertedDiscussions()** and **getUserByEmail()** which can help you building cartridges for migrating Email based services and portals with your Jive community.

Comprehensive support for binary file handling

You can now download binary files (such as images) into Uint8Array objects using **getBinaryFile()**. You can also upload images to a Jive community server using the **uploadImage()** API and then use the received Id to embed these images into discussions and documents. The generic **postMultipartRequest()** and **sendBinaryRequest()** helper methods of the restHandler can be used with a given Jive service URL to send a multipart request of mixed binary and textual parts while authentication is handled by the API.

Easier to target for multiple versions of Jive Anywhere and Jive server

The ModuleContext and ClientFacade objects now expose a **version** property that contains the version of Jive Anywhere and the server APIs correspondingly. **JCommon.isBinarySupported** property returns true if the browser supports binary APIs.

module.js now must implement a public property named **minimumRequiredVersion** which contains the minimum version of Jive Anywhere required for the cartridge.

- `this.minimumRequiredVersion = 2.1;`

Other improvements

JsRender is now available both in the page context (pagescript.js) and the background context (module.js) using the **$.render** and **$.templates** methods.

Common use cases for the SDK

The SDK can extend Jive Anywhere and also enrich third party websites with Jive content. Here is a list of the most common use cases can be achieved with the SDK:

- Create a custom dynamic preview which is added as a quote when posting a discussion. By default OpenGraph data is extracted from the web page but you can easily replace this logic with custom HTML template bound to any part of the DOM. For instance the included LinkedIn cartridge adds a preview of the visited profile into the created discussion.
- Enrich third party websites with content from Jive. For instance you can write code to show a user popup with details from Jive when hovering a user hyperlink in LinkedIn.
- Define a discussion list search logic for a website. For instance the included LinkedIn cartridge extracts the visited profile name from LinkedIn and performs a search for that name in Jive. The search results are then displayed in the sidebar instead of just searching for discussions contain the URL string.
- Bring content from other servers into the search results. For instance you can query other intranet web servers in the organization using standard HTTP requests and show the results in the Jive Anywhere sidebar.
- Change the logic used for connecting a web page to a group, space or project. By default the connection is made for just the visited URL but in some cases you want to connect the whole website or all pages related to the same context. After the logic was defined (for instance by extracting a title from the DOM), whenever a user is connecting a group, the same group will also appear as connected to all other pages of this context (e.g. other pages having the same title in the DOM).
- Change some UI elements in the Jive Anywhere sidebar when visiting specific web pages. For instance you can change the title of the “Discussions” and “Related” tabs, display custom HTML content on the sidebar and set the default new discussion title to be used when posting a discussion. You can also add custom checkboxes to the new discussion dialogue.
- Embed a discussion view IFRAME (same as the one displayed when clicking a discussion on the sidebar) into your web app or intranet portal by providing a Jive item id and type extracted from the content on your web pages or by using search keywords.
- Add file attachments, embedded images or other binary content to created discussions. When a discussion is created, you can easily issue a download request to download content from the visited web page then upload this data to the created discussion. You can also replace the complete logic by hooking to the onCreateDiscussion() event.

Developer Guide

Components

A cartridge is composed by the following main components:

- module.js is the cartridge's main script file that runs in the background context of Jive Anywhere. It can implement any or none of the pre-defined APIs to set runtime hooks. Cross-origin HTTP requests can be initiated here.
- pagescript.js (optional) is a script file that is injected into the page when the cartridge is active. It can contain public helper functions for reading and manipulating the DOM. Messages can be sent and received between module.js and this script.
- icon.png (optional) is a 16x16 pixels image file that is displayed on the sidebar when the cartridge is active.
- Other content files (optional) such as images, HTML templates, CSS files and so on. These files can be read by using **ModuleContext.getResourceFile()** and referred to by using **ModuleContext.getResourceUrl()** which returns a fully qualified URL that can be embedded into the HTML.
- Some metadata, such as the cartridge display name and a list of activation URL patterns. These are provided in the Admin console together with a zip file that contains all the above content.

Creating your first Jive Anywhere cartridge

To create a basic LinkedIn cartridge, create an empty folder in your local file system with the following files:

module.js

```
// Note: ModuleContext, JCommon and $ are provided by the module manager

// Specify the minimum required version of Jive Anywhere for the module
this.minimumRequiredVersion = 2.1;

// The init function runs only if the cartridge has been activated
this.init = function () {
   // Compile markup as named templates to be used by onGetPreviewData later
   var html = ModuleContext.getResourceFile("LinkedInPageDataTemplate.html");
   $.templates({
      linkedInPageDataTemplate: html
   });
};

// Invoked on URL (community) search
this.onGetUrlSearchResults = function (search, callback) {
   // Extracts the username from the page title
   var userName = extractUserName();

  if (userName != null) {
```

```
      // Performs an exact search based on the extracted username and pass the results
      // to the callback
      var query = "\"" + userName + "\"";
      ModuleContext.connectionContexts.activeConnection.clientFacade.search(query, search.offset,
searchData.limit, search.sortBy, search.isAscending, callback);
   }
};

// Display the username as the title of the Community tab
this.onGetModuleUI = function (callback) {
   var moduleUiInfo = { defaultTabId: 0, tabs: [{ title: extractUserName() }, {}] };
   callback(moduleUiInfo);
};

// Generates preview data when submitting a new discussion instead of the default OpenGraph
this.onGetPreviewData = function (openGraphMetadata, isFinal, customValues, callback) {
   // Extracting page data by invoking a page helper function on pagescript.js
   ModuleContext.runPageScript("getPreviewData", null, function (pageData) {
      // Use the LinkedInPageDataTemplate.html template to generate html markup
      var html = $.render.linkedInPageDataTemplate(pageData);
      // Pass the html back to the module manager
      callback(html);
   });
};

// Helper function to extract the username from the page title
var extractUserName = function () {
    var title = ModuleContext.pageInfo.title;
    return $.trim(title.substring(0, title.indexOf("| LinkedIn")));
};
```

pagescript.js

```
// PageModuleContext, JCommon and $ are provided by the module manager

// Page helper function called by onGetPreviewData to extract fields out of the page
this.getPreviewData = function (params, sendResponse) {
    var profile = new Object();

    profile.fields = [];
    profile.url = document.location.href;
    profile.name = $(".full-name").text();
    profile.locality = $(".locality:eq(0)").text();
    profile.industry = $(".industry:eq(0)").text();

    if ($(".profile-overview").length > 0) {
        // new LinkedIn site
        profile.image = $(".profile-picture img").first().attr("src");
```

```
        profile.title = $("p.title:eq(0)").text();

        $(".profile-overview table").find("th").each(function (index) {
            var title = $(this).text();
            if (title != "Websites") {
                var data = $(this).next("td").find("li");
                var values = [];
                data.each(function () {
                    var value = $(this).text();
                    if (value.substring(0, 1) == ",") {
                        value = value.substring(1);
                    }
                    values.push(value);
                });
                if (data.length > 0) {
                    profile.fields.push({ name: title, values: values });
                }
            }
        });
    }

    if (!profile.image) {
        // use default profile image if image not defined
        profile.image =
"http://static02.linkedin.com/scds/common/u/img/icon/icon_no_photo_40x40.png";
    }

    sendResponse(profile);
};
```

LinkedInPageDataTemplate.html

```
<div style="margin-top: 5px; background-color: #fbfbfb; border: 1px solid #D6D6D6;">
    <table style="border: 0px solid #CCC;">
        <tr>
            <td>
                <a href="{{>url}}" target="_blank">
                    <img src="{{:image}}" alt="Profile image" style="border: 1px solid #D1D1D1;" />
                </a>
            </td>
            <td>
                <b><a href="{{>url}}" target="_blank">{{>name}}</a></b><br />
                  {{>title}}<br />
                  {{>locality}} | {{>industry}}
            </td>
        </tr>
      {{for fields}}
        <tr>
            <td>{{>name}}</td>
            <td>
                {{for values}}
                   <div>{{>#data}}</div>
```

```
                {{/for}}
            </td>
        </tr>
    {{/for}}
</table>
</div>
```

Note: Any text editor can be used such as Notepad or Visual Studio for syntax highlighting.

Download the image file located at https://developer.linkedin.com/sites/default/files/LinkedIn_Logo16px.png and save it as **icon.png** into the same folder.

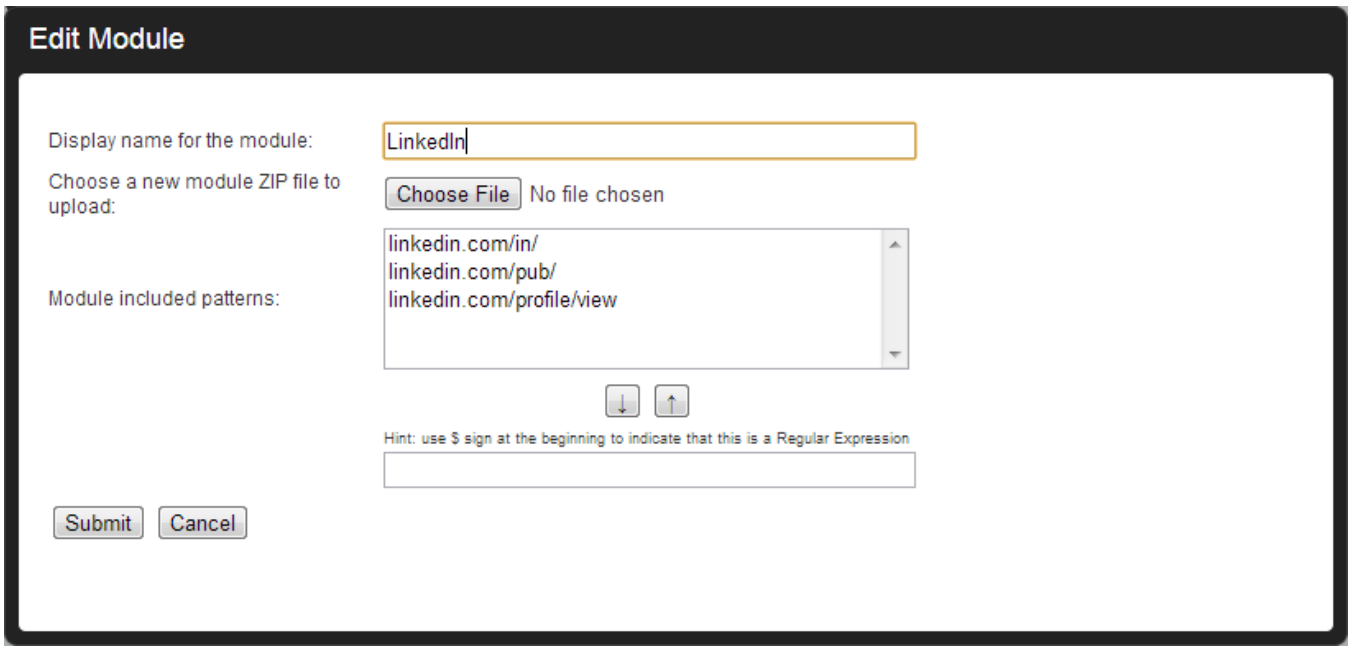
Pack the content of the folder into a zip file named **LinkedInModule.zip** (any name is okay).

Head to the Admin console of your Jive SBS (http://*your-jive-address*/admin), select **System -> Settings -> Jive Anywhere** at the menus and click the **Modules** tab. You must have an account with administrator privileges to enter this page.

Click **Upload new module**. In the **display name** field type LinkedIn. Click "Choose file" and select the created LinkedInModule.zip file.

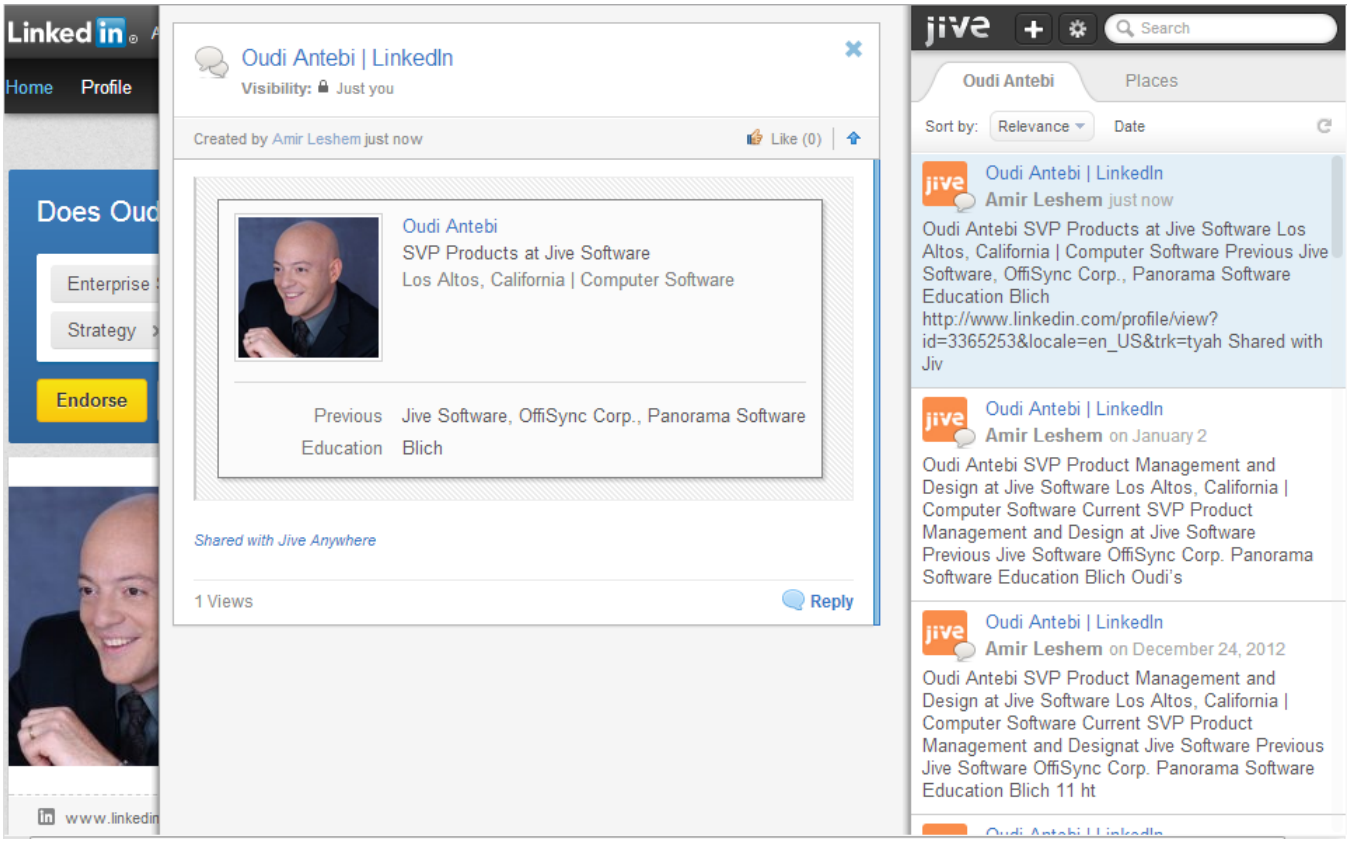
In the **included patterns** field type *linkedin.com/profile/view* and click the **up** button to move it into the patterns list. Repeat this step with *linkedin.com/pub/* and *linkedin.com/in/*. The created cartridge will be activated only for URLs matching these patterns.

When finished click **Submit** and make sure you receive no errors.



Restart your browser and head to a profile page on LinkedIn. You should now see the LinkedIn icon appears on the Jive Anywhere sidebar. Open the sidebar, the profile name should appear as the title of the Community tab.

Create a new discussion, choose any participant (or location) and click **Post**. Click the created discussion and you should see a preview of the LinkedIn profile embedded into its body.



API Reference

Script files

module.js

This is the cartridge's main script file that runs in the background context of Jive Anywhere. It can implement any or none of the following public callbacks to set runtime hooks. Unimplemented callbacks will provide the default behavior of Jive Anywhere.

The script can use the provided **ModuleContext** and **JCommon** helper APIs. In addition, **jQuery** and **JsRender** template engine are available (using the **$** keyword) out of the box. The script can freely use cross-origin Ajax calls using the standard XmlHttp object or jQuery. Jive specific services should be called using the provided **ModuleContext.connectionContexts.activeConnection.clientFacade** which automatically handles authentication and errors.

- **minimumRequiredVersion** **new in 2.1**
  A public property that must specify the minimum version of Jive Anywhere required for the module. If the property doesn't exist or has a higher value then the module won't be activate.
- **init ()**
  Invoked after the cartridge has been activated and pagescript.js has been injected into the page. This method is invoked for each "full" navigation but not for "in-page" navigations.
- **onGetNotificationUI (isSiteInWhitelist, callback)** **new in 2.1**
  Invoked after init() but before the sidebar is opened. Use the callback to return a *NotificationUiInfo* object, which contains UI customizations for the notification button. The isSiteInWhitelist boolean parameter is true if the URL was defined in the admin console under the "Notifications Whitelist" section. By convention, you should respect the user's privacy by not querying data from servers (to decide if the notification button should be highlighted) unless the site is in the whitelist. If not implemented, the default **onGetUrlSearchResults()** method will be invoked to query results.
- **onGetModuleUI (callback)**
  Invoked after the sidebar is opened for the first time after each "in-page" or "full" navigation or after switching from another community.
  Use the callback to return a *ModuleUiInfo* object, which contains UI customizations for the sidebar.
- **onMessage (command, data, sendResponse)**
  Invoked when **PageModuleContext.sendMessage** is called by pagescript.js. It should be used to transfer JSON objects and messages between the page and the background contexts of the cartridge. A message receives a command string, a JSON data object and a sendResponse callback which can be used to send back a JSON object.
- **onGetUrlSearchResults (searchData, callback)**
  Invoked when a URL (community) search is in progress. It can be overridden to replace the default search logic and display different search results.
- **onGetContainerSearchResults (searchData, callback)**
  Invoked when a connected place (group/space/project) search is in progress. It can be overridden to replace the default search logic and display different search results.

- **onGetNormalizedSearchUrl (url)**
  Invoked after the sidebar is opened. Returns the normalized URL of a visited page based on the original one. This function is useful for cleaning the actual URL being searched and attached to a new discussion or to generate a virtual URL for single-page apps.
- **onGetRelatedContainerTag ()**
  Invoked when connecting, disconnecting or getting the connected place(s) for the page. Returns the key tag for connecting a place to a URL. Identical tag keys for different URLs will handle the same connected place(s). By returning an empty string you can disable results for the Related discussions tab.
- **onGetPreviewData (openGraphMetadata, isFinal, customValues, callback)** **new in 2.1**
  Invoked when a new discussion is created. Returns the HTML preview for the discussion. If not provided, a preview based on OpenGraph will be used. You can use the openGraphMetadata dictionary object for creating your own html or invoke a page context helper to extract additional data. The isFinal parameter is true if a discussion is being created, otherwise it is just used to preview the discussion. The customValues parameter contains the dictionary of checkbox values defined in ModuleUiInfo.
- **onCreateDiscussion (discussion, isPublic, callbackCompleted, uploadProgressCallback)** **new in 2.1**
  Invoked when a new discussion should be created. **Note**: the customValues property of discussion will contain custom checkbox values defined in ModuleUiInfo. You can invoke the uploadProgressCallback(value) method to display a progress bar while a discussion is being created, and set the completed progress (float value between 0 to 1).

pagescript.js

This script file is injected into the page when the module is active, prior to calling the cartridge's **init()**. It can contain public helper functions for reading and manipulating the DOM. These functions are defined by the *(object params, func sendResponse)* signature and can be invoked by module.js using **ModuleContext.runPageScript()**. Each function received a JSON object from the background context and can send back a JSON object response using the sendResponse(object) callback.

The script can use the provided **PageModuleContext** and **JCommon** helper APIs. In addition, **jQuery** and **JsRender** template engine are available (using the **$** keyword) out of the box.

Container types and Helper classes

ModuleContext

Used by module.js to provide a set of helper APIs.

- **version**
  Returns the installed version of Jive Anywhere.
- ***PageInfo* pageInfo**
  Returns information about the current visited web page.
- **connectionContexts** **new in 2.1**
  - *ConnectionContext* activeConnection
  - *ConnectionContext* defaultConnection

    - *ConnectionContext[]* connections[*serverUrl or index*]
- **runPageScript (*string* funcName, *object* params, callback)**
  Runs the given function on the page context of a previously injected pagescript.js, passing a JSON object and a callback. When sendResponse(result) is called by the targeted function, the provided callback is invoked with the result as a parameter.
- **getResourceFile (url)**
  Gets the textual content of a file from the cartridge package. The provided URL should be relative to the cartridge's root folder. Content files such as scripts, HTML templates and CSS can be fetched using this way and then be appended into the DOM (or eval'ed) into the current scope.
- **getResourceUrl (url)**
  Gets a fully qualified URL of a resource file from the cartridge package. The provided URL should be relative to the cartridge's root folder. The returned URL can then be used as a source for image elements or other embedded content.
- **getInternalResourceUrl (url)** **new in 2.1**
  Gets a fully qualified URL of a static internal resource included with Jive Anywhere.
- **newTab (url, openInBackground)**
  Opens a new tab with the specified URL. If openInBackground is true, the tab will not be activated automatically.
- **reloadSidebar ()** **new in 2.1**
  Reloads the list of discussions, connected places and UI and reinvoke the relevant API calls such as onGetUrlSearchResults(). Calling this method also clears the content of the new discussion dialogue.
- **updateNewDiscussionCheckboxes (Array[] checkboxArr)** **new in 2.1**
  Updates the collection of custom checkboxes in the new discussion dialogue and the discussion preview without reloading the sidebar. The value should be an array of {name, checked, text, refreshDiscussionPreviewOnChange}.

PageModuleContext

Used by pagescript.js to provide a set of helper APIs.

- **sendMessage (*string* command, *object* data, callback)**
  Sends a command string and a JSON data object to the cartridge's onMessage function. When sendResponse(result) is called by the onMessage function, callback is invoked with the result as a parameter.
- **createDiscussionViewFrame (frameId, typeId, itemId, serverUrl)** **new in 2.1**
  Create a new IFRAME with the loaded Jive item specified by the typeId and itemId. You may specify the serverUrl of the community for the item or send null to select the default community. The IFRAME can then be appended into the DOM. Its height is managed automatically by Jive Anywhere so you'll never see scrollbars inside the frame. You should also implement the following public callback functions on pagescript.js:
    - **discussionFrameLoaded (params)**
      is invoked after the discussion frame has been loaded
      params { frameId, discussion, serverUrl, typeId, itemId }

        - **discussionFrameCloseButtonClicked (params)**
          is invoked when the close button is clicked, so you can hide or remove the IFRAME from the DOM.
          params { frameId }
        - **discussionFrameScrollTop (params)**
          is invoked when the scrollbar location should be changed due to scrolling to a comment.
          params { frameId, top }
- **showSidebar(shouldOpenNewDiscussion, serverUrl)** **new in 2.1**
  This function allows you to open the sidebar when necessary and optionally open the new discussion dialogue (for instance when a custom button is clicked). You may specify the serverUrl of the community to select or send null to select the default community.

JFile **new in 2.1**

A binary file type which contains the following properties.

- **data**
- **contentType**
- **name**
- **toDataUrl()**
  Converts the content to a base64 data URL.

JCommon

A single-ton helper class which provides many common functions. It is available for both module.js and pagescript.js contexts.

Compatibility flags

- **isIE**
- **isOldIE**
  Returns true if running on IE8 or lower.
- **isBinarySupported** **new in 2.1**
  Returns true if the client support binary file handling (IE10, Chrome and Firefox are currently supported)
- **isQuirksMode**
  Returns true if the current page is using quirksmode and running IE<10

DOM Helpers

- **trimHtmlWhitespaces(html)**
- **htmlEncode(value)**
- **htmlDecode(value)**
- **getNodeSource(htmlElement, shoudConvertRelativeUrls)** **new in 2.1**
  Gets an html from an element that can be sent to the server as the HTML body of a discussion/document/comment object. It should always be used instead of a plain innerHTML usage as it correctly standardize the HTML across all browsers and trims unnecessary white-spaces. If shouldConvertRelativeUrls is true, all relative URLs of src and href attributes will be converted to fully qualified URLs using the document's base address.

- **getNodeInnerSource(htmlElement, shoudConvertRelativeUrls)** **new in 2.1**
- **addCssFile(url)**
  Adds <style> element to the current document header with a link to external file. Note: Not recommended, might be blocked by the browser if the visited website is using HTTPS protocol but your Jive server is using HTTP.
- **addCssText(styles)** **new in 2.1**
  Adds <style> element to the current document header with the textual content of the given parameter. The CSS may contain background images with base64 encoded data URLs.

String Helpers

- **shortenString(str, maxLength)**
- **getCookie(name, cookieStr?)**
- **getQueryStringParameterByName(name, queryStr?)**
- **getNonRelativeUrl(url)**
  Returns a fully qualified URL that can then be accessed from other pages. This method must be called on the frame of the origin URL.
- **asPrettyDate(date)**
- **asPrettyFileSize(fileSizeInBytes)** **new in 2.1**

Asynchronous Programming Helpers

- **asyncRunMultipleCalls(numOfCalls, codeFunction(index, submitResultCallback), onCompleteCallback(Array[] results))** **new in 2.1**
- **asyncRunForArray(Array[] arr, codeFunction(item, submitResultCallback), onCompleteCallback(Array[] results))** **new in 2.1**
  Runs multiple async operations on each item from the given array by passing it to the codeFunction callback. The submitResultCallback must be called for each result after processing each item. After all operations completed the onCompleteCallback method is invoked with all the collected results ordered by their corresponding operation index. This method is very helpful for processing requests from multiple communities by passing ModuleContext.connectionContexts.connections as the array parameter.

Binary Helpers **new in 2.1**

- **arrayBufferToBase64(Uint8Array arrayBuffer)**
- **base64ToArrayBuffer(base64str)** **IE8/9 not supported**
- **joinArrayBuffers(Uint8Array ab1, Uint8Array ab2)** **IE8/9 not supported**
- **strToArrayBuffer(str)** **IE8/9 not supported**
  Creates a UTF-8 encoded Uint8Array from string.
- **createBinaryFilePart(jFile)** **IE8/9 not supported**
  Creates a Uint8Array object which can then be sent as a binary file content of a multipart request using clientFacade.restHandler.sendBinaryRequest().
- **createContentPart(body, isJson)** **IE8/9 not supported**
  Creates a Uint8Array object which can then be sent as the main content of a multipart request using clientFacade.restHandler.sendBinaryRequest().

- **getBinaryFile(url, callbackCompleted, callbackDownloadProgress)** **IE8/9 partially supported**
  Downloads a binary file from any website such as an image into a Uint8Array buffer which can then be uploaded to a Jive server or converted to a base64 URL. When calling from the pagescript.js context cross domain requests are not allowed, however, when calling from the background context cross domain requests are allowed but fully qualified URLs should be used. Cookies are automatically added to the request. Returns: jFile.
  Note: For IE8/9, jFile.data contains a fake array instead of a Uint8Array object, which is much slower and not recommended for big files. It can then by converted to a base64 string. For IE8 this method is only available in the background context.

Jive-specific Helpers

- **convertServerDate(serverDate)**
- **getJiveObjectTypeById(id)**
- **decodeEmail(emailAddress)** **new in 2.1**
  Parses a jive reply-to email address which contains { prefix, userId, itemType, itemId }

PageInfo

Used by module.js to provide details about the current visited web page.

- **url**
- **title**
- **domain**
- **bool isFrameSet**

NotificationUiInfo

Used by module.js to get an object that defined custom UI changes to the notification button. All properties are optional, if a value is not supplied the default one will be used.

- **autoHide**
  If true, automatically hides the notification button when the sidebar is closed.
- **highlight**
  If true, highlights the notification button. This property should be used to let the users there is some interesting content on the sidebar, such as discussions in progress.

ModuleUiInfo

Used by module.js to get an object that defines custom UI changes to the sidebar. All properties are optional, if a value is not supplied the default one will be used.

- **defaultTabId**
  The default tab to display when opening the sidebar in activated page (0 = discussions, 1 = related).
- **defaultTitle**
  The default title for the new discussion form
- **postButtonText** **new in 2.1**
  Set the label of the post button in the new discussion form

- **hideSnapshots**
  If true, the “add a new snapshot” button will be hidden
- **Array[] newDiscussionCustomCheckboxes** **new in 2.1**
  Create a list of additional checkboxes to display in the new discussion dialogue. The value should be an array of {name, checked, text, refreshDiscussionPreviewOnChange}.
- **Array[] tabs**
  This array (if defined) must contain 2 objects, the first one is for the “Discussions” tab and the second for the “Related” tab
  - **html**
    (optional) HTML of custom container displayed at the top of the search results when the module is active. It’s recommended to use the **moduleTitle** CSS class for decorating your module’s HTML (e.g. …) in order to get consistent look.
  - **title**
    (optional) The search title to display replacing the generic this page term, “Viewing discussion about **this page**”.
  - **hideDefaultUi** **new in 2.1**
    If true, the default UI displayed above the discussion list will be hidden
  - **hideFilters** **new in 2.1**
  - If true, the filters and sort controls above the discussion list will be hidden

ConnectionContext **new in 2.1**

Used to get metadata or communicate an instance of a connected community using its credentials.

- **index**
- **serverUrl**
- ***ClientFacade*** **clientFacade**
  Returns a communication proxy to the Jive server. If the object is not the default or active connection, you must first initializing it by calling getSettings() before accessing the clientFacade property and wait for the callback to be invoked. Later you can access it directly.
- **getSettings(callback)**
  Initialize the clientFacade property of the object if not yet initialized and receive the cached system-info properties returned by the server.
- **getConnectionSettings()**
  Gets current connection settings including serverUrl, embeddedAuthToken, favIconUrl and instanceName.

ClientFacade

This class can be used to access Jive services REST APIs. Authentication and errors are handled automatically.

- **version**
  Returns the version of the façade matching the installed version of EAPIs

- **searchUrl (url, offset, limit, sortBy, isAscending, callback)**
  Use to search for items based on URLs. Call on onGetUrlSearchResults and onGetGroupSearchResults to create custom queries and manipulating results before sending them to the callback function of the event.
- **search (query, offset, limit, sortBy, isAscending, callback)**
  Use to search for items based on text content. Call on onGetUrlSearchResults and onGetGroupSearchResults to create custom queries and manipulating results before sending them to the callback function of the event.
- **getContentInContainer (containerTypeId, containerId, offset, limit, filterType, callback)**
  Use to retrieve content in group, space or project.
- **getConnectedUser (callback)**
  Returns the current connected user.
- **getUserById (id, callback)**
  Returns a user matches the id.
- **getUserByEmail (emailAddress, callback)** **new in 2.1**
  Returns a user matches the email address.
- **searchMentions (query, callback)**
  Returns results for @mention query.
- **searchSpotlight (query, offset, limit, callback)**
  Returns lists of places, people and content for the given query.
- **searchUsers (query, limit, callback)**
  Returns list of people for the given query.
- **searchContainers (query, limit, callback)**
  Returns list of places for the given query.
- **createCommentForItem (itemTypeId, itemId, htmlContent, callback)**
  Creates a new comment or reply to the given item.
- **createPublicDiscussion (subject, htmlContent, isQuestion, containerTypeId, containerId, callback)**
  Creates a new public discussion at the specified container (place).
- **createPrivateDiscussion (subject, htmlContent, isQuestion, userIds, callback)**
  Creates a new private discussion between you and the given users.
- **createPublicConvertedDiscussion (messageData, containerType, containerId, emailMessageId, notifyExternalRecipients, JFile[] attachments, callbackCompleted, callbackUploadProgress)** **new in 2.1**
  Create a public discussion that has been converted from email messages. **Note**: attachments is not supported on IE8/9 and require EAPIs v3.
  messageData is: { question, Array[] notificationRecipients, Array[] messages{body, subject, sender, senderName, Array[] attachmentUris} }
- **createPrivateConvertedDiscussion (messageData, userIds, emailMessageId, notifyExternalRecipients, JFile[] attachments, callbackCompleted, callbackUploadProgress)** **new in 2.1**
  Create a private discussion that has been converted from email messages. messageData described in the former method.

- **getConvertedDiscussions (emailMessageId, callback)** **new in 2.1**
  Returns a list of discussion URLs (EAPIs v2.1) or discussion objects (EAPIs v3) for discussions that has been previously created using the createConvertedDiscussion API using the same emailMessageId.
- **getConnectedContainers (tag, callback)**
  Returns a list of connected places to a tag.
- **attachContainerToTag (containerTypeId, containerId, tag, callback)**
  Attaches a place into a new or existing tag.
- **detachContainerFromTag (containerTypeId, containerId, tag, callback)**
  Detaches a place from a tag.
- **createNewConnectedGroup (name, tag, description, groupType, callback)**
  Creates a new group attached to the given tag.
- **sendGroupInvites (groupId, groupName, userEmails, callback)**
  Sends group invites to the given emails.
- **getDefaultDiscussionFolders (callback)**
  Returns the recent containers used to create a discussion.
- **getDiscussion (typeId, itemId, increaseViewCount, callback)**
  Returns a discussion and its comments.
- **like (typeId, itemId, callback)**
  Sets the given item as liked.
- **unlike (typeId, itemId, callback)**
  Sets the given item as unliked.
- **markCorrect (messageId, boolToggleState, callback)**
  Marks the given item as correct or incorrect according to the toggle state.
- **markHelpful (messageId, boolToggleState, callback)**
  Marks the given item as helpful or un-helpful according to the toggle state.
- **uploadImage (JFile imageFile, callbackCompleted, callbackUploadProgress)** **new in 2.1, IE8/9 not supported, requires EAPIs v3**
  Uploads an image from JFile object (Uint8Array) to the Jive storage and returns an ID which can then be referenced to embed the image into discussions/documents/comments.
- **uploadAttachmentsToContent (contentId, JFile[] attachments, callbackCompleted, callbackUploadProgress)** **new in 2.1, IE8/9 not supported, requires EAPIs v3**
- **uploadAttachmentsToMessage (messageId, JFile[] attachments, callbackCompleted, callbackUploadProgress)** **new in 2.1, IE8/9 not supported, requires EAPIs v3**
- **restHandler** **new in 2.1**
  - **sendWebRequest (requestType, actionUrl, *object* data, callbackCompleted, bool isJson?, *bool* sendWithoutCredentials?)**
    Sends an Ajax request using the underlying communication layer. requestType can be either “GET”, “POST”, “PUT”, “HEAD” or “DELETE”. actionUrl is a URL for the service related to the server address. Error handling and authentication is managed by Jive Anywhere. If you wish to send an object as JSON, make sure to set isJson as true, otherwise it will be sent as a query string.
  - **sendBinaryRequest (requestType, actionUrl, Uint8Array[] arrayBuffers, callbackCompleted, callbackUploadProgress)** **IE8/9 not supported**

Sends a multipart request of binary parts of type Uint8Array. Each binary part represent a file and it must contain headers which can be added using the JCommon.createBinaryFilePart() or JCommon.createContentPart() methods. Note: In order to process just one binary part you have to place it inside a single-item array using "[*object*]".

- **postMultipartRequest (actionUrl, object data?, JFile[] files, bool isJson, callbackCompleted, callbackUploadProgress)**
  A shortcut for sending a POST request that optionally has one textual data part and multiple binary parts. In case binary mode is not available for the browser, or no files attached, a fallback to sendWebRequest() will be initiated.

JsRender Template engine Helpers

- **~getResourceUrl(relativeModuleUrl)**
  Returns a URL which can be embedded in HTML as a resource linked to a file in the cartridge package. For instance `<img src='{{:~getResourceUrl("icon.png")}}'/>`
- **~getInternalResourceUrl(relativeInternalUrl)** **new in 2.1**
  Returns a URL following an internal resource of the Jive Anywhere extension. This can be used to get default images such as the default profile image bundled with the extension.

Debugging your cartridge

For code changes to take effect, you have to restart the browser or reload the extension (Chrome/Safari) each time you upload the cartridge via the Admin console.

The easiest way to debug is using Chrome or Internet Explorer. In both cases you can add the "*debugger;*" statement in order to break into debugger when approaching the statement. You can also use *console.log(object)* to print output into the debugger console.

Debugging Internet Explorer

Press F12 to open the developer tools, click the Script tab and "start debugging".

Debugging Chrome

1. To debug pagescript.js - press F12 to open the developer tools, then click the Sources tab.
2. To debug module.js or other background context – go to Extensions in the Settings page of Chrome, make sure "Developer mode" is checked, and click the URL near "Inspect views", then click the Sources tab.

Debugging Safari

Open the preferences dialogue, click Advanced and make sure “Show Develop menu in menu bar” is checked.

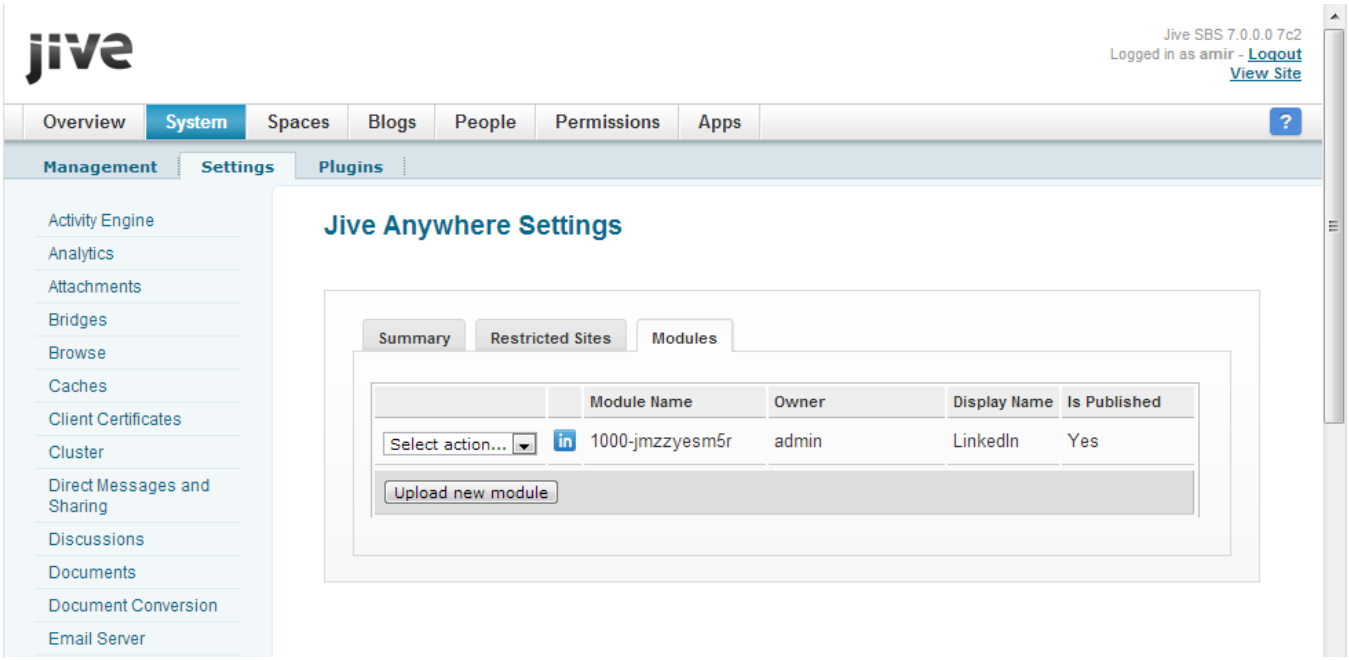
1. To debug pagescript.js - click the Develop menu and select “Start debugging JavaScript”.
2. To debug module.js or other background context – click the Develop menu and select “Show Extension Builder”, then click the “Inspect Global Page” button in Jive Anywhere.

Debugging Firefox

Due to the lack of ability to debug background extension context it is not recommended to use Firefox for debugging your cartridges. If no other option is available try using JavaScript alerts.

Manage cartridges via Admin console

Jive Anywhere cartridges are managed in the Admin console of Jive SBS. Go to **http://*your-jive-address*/admin** to enter the Admin console, then select **System -> Settings -> Jive Anywhere** at the menus and click the **Modules** tab. You must have an account with administrator privileges to enter this page.



Upload a new module

To upload a new cartridge, click **Upload new module** in the Modules tab.

In the displayed dialog enter the following fields:

- Type a display name for the cartridge.
- Choose the zip file of the new cartridge. The zip file must contain module.js and the optional files pagescript.js and icon.png as described in the Developer Guide section.
- Apply the cartridge included patterns. The cartridge will be activated only if one or more of the patterns matches the URL of the visited page. For instance *google.com* will match any web page in the

google.com domain. You can also use Regular Expression patterns by prefix the pattern with a **$** sign. For instance *$[http|https]://google.com* will match either http://google.com or https://google.com.
Type each URL pattern in the textbox field and click the “Up” arrow to apply. Select an existing URL from the list and click the “Down” arrow to edit. Note: only one cartridge can be applied for a web page. If more than one cartridge matches the current web page, only the first one will be applied.

Click the Submit button to upload the cartridge as Unpublished. While a cartridge is unpublished only its owner will be able to use it, in this case the Admin. After testing the cartridge you may choose to publish it.

Edit existing cartridge

To modify a cartridge click the “Select action…” dropdown list near the cartridge you wish to modify and select **Edit**. The Edit cartridge dialog will appear.

Remove existing cartridge

To remove a cartridge click the “Select action…” dropdown list near the cartridge you wish to remove and select **Delete**.

Publish and Un-publish a cartridge

After a cartridge has been tested it can be published to all users. Click the “Select action…” dropdown list near the cartridge you wish to publish or un-publish and select **Publish** or **Unpublish**.

Un-published cartridges will be only usable by their owners, while published cartridges will be usable by all users. All cartridges should be tested before being published.