JIRA Reporting Gadgets

Zsolt Nagy

Bachelor's Thesis
May 2013

Software Engineering
School of Technology

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

DESCRIPTION

Author(s)	Type of publication	Date
NAGY, Zsolt	Bachelor´s Thesis	17052013
	Pages 56	Language English
		Permission for web publication (X)

Title
JIRA Reporting Gadget

Degree Programme
Software Engineering

Tutor(s)
PELTOMÄKI, Juha

Assigned by
AIRAKSINEN, Risto

Abstract

JIRA is one of the most popular issue tracking product. It holds high variety of data regarding issues stored in it. Reaching information in this enormous amount of data needs special tools - JIRA provides a few ones: issue navigator, filters, and gadgets. Even though these tools are easy to use for professional JIRA users, reaching information might be challenging for users with less knowledge about the product.

This need was recognized by Landis+Gyr Oy. The company started to develop different internal tools for reaching information stored in JIRA on very easy to use user interfaces. Three gadgets were implemented, two JIRA gadgets and a Windows 7 gadget. Development took three months.

The gadgets were built around Web technologies: HTML, CSS and JavaScript. Different libraries were created in order to reach data stored in JIRA. These libraries provide example for using a not well documented service of JIRA: searchrequest-xml.

Each implemented tool increased efficiency of searching in data stored in JIRA. As a direct result of using these gadgets, manager meetings became faster; much less effort had to be spent on searching for specific issues.

Keywords
HTML, CSS, JavaScript, JIRA, Windows, Google, OpenSocial, gadget, gadgets, searchrequest-xml, searchrequest.xml, jQuery, DataTables

Miscellaneous

Table of Content

List of Figures

List of Tables

1 Introduction

This document is the bachelor thesis of Zsolt Nagy, student of JAMK University of Applied sciences. The thesis describes the result of his internship at Landis+Gyr Oy from 2 January to 28.03.2013.

Landis+Gyr is market leader in electricity, gas, heat/cold and water metering solutions. The mission of the company is: “to help the world manage energy better”. (About Landis+Gyr)

The writer of the document implemented different tools while he spent his internship at the company. This document describes each implemented tools and also provides the basics of technologies that were used.

2 Used Technologies

This section explains the basics of technologies that are used for creating result of the writer’s internship at Landis+Gyr Oy.

2.1 XML

Extensible Markup Language (XML) is a markup language originally designed for large-scale electronic publishing. The language is used for exchange of different data on the Web end elsewhere. A few design goals of XML:

- XML shall be straightforwardly usable over the Internet.
- It shall be easy to write programs which process XML documents.
- XML documents should be human-legible and reasonably clear.
- XML documents shall be easy to create.

XML documents consist of markup and content. Markups are strings of characters that either begin with < and end with > or begin with & and end with a ; character. Strings of characters that are not markup are content. Except in CDATA section

where <![CDATA[and]]> are classified as markups, while the text between them is classified as content. (XML)

Tags are those markup constructs that begin with < and end with >. There are three types of tags:

- start-tags, e.g.: <section>
- end-tags; e.g.: </section>
- empty-element tags; e.g.: <line-break />

Element is a component that either begins with a start-tag or ends with a matching end-tag or only an empty-element tag. Start-tags and empty-elements may contain attributes: attribute is a name-value pair that exists within a start-tag or empty-element tag. e.g.:

```
<img src="img.png" alt='Picture taken by Photographer'/>
```

In this example “src” and “alt” are names –, “img.png” and “Picture taken by Photographer” are values of attributes.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Module>
    <ModulePrefs title="hello world example" />
    <Content type="html">
        <![CDATA[
<h1>Hello, world!</h1>
        ]]>
    </Content>
</Module>
```

Figure 1 example XML document – specification of a Google gadget

2.2 Client-side Web Technologies

Client side web technologies consist of markup languages, programming interfaces and languages, and standards for different standards related to document display and identification. This section explains those web technologies that were used for implementation of described tools. (What is Web Technology)

2.2.1 HTML

HTML (HyperText Markup Language) is the most commonly used markup language for creating WebPages and tools displayed by web-browsers (e.g. JIRA and Windows gadgets). (HTML)

HTML documents are built up using HTML elements. An HTML element is either content wrapped between start- and matching closing-tag or an empty element. Start-tags and empty-elements may contain attributes: attribute is a name-value pair that exists within a start-tag or empty-element tag, e.g.:

```
<img src="picture.png" alt="Example Picture"/>
```

General form of a HTML element with start- and closing-tags:

```
<tag attribute1="value1" attribute2="value2">content</tag>
```

HTML documents may contain scripts. Scripts are programs that can be executed by an interpreter. The most common scripting language used in browsers is JavaScript. See 2.2.3 for more information about JavaScript. (Scripts)

2.2.2 CSS

"Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation semantics (the look and formatting) of a document written in a markup language. Its most common application is to style web pages written in HTML and XHTML, but the language can also be applied to any kind of XML document, including plain XML, SVG and XUL." (Cascadian Style Sheets)

A stylesheet is a list of rules. A rule consists of a declaration block and selector(s). Selector defines the part of the markup that the style applies to. A selector may apply to:

- elements of a specific type: e.g. all divs: "div"
- elements having specific class: ".classname"
- elements having specific id: "#idname"

- elements in a specific position in the document tree
- elements having specific pseudo-class: ":pseudoclassname", e.g.: ":hover"

Selectors can be combined in many ways.

Declaration-block defines the style for those elements that are selected by the selector(s) of the rule. A declaration-block is a list of declarations between braces. A declaration is a property, a colon (:) and a value. Declarations are separated by a semi-colon (;).

```
    background-color: #7AB800;
    color: #FFF;
    white-space: nowrap;
}

.dataTable a {text-decoration:none}
.dataTable a:hover {text-decoration:underline;background-color:#DDD}

.dataTable tbody tr{background-color: #FFF;}
.dataTable tbody tr:hover{background-color: #EEE;}

.dataTable thead tr th.sorting_asc, .dataTable thead tr th.sorting_desc {background-color: #9AD820;}
.dataTable thead tr th:hover {background-color: #AAE830;}

.limitedColumn {width:1px;overflow:hidden;}   /*1px: 0 would work in firefox and IE, but not in chrome*/
.wideColumn {overflow:hidden;}

/*length of divs are limtied in cells - but overflow is visible - cells can be pushed until they reach max size of div*/
.limitedColumn div, .wideColumn div{max-width:10em;overflow:none;white-space:nowrap; margin: 0 2px;display:inline-block;}
newline after one label + block to have max-width working*/
```

Figure 2 example: CSS of one implemented gadget opened in a text editor

2.2.3 JavaScript

JavaScript is an interpreted computer programming language. It was developed in order to implement client-side scripts running in web-browsers that can interact with user, control the web-browser, communicate asynchronously and change content of document.

"JavaScript is a prototype-based scripting language that is dynamic, weakly typed, and has first-class functions. Its syntax was influenced by the language C. JavaScript copies many names and naming conventions from Java, but the two languages are otherwise unrelated and have very different semantics. The key design principles within JavaScript are taken from the Self and Scheme programming languages. It is a multi-paradigm language, supporting object-oriented, imperative, and functional programming styles." (JavaScript)

JavaScript programs can be integrated in HTML pages between <script> tags. All modern web-browser have JavaScript interpreter that makes the language usable in most platforms. (Comparison of Web browsers)

2.2.4 jQuery

"jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. With a combination of versatility and extensibility, jQuery has changed the way that millions of people write JavaScript." (jQuery)

Features of jQuery are listed below as follows (jQuery):

- DOM element selections using Sizzle selector engine
- DOM traversal and modification (including support for CSS 1-3)
- DOM manipulation based on CSS selectors that uses node elements name and node elements attributes (id and class) as criteria to build selectors
- Events
- Effects and animations
- AJAX
- Extensibility through plug-ins
- Utilities - such as user agent information, feature detection
- Compatibility methods that are natively available in modern browsers but need fall backs for older ones - For example the inArray() and each() functions.
- Multi-browser (not to be confused with cross-browser) support.

2.2.5 Datatables

"DataTables is a plug-in for the jQuery Javascript library. It is a highly flexible tool, based upon the foundations of progressive enhancement, which will add advanced interaction controls to any HTML table." (DataTables)

The tool is an extension of HTML tables. It provides several features. A list of features used in implemented tools is given below:

- Multi-column sorting with data type detection
- Smart handling of column widths
- Scrolling options for table viewport
- Sorting column(s) highlighting
- Sorting plugin

2.3 JIRA gadgets

"JIRA is a proprietary issue tracking product, developed by Atlassian, used for bug tracking, issue tracking and project management." (JIRA)

JIRA provides dashboard for its users (Using Dashboard Gadgets). Generally, dashboard is an easy to read, real-time user interface that provides information to its reader about current status or trends of an organization's performance indicators. (Dashboard (management information systems))

JIRA dashboards contain gadgets. Gadget is an information-box in which user can see information about projects and issues that are relevant for him/her (Customising the Dashboard). Technically gadgets are XML files that wrap around an HTML, CSS, JavaScript content.

Atlassian gadgets are extension of Google gadgets, they are created using Google gadgets.* API defined by OpenSocial specification. (Getting Started with Gadget Development)

There are two types of Atlassian gadgets: standalone –, and plugin gadgets which are described below. (Developing Standalone or Plugin Gadgets)

- Standalone gadgets consist entirely of HTML, CSS, and JavaScript. They are wrapped in the gadget-specification, which is an XML file. For installation, user must upload gadget-specification to a server where from the file can be

reached with a URL by the JIRA instance in which it will be installed. Gadget can be installed by providing this URL.

- Plugin gadgets are embedded into the plugin XML using the gadget plugin module. JIRA developers can use #-directives and Atlassian Gadget JavaScript Framework. To install the gadget, user must install the plugin.

The implemented Atlassian gadgets are standalone ones – plugin gadgets will not be described in more details in this document. Atlassian gadgets will be referred as JIRA gadgets later in this document.

Content of Google gadgets basically is a webpage in which developer can show any information that can be shown in the browser that opens gadget. Also, Google gadget framework provides different tools that are accessible from the content of the gadget. The Google gadget framework tools used in the implemented gadgets are User preferences and Title setting. (Gadgets XML Reference)

User preferences let developer to store and change user preferences for gadgets. The implemented tools store settings of filters and for limitation of number of fetched rows.

```
</ModulePrefs>
<UserPref name="rowNumLimit" display_name="maximum number o
<UserPref name="baseTitle" display_name="text of header" de
<UserPref name="foundfrom" display_name="Found from" defaul
required="true"/>
<UserPref name="labels" display_name="Labels" default_value
<UserPref name="statuses" display_name="Statuses" default_v
<UserPref name="priorities" display_name="Priorities" defau
<UserPref name="phases" display_name="Phases" default_value
required="true"/>
<Content type="html">
    <![CDATA[
```

Figure 3 UserPref elements of gadget specification of AimFilteringGadget

The title setting lets the developer change title of gadget . The implemented tools use the title of gadgets to inform users about the status of searching and loading

2.4 Windows gadgets

Windows gadgets are lightweight HTML, CSS and JavaScript based applications that can provide information and functionality from a high variety of sources such as local applications and controls, or websites and services. Technically, Windows gadgets are an XML file (manifest) and an HTML file that provides the shell for the user interface and the core functionality of the gadget. (Developing a Gadget for Windows Sidebar Part 1: The Basics)

3 Requirements

This section describes the environment used at Landis+Gyr R&D, high-level, and technology-specific requirements regarding each implemented tool.

3.1 Landis+Gyr R&D JIRA environment

This section describes environment in which requirements are defined.

Landis+Gyr uses JIRA for issue tracking in software development and maintenance. The implemented tools were designed to be used at R&D department of the company. This department uses a special configuration of JIRA.

Generally, JIRA defines different fields for each issue. Each field holds a property of an issue, e.g. type of issue, priority, creation date, etc. There are built-in and custom fields (Adding a Custom Field). Issues are separated to projects in JIRA. A JIRA project is basically a collection of issues (What is a Project).

Each project can have its own field configuration: behavior of built-in fields is defined and custom fields can be added (Specifying Field Behaviour). At Landis+Gyr, these projects were defined by software development lifecycle phase where issues were found from. The defined phases at Landis+Gyr R&D were:

- Development
- System Testing
- Release Testing and System Integration Testing

- Maintenance: in case issue was reported by a customer or Maintenance team recognized it

There are one or more JIRA projects defined for each phase.

Also, custom fields were defined for each project with the next names:

- “Severity”: severity of issue (high, normal, or low)
- “Found from”: stores the version of application where the issue was identified
- “CAL id”: in case the issue was reported by a customer then the issue is first stored in a CRM (Customer Relationship Management) system - this field holds the id of the report in the CRM system.

JIRA projects used at R&D of Landis+Gyr are also referred as AIM projects in this document.

3.2 High-Level requirements

There were three expected tool. This chapter collects High-level, technology-independent requirements regarding these tools.

3.2.1 Issue searcher by filters

“Tool for filtering issues” – The requirement is a tool that provides easy access for issues stored in JIRA. It can list issues that match filters set by user.

3.2.1.1 Filters

The user is able to select/set what values he/she accepts for different properties of issues. These properties are listed below:

- Found from: value of this property is stored in a custom field defined for projects of Landis+Gyr R&D
- Labels: the value of this property is the JIRA labels set for issue

- Status: value of this property is stored in field with name “status”, possible values: “Open”, “Resolved”, “Tested”, “Closed”
- Priority: value of this property is stored in field with the name “Priority”, possible values: “Urgent”, “High”, “Normal”, “Low”
- Phase: value of this property is defined by the project in which the issue is stored
- Created date: value of this property is defined in field with name “Created”.

3.2.1.2 Result of searching

Requirements regarding result of searching are:

1) The tool is able to show value of built-in –, and some custom fields. List of shown fields:
   - Type
   - Key
   - CAL id
   - Found from
   - Labels
   - Summary
   - Priority
   - Severity
   - Status
   - Resolution
   - Created date
   - Updated date
2) User can open each issue in JIRA.
3) Result can be exported in file format of Microsoft Excel.

3.2.2 Issue searcher by CAL id

The requirement is a tool for searching issues with specific CAL id. CAL id: in case the issue was reported by a customer then the issue is first stored in a CRM (Customer

Relationship Management) system - this field holds the id of the report in the CRM system.

The tool must have a simple interface for searching issues by CAL id. The user is able to set what value he/she accepts as CAL id or part of CAL id of issues. If the user sets only a part of CAL id then the tool is able to list all matching issues. Wildcards are usable when searching.

Generally, a wildcard is a character that represents any character or any string in a string (Wildcard character). JIRA defines two wildcards: “?” character stands for “any character exactly once” and “*” stands for any string (Performing Text Searches). Below are two examples of wildcards.

Example 1: when searching for "1.6.?"'

- "1.6.1" is accepted
- "1.6." is not accepted
- "1.6.123456" is not accepted

Example 2: when searching for "1.6.*”

- "1.6.1" is accepted
- "1.6." is accepted
- "1.6.1.123456" is accepted

JIRA does not let wildcards to be the first character for search requests. (Allow searching for part of a word (prefix / substring searches))

All requirements set for result of searching in AIM filtering gadget (see 3.2.1.2) are required for result of searching in this gadget too.

3.2.3 JQL query follower gadget

The requirement is a tool for following result of a JQL query on user's desktop. A JQL query (or clause) is an expression written in JQL (JIRA Query language). It is an SQL like query language for searching issues stored in JIRA. (Advanced Searching)

JQL example:

```
priority IN (Urgent, High) AND project IN (TEST1, TEST2)
and comment ~ "text"
```

This query searches for issues where priority is either "Urgent" or "High" and project is either TEST1 or TEST2 and at least one comment contains "text" string.

The next properties should be visible for issues:

- Key
- Summary
- Status
- Priority
- Type

The result should be updated in every set interval. This interval has to be able to be changed by user.

The user has to be able to open each issue in JIRA from interface of gadget.

The user has to be able to open result in searching interface of JIRA from interface of gadget.

3.3 Technology Specific requirements

This section explains technology specific requirements regarding implemented tools.

3.3.1 Issue searcher by filters

The selected technology for this tool is standalone JIRA gadget. JIRA gadgets are part of JIRA user interface; it is easy to reach them while working in JIRA. Also programmer doesn't need to take care of authentication when searching from these tools.

3.3.1.1 Filters

One dropdown-menu is defined for each of the next properties as follows below:

- Found from
- Labels
- Status
- Priority
- Phase

These drop-down menus contain multiselects. Multiselect is a list where user can select for each value whether or not it is accepted.

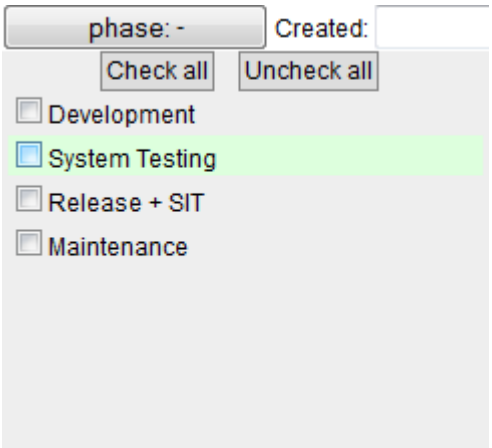

Figure 4 Eric Hynds' multiselect as filter (jQuery UI MultiSelect Widget)

The user has to be able to set what values should appear in each drop-down menu.

Created date property can be limited by 2 textfields that define the range in which the creation of issues is accepted.

3.3.1.2 Result

The user can see the result of searching in a table in the gadget itself or on the searching interface of JIRA. Two separate buttons are defined for these functions. When showing the result in the gadget, the data is fetched to a table - each row of this table represents one issue.

There are two types of columns: limited-width columns, and summary. Limited-width columns always strive to have the minimum possible width: when the content of a column is wider than the defined limit, then width of column is set to limit and the overflow of the content becomes hidden. When the width of content is smaller than the limit, then the width of the column is set to match the width of the content. The width of summary is always set to fill the usable space. There are no line-breaks in cells of table.

Rows of the table are sortable by any of the columns. The order of sorting is alphabetical by default. There are exceptions.

The next columns are sorted using Natural order:

- Key
- CAL
- Found from

Natural sorting refers to the process when letters are compared in alphabetical order, however when numbers appear in strings, they are compared by their value instead of character-by-character. Example: a10b compared with a9b. In alphabetical order: a10b < a9b, but in natural order: a9b < a10b. (What is natural ordering)

The next columns are sorted using “LNHU” order:

- Priority
- Severity

LNHU order is defined in the next way: null or unknown < “Low” < “Normal” < High” < “Urgent”.

Other requirements regarding the tool are as follows:

- The user can open each specific issue by clicking on its key or summary.
- When issues are being fetched into the gadget, the user is able to see the status of the process; the user is also able to stop the process with a button.
- The user is informed in case any error appears while searching.
- The user is able to see the content of the gadget on full page by clicking on a button.

3.3.2 Issue searcher by CAL id

The selected tool was standalone JIRA gadget for the same reasons as it was selected for Aim Filtering Gadget. One advantage is also that a large ration of code is shared between the two tools; it can be reused this way.

A text field is defined where users can write the CAL id or part of CAL id that he/she wants to search for. Wildcards are usable in this field. The tool searches for CAL ids both with and without “CAL-” prefix.

Requirements regarding result are the same as for filtering gadget (see 3.3.1.2).

3.3.3 JQL Query Follower

The selected tool was Windows gadget. Using this tool user is able to see the result without opening web-browser.

The user is able to set up connection: URL of JIRA instance in which user wants to search, username and password of user. User is able to test if set parameters are correct with a button: when button is clicked, user sees a message whether or not username and password are accepted for set JIRA instance.

The user is also able to set the JQL query in a textfield that will be searched. User is informed whether or not the set query is syntactically correct by clicking on a button.

The user can set the interval between refreshes in a textfield.

The user can search the query anytime by clicking on a button. The result is then fetched to gadget. The result is updated within every set interval.

Each found issue is represented by a rectangle where the next values are visible:

- Key
- Summary
- icon of status
- icon of priority
- icon of type

Result shows all loaded issues. In case the number of issues is too high to show in gadget, user can use a scrollbar for navigation.

The user can open result of searching in searching interface of JIRA by clicking on a button.

The user can open each specific issue by clicking on its summary.

4 Reporting Libraries

Two libraries were created to fulfill the requirements as follows:

- JqlQuerySearcherJs for searching issues that match given JQL query
- AimIssuesToTableJs for searching issues by given filters –, and loading them to an HTML table

4.1 JqlQuerySearcherJs

JqlQuerySearcherJs is a library written in JavaScript that provides searching functionality for issues in JIRA. It lets users search JIRA issues matching JQL queries.

```
<script src="JqlQuerySearcher.js"></script>
<div id="result"></div>
<script>
    /* constructs the loader object
    iterates through each found issues and loads them to the "result" div
    action: defines the action to be run for each xml representation of issues*/
    var loader = new IssueListIterator({
        action: function(issueXml) {
            //object that makes fields of xml representation of issue easily reachable
            var issue = new Issue(issueXml);
            document.getElementById("result").innerHTML += "<div>"+issue.fieldText("key")+"</div>";
        },
        maxNum: 100 //maximum number of issues to be fetched
    });

    /* constructs the searcher object.
    baseUrl: take a look at the addressbar of your browser when you open JIRA
    successCallback: function to run when searching is successfull
    errorCallback: function to run when error occurs while searching
    abortedCallback: function to run when searching is aborted by user or when same origin policy is offended*/
    var jqlQuerySearcher = new JqlQuerySearcher({
        baseUrl:        "http://jira:8080",
        successCallback:    function(xml) {loader.start(xml);},
        errorCallback:      function(responseText) {document.getElementById("result").innerHTML = responseText;},
        abortedCallback:    function() {document.getElementById("result").innerHTML = "aborted searching";}
    });
    jqlQuerySearcher.search({
        query:      "project = test",    //the jql query to be searched
        username:   "YOUR_JIRA_USERNAME",
        password:   "YOUR_JIRA_PASSWORD"
    });
</script>
```

Figure 5 Tutorial explaining the basic usage of JqlQuerySearcherJs.

Figure 5 explains the basic usage of the library. In case this HTML page is content of a JIRA gadget then the example works in all main web-browsers. Otherwise it works only if Same Origin Policy is not followed (e.g. Internet Explorer 9). (Same Origin Policy)

4.1.1 SearchRequest-xml

JqlQuerySearcherJs uses the searchrequest-xml. Searchrequest-xml is a service of JIRA for searching issues with a simple URL (Retrieving and Filtering JIRA issues). The requested data is returned in an XML file. The URL of the service is:

`_Jira_base_URL_/sr/jira.issueviews:searchrequest-xml/temp/SearchRequest.xml`

Searching is configured using parameters. The list and explanation of some parameters is given below:

- Field value restrictions: the left part of expression is a fieldname, the right part is a value. Only those issues will be returned where the value of given field equals the right side of the expression (e.g. project=TEST)
- JQL query: only those issues will be returned that match the given JQL query, e.g. jqlQuery=project%3DTEST (note that equals mark is escaped)
- Field name: only those fields are returned that are set with “field=fieldname” parameters (field=key). If no field name parameter is provided then all fields are returned.
- username and password (os_username=name&os_password=password)

Parameters are separated from URL of searchrequest-xml by a ? character. Parameters are separeted by & characters. Below is an example of this as follows:

```
http%3A//jira%3A8080/sr/jira.issueviews:searchrequest-
xml/temp/SearchRequest.xml?jqlQuery=%u2019project%3DTEST%
u2019%26field%3Dsummary
```

Note to the reader: query has to be escaped in URL.

JqlQuerySearcherJs uses JQL query, field names and optionally username+password as parameters.

The library contains three classes:

- JqlQuerySearcher: provides searching functionality for JIRA issues
- IssueListIterator: iterates through XML representing issues
- Issue: provides functions to reach content of JIRA issues

4.1.2 JqlQuerySearcher

JqlQuerySearcher provides searching functionality for JIRA issues – after configuration, programmers can use “search” method that returns an XML representing the issues that match the query.

The member functions of the class are the following:

JqlQuerySearcher(c)

JqlQuerySearcher is the constructor of the class.

name of parameter	
c.baseUrl required	base URL of JIRA e.g. "http://jira:8080"
c.searchRequestXml	URL of searchRequestXml; default: "sr/jira.issueviews:searchrequest-xml/temp/SearchRequest.xml"
c.navigatorJspa	URL of navigator.jspa; default: "secure/IssueNavigator!executeAdvanced.jspa"
c.successCallback	function(xml) runs when searching was successful without error
c.abortedCallback	function() runs when searching was stopped by user or when same origin policy is offended
c.errorCallback	function(responseText) runs when error occurs while searching

Table 1 API documentation of JqlQuerySearcher(c)

The constructor expects one configuration parameter. Configuration parameter is an object, attributes of this object hold the information that programmer passes to

method. In case the required parameter is not given, the constructor provides an error message for the user.

```
function JqlQuerySearcher(c) {
    this.baseUrl       = c.baseUrl;
    this.searchRequestXml   = c.searchRequestXml   || "sr/jira.issueviews:searchre
    this.navigatorJspa      = c.navigatorJspa      || "secure/IssueNavigator!execu
    this.successCallback    = c.successCallback    || function(xml) {};
    this.abortedCallback    = c.abortedCallback    || function() {};
    this.errorCallback      = c.errorCallback      || function(responseText) {};

    if(!this.baseUrl) {
        alert("JqlQuerySearcher: error, baseUrl has to be set");
        return;
    }
```

Figure 6 part of implementation of constructor of JqlQuerySearcher

search(c)

search(c) searches issues matching given query, calls successCallback(xml) when successful.

parameter	
c.query required	query that will be searched
c.fields	array of fields: only these fields will be returned - leaving unnecessary fields increases speed of searching dramatically e.g. description
c.username	username of JIRA user (used only if password is also given)
c.password	password for JIRA user (used only if username is also given)

c.tempMax	maximum number of searched rows

Table 2 API documentation of search(c)

JqlQuerySearcher has an instance of XMLHttpRequest. The tool provides scripted client functionality for transferring data between a client and a server (XMLHttpRequest).This method sets up this XMLHttpRequest instance so that it can send a request to searchrequest-xml (see 4.1.1) service of JIRA. Sending request to searchrequest-xml responses the XML representation of matching issues.

The method runs different callback functions of JqlQuerySearcher:

- successCallback is called when searching is successfully finished. Parameter of this function is the XML representation of issues found - the data given by searchrequest-xml.
- abortedCallback is called when request is aborted by user or when same origin polocy is offended. There is no parameter for this callback.
- errorCallback is called in any other case. Responsetext of request is given as parameter.

stop()

stop() method stops searching. The method executes abort() method of XMLHttpRequest instance of JqlQuerySearcher and executes abortedCallback method.

openinJira(query)

openinJira(query) opens query in a new browser window using "GET" request. The function opens the query on searching interface of JIRA. This interface can be reached by opening an URL. Parameters of this URL define the query to be searched.

When using from Windows gadget, then user has to log in to JIRA to use this function. Username and password cannot be given for this function; it is a serious security risk to provide these parameters as they are visible in the URL.

```
this.openinJira = function(query) {
    if(query == null) {//why not "!c.query"? => because i want to accept "" as query
        alert("JqlQuerySearcher.search: error, query has to be set");
        return;
    }
    var url = this.baseUrl+"/"+this.navigatorJspa + '?jqlQuery=' + escape(query) + "&runQuery=true";
    window.open(url);
}
```

Figure 7 Implementation of openinJira function

4.1.3 IssueListIterator

IssueListIterator iterates through XML representation of issues returned by searhrequest-xml. Runs given action for each iteration and callbacks to provide information about process. The browser is kept responsive while the iteration is going on.

The members functions of the class are the following:

IssueListIterator(c)

This function is the constructor of the class, with the following parameters:

parameter	
c.action	function(issueXml) run for each issue
c.maxNum	maximum number of iterations, negative value => no limit (default: 1200)
c.iteratedCallback	function(looped,found) runs after action is executed on an issue (after each iteration)
c.finishedCallback	function(looped,found) runs when searching is finished (all issues were looped or limit reached or user stopped

	iteration)

Table 3 API documentation of IssueListIterator(c)

start(issueListXml)

start(issueListXml) starts iteration. Given parameter is XML representing issues, data returned from JIRA searchrequest-xml.

There is a method defined in IssueListIterator: the step method. This is a recursive function that executes itself as the last step of its running. The execution occurs using "window.setTimeout" method - that gives the resources back to the browser for a while when asked. This way the browser is responsive while the iteration goes on.

```
    this.action(this.issueList[i]);
    this.iteratedCallback(i+1, this.issueList.length);
    this.loopCount = i+1;

    var parent = this;
    window.setTimeout(function() {parent.step(i+1);}, 0);
}
```

Figure 8 part of implementation of step method of IssueListIterator

stop()

stop() method stops iteration and calls finished callback is.

4.1.4 Issue

This class provides functions to reach content of issues easily: value of fields, value of custom fields, etc. Issues are represented as XML elements in data returned by searchrequest-xml (see 4.1.1); this class provides easy access to information stored in this structure.

Issue(issueXml)

This function is the constructor of the class, with one parameter: XML representing the issue.

fieldText(field)

fieldText(field) returns text value of given built-in field name. There is an element for

each built-in field in XML representation of JIRA issues returned by searchrequest-xml. The names of the tag of these elements are names of the built-in fields. For example: tagname of project field is “project”. This function simply returns the value of given built-in field of the issue.

fieldAttribute(field,attribute)

fieldAttribute(field,attribute) returns value of given attribute of given field.

Some built-in elements provide extra information stored in attributes. For example: there is an attribute defined for type field that stores the URL of the icon of the type. This function returns the value of the given attribute of the given field.

customFieldText(field)

customFieldText(field) returns the text value of the given custom field

JIRA users can define custom fields (see 3.1). Values of custom fields are stored differently than values of the built-in fields in XML representation of issues. There is an element defined for issues with the name: “customfields”. This element contains elements with the name: “customfield”. Each of these elements contains two elements: one with the name “customfieldname” that holds name of custom field and one with the name “customfieldvalue” that holds value of field.

4.2 AimIssuesToTableJs

AimIssuesToTableJs is a library written in JavaScript that provides searching functionality for issues of AIM projects in JIRA. The library provides tools to fetch data into an HTML table. The dependencies are jQuery, jQuery Datatables, jQuery UI (datePicker) and JqlQuerySearcherJs.

```
<script src="http://code.jquery.com/jquery-1.8.3.min.js"></script>
<script src="http://code.jquery.com/ui/1.9.2/jquery-ui.js"></script>
<script src="http://cdnjs.cloudflare.com/ajax/libs/datatables/1.9.4/jquery.dataTables.min.js"></script>
<script src="JqlQuerySearcher.js"></script>
<script src="AimIssuesToTable.js"></script>

<table id="issuesTable">
    <thead>
        <tr>
            <th>Type</th>
            <th>Key</th>
            <th>Found from</th>
            <th>Summary</th>
            <th>Created</th>
        </tr>
    </thead>
</table>

<script>
    var issueSearcher = new AimIssuesToTable({
        baseUrl:        "http://jira:8080",
        tableId:        "issuesTable", //result will be fetched to table with this id
        rowNumLimit:    10,
        errorCallback: function(responseText) {alert("error while searching");}
    });

    //searchers issues of AIM projects that match given filters
    issueSearcher.search({
        severities:     ["high","normal"], //filter for severity field
        foundfroms:     ["6.1.0","6.1.1"], //filter for 'found from' field
        username:       "YOUR_JIRA_USERNAME",
        password:       "YOUR_JIRA_PASSWORD"
    });
</script>
```

Figure 9 Tutorial explaining the basic usage of AimIssuesToTableJs

Figure 9 explains the basic usage of the library. In case this HTML page is content of a JIRA gadget then the example works in all main web-browsers. Otherwise it works only if Same Origin Policy is not followed (e.g. Internet Explorer).

The library contains four classes:

- AimIssuesToTable: searching JIRA issues of AIM projects and loading them to a table
- ExpressionBuilder: tool for creating general expressions
- AimJqlQueryBuilder: a simple JQL Query builder for AIM projects
- AimIssue tool representing Landis AIM projects' one JIRA issue

4.2.1 AimIssuesToTable

AimIssuesToTable is a class for searching JIRA issues of AIM projects and loading them to a table. The class has an instance of JqlQuerySearcher and an instance of IssueListIterator.

AimIssuesToTable(c)

AimIssuesToTable(c) is the constructor for AimIssuesToTable class.

parameter	
c.baseUrl required	base URL of JIRA e.g. "http://jira:8080" (required)
c.searchRequestXml	URL of searchRequestXml; default: "sr/jira.issueviews:searchrequest-xml/temp/SearchRequest.xml"
c.tableId required	the id of table where data is put
c.searchingCallback	function() runs when searching is started
c.loadedCallback	function(loaded,found) runs when an issue is loaded to the table
c.finishedCallback	function(shown,found) runs when searching is finished
c.errorCallback	function(responseText) runs when error occurs while searching

Table 4 API documentation of AimIssuesToTable(c)

search(c)

search(c) searches for issues that matches given filter criterias defined in parameters

parameter	
c.foundfroms	array of acceptable versions (found from)
c.allFoundfroms	array of all versions available for selection -> in order to make "other" option usable
c.statuses	array of acceptable statuses
c.allStatuses	array of all statuses available for selection -> in order to make "other" option usable
c.priorities	array of acceptable priorities (urgent,high,normal,low)
c.allPriorities	array of all priorities for selection -> in order to make "other" option usable
c.phases	array of acceptable phases (dev, st, sit, maint)
c.createdDateFrom	only those issues will be listed that were created later/equal than this date
c.createdDateTo	only those issues will be listed that were created

	earlier/equal than this date

Table 5 API documentation of search(c)

The function creates a JQL query from given parameters. The query is created using AimJqlQueryBuilder class. After query is created, search method of JqlQuerySearcher instance of the class is executed. Callback functions of this class are connected to callback functions of JqlQuerySearcher instance.

openinJira(c)

openinJira(c) opens query in searching interface of JIRA on a new browser window using "GET" request.

parameter	
c.foundfroms	array of acceptable versions (found from)
c.allFoundfroms	array of all versions available for selection -> in order to make "other" option usable
c.statuses	array of acceptable statuses
c.allStatuses	array of all statuses available for selection -> in order to make "other" option usable
c.priorities	array of acceptable priorities (urgent,high,normal,low)

c.allPriorities	array of all priorities for selection -> in order to make "other" option usable
c.phases	array of acceptable phases (dev, st, sit, maint)
c.createdDateFrom	only those issues will be listed that were created later/equal than this date
c.createdDateTo	only those issues will be listed that were created earlier/equal than this date

Table 6 API documentation of openinJira(c)

The function creates a JQL query from given parameters. The query is created using AimJqlQueryBuilder class. After the query is created, openinJira method of JqlQuerySearcher instance of the class is executed.

stop()

stop() method stops searching and loading if any of them is going on. It executes stop method of IssueListIterator and JqlQuerySearcher instances of the class.

4.2.2 ExpressionBuilder

ExpressionBuilder is a tool for creating expressions by putting an operator between a list of expressions and surround them with brackets. If expressions are: {e1,e2,e3,e4,e5,e6...} and operator is 'o', then toString() provides: "(e1) o (e2) o (e3) ..."

member functions of the class:

ExpressionBuilder(operator)

ExpressionBuilder(operator) is the constructor for ExpressionBuilder class. Parameter

of the function is the operator that is put between expressions when executing toString().

addExpression(value)

addExpression(value) adds given value to expressions.

toString()

If expressions are: {e1, e2, e3, e4, e5, e6...} and operator is 'o', then toString() provides: "(e1) o (e2) o (e3) ..."

```
/** if expressions are: {e1,e2,e3,e4,e5,e6...} and operator is 'o', then toString() pr
this.toString = function() {
    if(this.expressions.length == 0) return "";
    var result = "("+this.expressions[0]+")";
    for(var i=1; i<this.expressions.length; i++)
        result += " "+this.operator+" ("+this.expressions[i]+")";
    return result;
}
```

Figure 10 implementation of toString method of ExpressionBuilder

4.2.3 AimJqlQueryBuilder

Builds a JQL query from given filters. Different expressions are connected with AND logical operator. Member functions of the class are the following:

AimJqlQueryBuilder()

AimJqlQueryBuilder() is constructor for the class.

addFieldEqualsFilter(field,values,allValues)

Searches for issues where the value of the specified field is one of the given values**.** When '?' is in values then those values will also be searched that are NOT member of allValues**.** When '-' is in values then empty value is also accepted

Examples:

- addFieldEqualsFilter(“project”, [“A”,”B”, “?”], [“A”, “B”, “C”]) => searches for issues where name of project is “A”, “B” or anything that is not “A”, “B” pr “C”
- addFieldEqualsFilter(“priority”, [“high”,”-”], [“high”,”low”]) => searches for issues where priority is “high” or non-set.

parameter	
field	name of field thats value is checked
values	array of values that are accepted as value of field (except for undefined, null, "", '?','-')
allValues	if '?' is part of 'values' array, values that are NOT part of allValues are also accepted

Table 7 API documentation of addFieldEqualsFilter(field,values,allValues)

addFieldContainsFilter(field,values,allValues)

Searches for issues where the value of the specified field matches any of given values - either an exact match or a "fuzzy" match (Advanced Searching). When ? is in values then all fields will also be searched that do NOT match any of allValues. When "-" is a member of values then empty value is also accepted.

Example:

- addFieldContainsFilter(“foundfrom”, [“1.6*”, “?”], [“1.6*”, “1.7*”]) => searches for issues where value of field with name “foundfrom” matches “1.6*” expression or doesn’t match either “1.6*” nor “1.7*”. This functionality is usable when the user wants to see what values are not listed as options.

parameter	

field	name of field thats value is checked
values	array of values that are accepted as matching value of field (except for undefined, null, "", '?','-')
allValues	if '?' is part of 'values' array, values that are NOT part of allValues are also accepted

Table 8 API documentation of addFieldContainsFilter(field,values,allValues)

4.2.4 AimIssue

This is the class representing Landis AIM projects' one JIRA issue. It has an instance of Issue class (see 4.1.4 for details).

AimIssue(issueDom)

AimIssue(issueDom) is constructor for AimIssue class. Given parameter is the DOM element representing the issue.

toRow(cols)

toRow(cols) returns an array with data for given column names. The function goes through each elements of the given array and collects information defined for each column names using its Issue instance.

5 Implementation of gadgets

This section explains implementation of each expected gadget.

5.1 AimFilteringGadget

Requirements are collected at 3.2.1 and 3.3.1 (High-level - and technology-specific requirements of AIM Filtering Gadget).

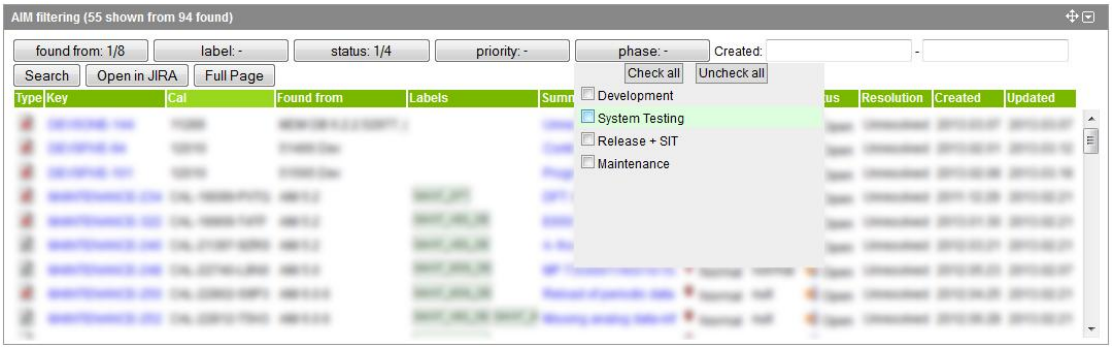

Figure 11 AIM filtering gadget

The tool has one instance of AimIssuesToTable that searches for issues and fetches data to an HTML table. Please see 4.2.1 for more details about the tool. The user interface of the gadget is divided into 4 parts:

- Titlebar
- Toolbar
- Result
- Search div

5.1.1 Titlebar

Titlebar is a part of each Google gadget. By default, it shows the title of the gadget, however gadgets can change the shown string. AIM Filtering gadget provides information about the status of searching and loading in brackets next to the title of gadget.

- When searching is going on, the shown text is “searching...”
- if an error occurs, “error” string is shown
- when loading is going on, “Loading: x/y...” is shown where x is the number of already loaded rows and y is the overall number of found issues
- if loading is finished, “x shown from y found” text appears where x is the number of loaded issues and y is number of found issues

5.1.2 Toolbar

Toolbar contains filters: buttons that open dropdown menus, and textfields for date limits. In dropdown menus the user can select which values he/she accepts for specific fields. Available options can be set in settings of the gadget. There are buttons for selecting all and a button for selecting none of the options listed. When no option is selected then the filter is not used while searching.

These dropdown menus are multiselects. Multiselects are implemented using Eric Hynds' multiselect plugin for jQuery (jQuery UI MultiSelect Widget). This tool provides a drop-down menu for selecting options of <select> tag of HTML.

When the user clicks on textfields of date limits, a calendar pops up.

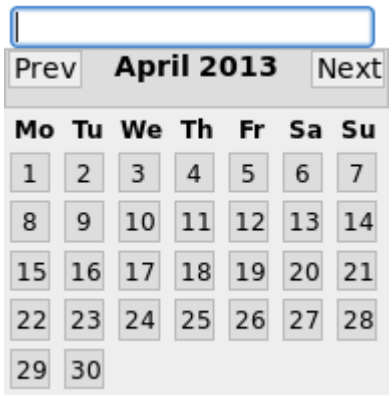

Figure 12 calendar where user can select date

Calendars are implemented using DatePicker plugin of jQuery. DatePicker lets user to open a calendar by focusing on a text field. (DatePicker)

The toolbar contains a button with text "Search". By clicking this button, the user can start searching issues that match the set up filters. While searching and fetching is going on, the table of result becomes invisible and search div becomes visible. For more details about these elements please see 5.1.2 and 5.1.3. By clicking this button, "search" method of gadget's AimIssuesToTable instance is executed.

Toolbar also contains a button with text “Open in JIRA”. By clicking this button, user can open result in searching interface of JIRA. By clicking this button, “openinJira” method of gadget’s AimIssuesToTable instance is executed.

The last button on the toolbar is a button for opening gadget on full page. The content of gadgets in JIRA is generally in iframe on dashboards. Iframe is an HTML element that lets programmers place an HTML page in a frame of another HTML page (HTML element). The size of the provided frame is limited; user may need to see content of the gadget on full page. This button provides this functionality.

5.1.2.1 Settings of filters

The user can set the list of options that are visible on each filter in user preferences of gadget. Available options are divided by only a comma character. For example the options for priority filter might be: urgent, high, normal, low.

Options can have different real values than shown values, e.g.:

- "sit|Release + SIT" => has the value "sit", shown as "Release + SIT" in the list of options.
- "?|other" => has the value "?", shown as "other"

There are two special values:

- "?" means “all values that are not listed in the list of options”
- "-" means “value is not exists” or “value is null”

These options are highlighted in the select menu.

Wildcards can be used at "Found from" field. Please see 3.2.2 for details about wildcards.

```
/** adds options to 'selectId' select that are listed in prefName preference
* options are devided by coma
* options can have different value and shown name: value|shown_name*/
function setOptionsFromUserPrefs(selectId,prefs,prefName) {
    var prefarr = prefs.getString(prefName).split(",");
    var selectdom = document.getElementById(selectId);

    for(var i=0; i<prefarr.length; i++) {
        var value = prefarr[i].split("|")[0];
        var shown = prefarr[i].split("|")[1] || value;// "value|shown_value" (shown value is optional)

        var optiondom = document.createElement("option");
        optiondom.setAttribute("value",value);
        optiondom.innerHTML = shown;
        if(value == "?" || value == "-") optiondom.className = "highlightedoption";

        selectdom.appendChild(optiondom);
    }
}
```

Figure 13 method that sets up filters from user preferences

5.1.3 Result

Result is an HTML table, the content of which is generated by AimIssuesToTableJs instance of the gadget. It is visible only if searching is successfully finished. The element is hidden while searching or loading is going on, or when error occurs while searching.

The table is implemented using Datatables plugin of jQuery.

Each column is sortable. Rows of table are sortable by any of the columns. Order of sorting is alphabetical, except:

- Key, CAL, Found from: natural sorting is defined for these columns
- Priority, Severty: order of values is the next: null or unkown < “Low” < “Normal” < High” < “Urgent”. This sorting method will be refered as “LNHU” sorting later in this document.

For more details about the requirements regarding sorting please see 3.3.1.2.

Natural and LNHU sorting are implemented by creating sorting plugins for Datatables. Sorting plugins lets programmer to define special sorting for rows stored in Datatables. There are two APIs provided: typed based sorting for static data; and custom data source sorting for dynamically changing date. JiraAimFilteringGadget uses only type based sorting. (Sorting plug-ins)

Natural sorting plugin is implemented based on Jim Palmer's algorithm (NaturalSort.js). The implementation of LNHU sorting can be seen on Figure 20.

```
jQuery.extend(jQuery.fn.dataTableExt.oSort, {
    /**this sorting plugin sorts low-normal-high-urgent values that are wraped in an HTML element (in divs mostly)*/
    "domlnhu-pre": function(a) {
        var div = document.createElement('div');
        div.innerHTML = a;
        value = div.textContent.toLowerCase();
        switch(value) {
            case "low":     return 1;
            case "normal":  return 2;
            case "high":    return 3;
            case "urgent":  return 4;
            default:        return 0;
        }
    },
    "domlnhu-asc": function(a,b) {return ((a < b) ? -1 : ((a > b) ? 1 : 0));},
    "domlnhu-desc": function(a,b) {return ((a < b) ? 1 : ((a > b) ? -1 : 0));}
});
```

Figure 14 Implementation of LNHU sorting plugin

The requirements regarding the width of columns are implemented using CSS rules. The contents of cells are always wrapped in div elements. The max-width attribute of these elements are defined; however they let their content overflow.

The width of columns that have limited width is set to 1 pixel. This way the width of these columns is set to minimum possible, which equals the width of divs in these columns - but columns do not let their content to overflow. The width of the table is set to 100% so the columns that have no limitation use the possible space.

```
<style type="text/css">
/*keeps table filling page
keeps wide columns fill space*/
table {width:100%;}

/*show border and hide overflowing text*/
td {border:1px solid;overflow:hidden;}

/*limits width of column to minimum possible
1px: 0 would work in firefox and IE, but not in chrome*/
.limitedColumn {width:1px;}

td div {
    /*when width of cell is set to 1px but width of div is more then 1px =>
    cell will have the width of div*/
    max-width:10em;

    /*this option lets text overflow in case cell is wider then max-width =>
    text is visible in wide columns*/
    overflow:none;
    white-space:nowrap;/*no ew line*/
    display:block; /*max-width is used when element is block-type*/
}
</style>

<table>
    <tr>
        <td class="limitedColumn"><div>looooong text</div></td>
        <td><div>looooong text demonstrating summary column</div></td>
        <td class="limitedColumn"><div>another looong text</div></td>
    </tr>
</table>
```

Figure 15 HTML page demonstrating expected behavior of width of columns

The requirements regarding openable issues are implemented in AimIssue (see 4.2.4).

5.1.4 Search div

This element is visible when searching and loading is going on, or when an error occurs while searching.

It consists of an element that informs the user about searching by showing the text: “searching...” and informs the user about the status of loading by showing: “Loading: x / y...” where x is the number of loaded rows and y is the number of found rows. When an error occurs while searching, this element shows the response gotten from JIRA.

The other element in Search div is a button that lets the user stop searching or loading. If the button is pressed while searching is going on, the table of results

become visible with no row. If the button is pressed while loading is going on, the table becomes visible with the rows that are already loaded.

This feature is implemented using different callback functions of AimIssuesToTable. For more information, please see 4.2.1.

5.2 CalSearcherGadget

CAL searcher gadget provides searching ability for CAL id of issues of AIM projects. It generates result to a table in the gadget itself or open the result in searching interface of JIRA.

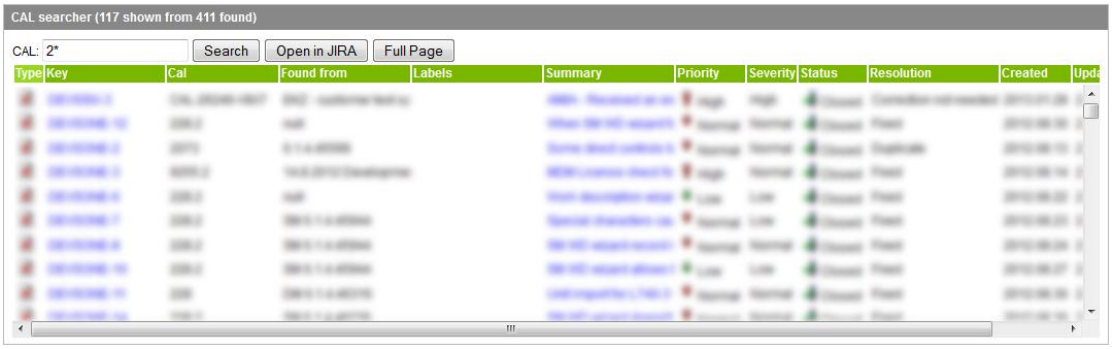

Figure 16 CAL searcher gadget

5.2.1 Toolbar

The toolbar contains a text field, which is a filter for cal id of issues. The user can set what value he/she accepts as cal id of issues. For detailed requirements, please see 3.2.2.

Toolbar contains a button with text “Search”. By clicking this button, user can start searching issues that match given text. The text is searched in ways: with - and without “CAL-” prefix. While searching and fetching is going on, the table of result becomes invisible and search div becomes visible. For more details about these elements please see 5.2.2 and 5.2.3. By clicking this button, “search” method of gadget’s AimIssuesToTable instance is executed.

Toolbar also contains a button with text “Open in JIRA”. By clicking this button, user can open result in searching interface of JIRA. By clicking this button, “openinJira” method of gadget’s AimIssuesToTable instance is executed.

The last button on the toolbar is a button for opening the gadget on full page. The content of gadgets in JIRA is generally in an iframe on dashboards. Iframe is an HTML element that lets programmers place an HTML page in a frame of another HTML page. The size of the provided frame is limited, user may need to see content of gadget on full page. This button provides this functionality.

5.2.2 Result

The implementation of result is the same as implementation of result of AIM Filtering gadget, please see 5.1.2 for details.

5.2.3 Search div

Implementation of search div is the same as implementation search div of AIM Filtering gadget, please see 5.1.3 for details.

5.3 JqlQueryFollowerGadget

JQL query Follower gadget is a Windows gadget that shows the result of a JQL query on a Windows 7 desktop and refreshes the list frequently. Detailed requirements are collected at 3.3.3.

5.3.1 Settings dialog

Connection settings are part of settings dialog of gadget.

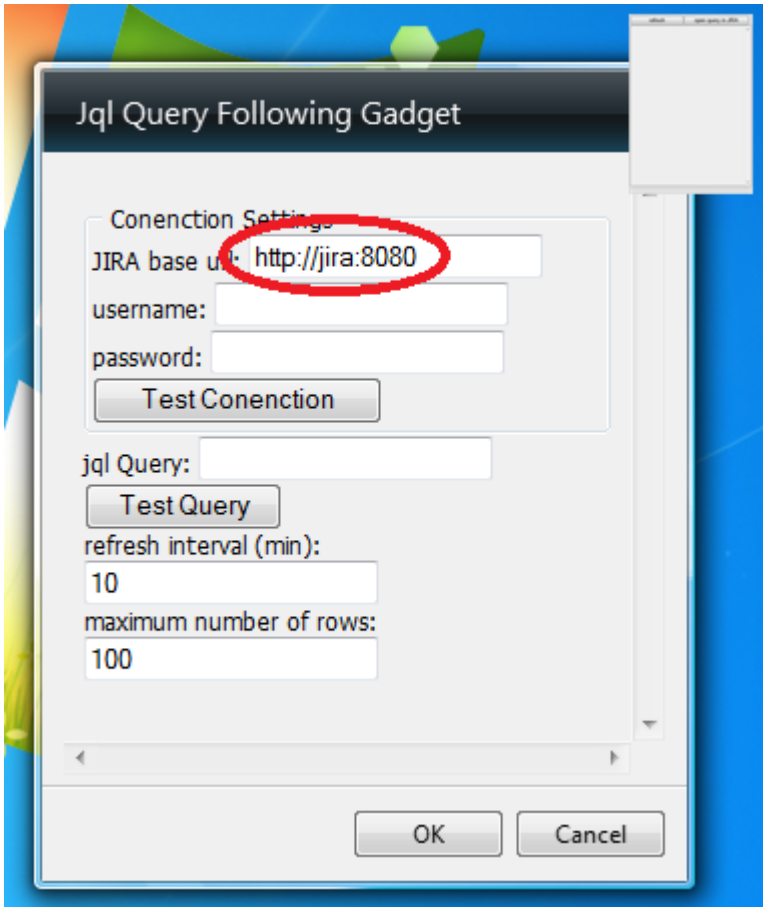

Figure 17 settings dialog of JQL Query Following gadget

5.3.1.1 Connection settings

A fieldset is defined for the next settings: JIRA base URL, username, password, message showing status of connection and button for testing connection. A fieldset generally is an HTML element to group related elements in a form (Fieldset tag).

By clicking on “Test connection” button, a request is sent to JIRA instance of the dialog. The request contains a simple JQL query that tests if the user (with given username and password) has access to one of the projects stored in JIRA. The result becomes visible above the button.

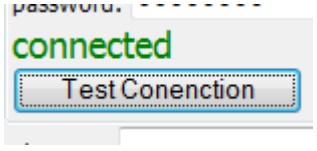

Figure 18 successful testing of settings

Technically, by clicking on this button the user constructs a new JqlQuerySearcherJs instance and executes “search” method of it. The given query for the button is: “project = PROJECTNAME AND created >= now()”. This request sends error in case user does not have access to PROJECTNAME project, otherwise it sends an empty list of issues.

All callback functions of this JqlQuerySearcherJs instance are defined:

- successCallback: a green “connected” string becomes visible above “Test Connection” button (see Figure 18)
- errorCallback: a red “not connected” string becomes visible
- abortedCallback: a red “unknown” string becomes visible

5.3.1.2 Other settings

A text field is defined in which user can set the JQL Query to be followed. A button under this textfield lets user to check if the query is syntactically correct. In case query is correct, a green “connected” string becomes visible, otherwise a red “icorrect or not connected” text becomes visible.

By clicking the button, user constructs a new JqlQuerySearcherJs instance and executes the “search” method of it with a row limitation of 1. Callback functions are implemented:

- successCallback: a green “correct” string becomes visible above “Test Query” button (see Figure 18)
- errorCallback: a red “incorrect or not connected” string becomes visible
- abortedCallback: a red “unknown” string becomes visible

These strings are technically the content of a div element above “Test Query” button. When callbacks functions are called, the content and class of this div are changed. The class marks the color the text should have.

These strings are put into content of a div element above “Test Connection” button. When content of this element is changed, class of this div is also set. The color of the text is controlled by a CSS depending on this class.

A textfield for setting the refresh interval is also a part of settings dialog. The user can set the length of the period between refreshing list of found issues, given in minutes.

The last textfield on the setting dialog is the limitation of number of rows that can be fetched to the gadget. This limitation can be set for performance purposes. This value effects tempMax parameter of JqlQuerySearcher instance of the main interface of the gadget.

5.3.2 Main interface

The main interface of the gadget is the interface of the gadget when settings dialog is not opened. It consists of a message, a toolbar and the list of issues. It is technically an HTML page. It defines a class with name: QueryFollower. An attribute of the class is an instance of jqlQuerySearcher.

The gadget has an instance of QueryFollower, this object will be referred as queryFollower later. It controls the user interface of the gadget.

The main member functions of QueryFollower class are as follows:

QueryFollower(e)

QueryFollower(e) is constructor for QueryFollower class.

parameter	
e.message	id of a div element, content of it is updated while searching is going on

e.result	id of div element that contains result of searching
e.refreshButton	id of refresh button

Table 9 API documentation of QueryFollower(e)

updateClicked()

updateClicked() method is executed when the user clicks on the refresh button. This function updates the result and sets up timers for the automatic updates and for counting back on the refresh button.

openinJira()

Opens the followed query on searching interface of JIRA.

Figure 19 Main interface of JQL Query Follower gadget with highlighted list of issues

5.3.2.1 Message

The message provides information about the number of found and fetched issues. When searching is going on, the next text is shown: “Loading: x loaded / y found...” where x is the number of fetched issues, y is number of found issues. Once the fetching is finished, only the number of found issues is shown together with the “shown” string.

The message is controlled by action and callback functions of IssueListIterator instance of queryFollower. It will be explained in more details in 5.3.2.2.

5.3.2.2 Toolbar

The toolbar contains two buttons: one that refreshes the result and shows the duration before the next automatic one: the refresh button; and one for opening the followed query on searching interface of JIRA.

By clicking on this button, the user executes updateResult function of queryFollower. The function sets up timers for updating the shown duration on the button every second and for executing an automatic refresh after the set duration is over.

Updating data is implemented using callback function of JqlQuerySearcher instance of queryFollower:

- successCallback: the previously loaded result is removed and IssueListIterator instance of queryFollower is started. This tool iterates through all found issues and loads them to result div of the user interface. Loading data will be explained in more details at 5.3.2.3.
- errorCallback, abortedCallback: gadget shows a message with the next text: “error while searching, please check if settings are correct and query is correct".

```
this.refreshButton = e.refreshButton;
this.loader        = new IssueListIterator({
    action:        function(issueXml) {parent.addRow(issueXml);},
    maxNum:        -1,
    iteratedCallback:  function(loaded,found) {parent.setUiLoaded(loaded,found);},
    finishedCallback:  function(listed,found) {parent.setUiFound(listed,found);}
});
this.jqlQuerySearcher = new JqlQuerySearcher({
    baseUrl:       "-",    //will be set later..
    successCallback:   function(xml){parent.clearResult(); parent.loader.start(xml);},
    errorCallback:     function(url) {parent.setUiError();},
    abortedCallback:   function() {parent.setUiError();}
});
this.scrollPos = 0;    //to store scroll position for restoring it after refresh
```

Figure 20 construction of IssueListIterator and JqlQuerySearcher instances of queryFollower

5.3.2.3 Result

Result contains information about found issues: key, summary, status, priority, type. Each issue is represented with a white rectangle. The rectangle shows the key and summary as texts. Summary is clickable - by clicking on it, a new browser window opens and the issue is opened in JIRA. Status, priority and type are represented by icons. Result is scrollable. Technically, result is a div element that contains other div elements representing issues.

Issues are fetched in result div in addRow function of queryFollower. This function is the implementation of action of IssueListIterator instance of queryFollower. Its parameter is an XML representing one JIRA issue. It is executed for each found issue. The function defines an Issue (see 4.1.4) for reaching data represented by this XML.

```
/** adds 1 row to result*/
this.addRow = function(issueXml) {
    var row = document.createElement("div");
    row.className = "jira_issue";
    var issue = new Issue(issueXml);
    row.innerHTML = "<div>"+issue.fieldText("key")+"</div>";
    row.innerHTML += "<a href='"+issue.fieldText("link")+"' target='_blank'>"+issue.fieldText("summary")+"</a>";
    row.innerHTML += "<img class='statusIcon' src="+issue.fieldAttribute("status","iconUrl")+"></img>";
    row.innerHTML += "<img class='priorityIcon' src="+issue.fieldAttribute("priority","iconUrl")+"></img>";
    row.innerHTML += "<img class='typeIcon' src="+issue.fieldAttribute("type","iconUrl")+"></img>";
    this.result.appendChild(row);
}
```

Figure 21 Implementation of addRow method

6 Conclusion

Each implemented gadget retrieves data from JIRA. The service used for fulfilling this need is searchrequest-xml, a non (or not well) documented service of JIRA. The fact that documentation of this tool is missing or is hard to be found was the biggest

challenge of implementing these tools. After understanding high level requirements, more than a week was spent on finding a usable solution.

There is an official tutorial that provides an example for reaching issue data stored in JIRA (Standalone Gadget Tutorial - Writing a JQL Gadget), but the tutorial is not up-to-date, it does not work in current version of JIRA. After several trial and errors I found Dushan Hanuska's issue listing gadget (JIRA Issues Gadget, 2007).

After understanding the source code of the gadget I was able to start searching for the exact service I need. I found the post "Retrieving And Filtering JIRA Issue Views Through Request URL" (Retrieving and Filtering JIRA issues) that made me able to start implementation.

After I was able to reach issue data stored in JIRA, there was no more serious technical challenge. Development became straight-forward. Most problems that I ran into were related to web development. I easily found solution for these problems by asking on Stack Overflow and by searching, observations, and trial and errors.

I learnt a lot while I implemented these tools. I did not have any previous practical experience about web technologies. By implementing these tools I became confident in using HTML, CSS and JavaScript, that all are important technologies in nowadays' and future's Information Technology.

References

JIRA Issues Gadget. (2007). Retrieved 05 05, 2013, from https://sites.google.com/site/dushanhanuska/: https://sites.google.com/site/dushanhanuska/jiraissuesgadget

About Landis+Gyr. (n.d.). Retrieved 05 05, 2013, from Website of Landis+Gyr Oy: http://www.landisgyr.com/about/

Adding a Custom Field. (n.d.). Retrieved 05 05, 2013, from JIRA documentation: https://confluence.atlassian.com/display/JIRA/Adding+a+Custom+Field

Advanced Searching. (n.d.). Retrieved 05 05, 2013, from JIRA documentation: https://confluence.atlassian.com/display/JIRA/Advanced+Searching

Allow searching for part of a word (prefix / substring searches). (n.d.). Retrieved 05 05, 2013, from JIRA: https://jira.atlassian.com/browse/JRA-6218

Cascadian Style Sheets. (n.d.). Retrieved 05 05, 2013, from Wikipedia: https://en.wikipedia.org/wiki/Cascading_Style_Sheets

Comparison of Web browsers. (n.d.). Retrieved 05 05, 2013, from Wikipedia: http://en.wikipedia.org/wiki/Comparison_of_web_browsers

Customising the Dashboard. (n.d.). Retrieved 05 05, 2013, from JIRA documentation: https://confluence.atlassian.com/display/JIRA/Customising+the+Dashboard

Dashboard (management information systems). (n.d.). Retrieved 05 05, 2013, from Wikipedia: http://en.wikipedia.org/wiki/Dashboard_%28management_information_systems%29

DataTables. (n.d.). Retrieved 05 05, 2013, from DataTables: http://www.datatables.net/

DatePicker. (n.d.). Retrieved 05 05, 2013, from jQuery UI: http://jqueryui.com/datepicker/

Developing a Gadget for Windows Sidebar Part 1: The Basics. (n.d.). Retrieved 05 05, 2013, from Microsoft Developers Network: http://msdn.microsoft.com/en-us/library/windows/desktop/bb456468%28v=vs.85%29.aspx

Developing Standalone or Plugin Gadgets. (n.d.). Retrieved 05 05, 2013, from Atlassian Developers: https://developer.atlassian.com/display/GADGETS/Developing+Standalone+or+Plugin+Gadgets

Fieldset tag. (n.d.). Retrieved 05 05, 2013, from w3schools: http://www.w3schools.com/tags/tag_fieldset.asp

Gadgets XML Reference. (n.d.). Retrieved 05 05, 2013, from Google Developers: https://developers.google.com/gadgets/docs/xml_reference

Getting Started with Gadget Development. (n.d.). Retrieved 05 05, 2013, from Atlassian Developers: https://developer.atlassian.com/display/GADGETS/Getting+Started+with+Gadget+Development

HTML. (n.d.). Retrieved 05 05, 2013, from Mozilla Developers Network: https://developer.mozilla.org/en-US/docs/HTML

HTML element. (n.d.). Retrieved 05 05, 2013, from Wikipedia: http://en.wikipedia.org/wiki/HTML_element

JavaScript. (n.d.). Retrieved 05 05, 2013, from Wikipedia: http://en.wikipedia.org/wiki/JavaScript

JIRA. (n.d.). Retrieved 05 05, 2013, from Wikipedia: http://en.wikipedia.org/wiki/JIRA

jQuery. (n.d.). Retrieved 05 05, 2013, from jQuery: http://jquery.com/

jQuery. (n.d.). Retrieved 05 05, 2013, from Wikipedia: http://en.wikipedia.org/wiki/JQuery

jQuery UI MultiSelect Widget. (n.d.). Retrieved 05 05, 2013, from Eric Hynds' blog: http://www.erichynds.com/blog/jquery-ui-multiselect-widget

NaturalSort.js. (n.d.). Retrieved 05 05, 2013, from Google code: http://js-naturalsort.googlecode.com/svn/trunk/naturalSort.js

Performing Text Searches. (n.d.). Retrieved 05 05, 2013, from JIRA documentation: https://confluence.atlassian.com/display/JIRA/Performing+Text+Searches

Retrieving and Filtering JIRA issues. (n.d.). Retrieved 05 05, 2013, from Pulasthisupun blog: http://pulasthisupun.blogspot.fi/2011/03/retrieving-and-filtering-jira-issue.html

Same Origin Policy. (n.d.). Retrieved 05 05, 2013, from JavaScript.info: http://javascript.info/tutorial/same-origin-security-policy

Scripts. (n.d.). Retrieved 05 05, 2013, from W3C: http://www.w3.org/TR/html4/interact/scripts.html

Sorting plug-ins. (n.d.). Retrieved 05 05, 2013, from DataTables: http://datatables.net/plug-ins/sorting

Specifying Field Behaviour. (n.d.). Retrieved 05 05, 2013, from JIRA documentation: https://confluence.atlassian.com/display/JIRA/Specifying+Field+Behaviour

Standalone Gadget Tutorial - Writing a JQL Gadget. (n.d.). Retrieved 05 07, 2013, from Atlassian Developers: https://developer.atlassian.com/display/JIRADEV/Standalone+Gadget+Tutorial+-+Writing+a+JQL+Gadget

Using Dashboard Gadgets. (n.d.). Retrieved 05 05, 2013, from JIRA documentation: https://confluence.atlassian.com/display/JIRA/Using+Dashboard+Gadgets

What is a Project. (n.d.). Retrieved 05 05, 2013, from JIRA documentation: https://confluence.atlassian.com/display/JIRA/What+is+a+Project

What is natural ordering. (n.d.). Retrieved 05 05, 2013, from StackOverflow: http://stackoverflow.com/questions/5167928/what-is-natural-ordering-when-we-talk-about-sorting

What is Web Technology. (n.d.). Retrieved 05 05, 2013, from Wiki Answers: http://wiki.answers.com/Q/What_is_web_technology

Wildcard character. (n.d.). Retrieved 05 05, 2013, from Wikipedia: http://en.wikipedia.org/wiki/Wildcard_character

XML. (n.d.). Retrieved 05 05, 2013, from Wikipedia: http://en.wikipedia.org/wiki/XML

XMLHttpRequest. (n.d.). Retrieved 05 05, 2013, from W3C: http://www.w3.org/TR/XMLHttpRequest/