Sample Tracking

Search

'Fixing' JIRA's search box

Atlassian, for reasons unknown, have decided that the 'quick search' box should not actually search all the fields. Even worse searching for numbers results it an attempt to open an issue with that ID.

Limited fields

The fields searched are:

-

Underscores

The summary field, which is one of the fields that is searched, contains the BAC ID. This doesn't work though, as when the IDs are in the format giv3_xxx_36020 JIRA doesn't recognize the number separately. Only searches for the whole value, 'giv3_xxx_36020', work while searches for 36020 don't work(infact they fail for two reasons, any number is assumed to be a JIRA ID and so searching for 36020 results in a page saying the issue doesn't exist).

If we modify the spec so that Summary uses spaces in place of underscores, e.g. 'giv3 xxx 36020', then searching by DB, Collection Code and BAC ID will all work without further modifications. BAC ID will still require quotes around the number to stop JIRA attempting to open the Issue with the same number. Sample ID can be left with the underscores if preferred, although it can't be searched via the Quick Search box.

Through a custom FieldIndexer the value can be stored with and without underscores.

Copying into other fields

The environment field is on the list of searched fields, but we don't use it. To make it into a hidden search fixer would require:

- Hiding it, but still setting its value
  - Hiding while editing can be done with a Javascript field hijacking it
  - Hiding while viewing the Issue will have to be done via the 'Screen' configuration
- Copying the values of all the other fields into the environment field. This only needs to happen when changes are made. Easy for manual changes, harder for automatic updates. Options are:
  - Javascript - Manual changes only
  - Workflow plug-in - editing can be made a workflow task by only allowing edits in transition screens
- A Custom Field Indexer can used to ensure an action is carried out whenever the values may have been updated. If the indexer could write to one of the fields used in quick searches then it would be indistinguishable from that field being added to the quick search.

Jump to Issue

The 'Quick Search' box can be used to go directly to an Issue. Unfortunately this only works with JIRA's own IDs. Searching for the ID first needs to be fixed (see above). Once the Issue can be found, using quick search, then the following modification will redirect the browser from the search page to the browse Issue page if a search returns only one Issue. This has the following limitations:

- It effects all projects
- It requires two hits on the server (probably not significant)
- If the user is actually searching / building a query then being redirected to browse Issue may be annoying.

The changes need to be made to:

```
/atlassian-jira/secure/views/navigator/navigator.jsp
```

The following code should be inserted between

```
<%@ taglib uri="webwork" prefix="webwork" %>
```

and

```
<HTML>
```

at the top of the file. The added code is evaluated after the search has been carried out but before the results have been output.

Code

```
<%-- start of section added to allow simple search to return a bug using other key
fields--%>

<%@ page import="com.atlassian.jira.web.action.issue.IssueSearchResultsAction" %>
<%@ page import="com.atlassian.jira.issue.search.SearchException" %>
<%@ page import="com.atlassian.jira.issue.search.SearchResults" %>
<%@ page import="com.atlassian.jira.issue.Issue" %>
<%@ page import="java.util.List" %>

<webwork:if test="/searchResults/total == 1">
<webwork:property value="/" id="issueSearchResultsAction" />
<%
    IssueSearchResultsAction issueSearchResultsAction =
            (IssueSearchResultsAction)
request.getAttribute("issueSearchResultsAction");
    try {
      if (issueSearchResultsAction != null) {
          //throws SearchException
          SearchResults searchResults =
issueSearchResultsAction.getSearchResults();
          if (searchResults != null) {
            //getIssues only returns the Issues for the current page. i.e.
            //<#issues> mod <#issues per page>. Just getting one issue back,
            //from this method, therefor doesn't prove that only one issue
            //matched. We also have to check that only one page of issues exists.
            // i.e. floor(<#issues> / <#issues per page>) + 1 == 1
            List pages = searchResults.getPages();
            if (pages != null && pages.size() == 1) {
                //only one page. If that one page only contains 1 issue we know
                //that the search was for a unique value.
                List<Issue> issues = searchResults.getIssues();
                if (issues != null && issues.size() == 1) {
                  Issue issue = issues.get(0);
                  String key = issue.getKey();
                  response.sendRedirect(request.getContextPath()+"/browse/"+key);
                } //end if 1 issue
            } //end if 1 page
          } //end if searchResults
      } //end if issueSearchResultsAction
    } catch (SearchException e) {
      //ignore our added bit and hope that the main page will show an error message
    }
%>
</webwork:if>
<%-- end--%>
```

Improvements

- A test of which project is in use
- Intercepting the action instead of modifying the JSP
  - Modify the query to wrap numbers in "" to avoid being interpreted as project IDs
    - Possibly try twice, first go as an ID and then as a number if that failed.
  - If the query is being modified, could the fields be explicitly named?

    Quick Search doesn't allow the 'advanced' queries that are required to use field names.
- Packaging as a plug-in with config of which fields and projects to do
  - Intercept Issue creation to carryout the copying?

Code Path for QuickSearch's Action

QuickSearch

Has quite a deep ancestry. It's family tree is given below, the bracketed and italicized names are the interfaces added with that class, although not necessarily implemented by it.

- JiraWebActionSupport (*ErrorCollection*)
- JiraActionSupport (*I18nHelper, CommandDriven*)
- ActionSupport (*IllegalArgumentAware, Serilizable, **Action***)

The Action methods implemented in Quick Search are doExecute and a bean method setSearchString. The bean method is set to the value of 'searchString' in the query, if it is present.

doExecute

The doExecute method is the equivalent of the main method in an application.

- Attempts to parse the search string as an issue key, via getKey
- If that fails It uses a QueryCreator to convert the search string into a URL that the request is forwarded to.

getKey

This method attempts to parse the search string as an issue key.

It first tests if the string is already valid by calling JiraKeyUtilities.validIssueKey. If it is then the search string is passed back.

If the search string isn't a valid key then an attempt is made to parse it as a number and to then add the project prefix. If parsing as a number didn't fail then getSearchProject is called in an attempt to determine what project is being worked on. If a project is returned then the key of the search project, and if necessary a hyphen, are pre-pended to the search string to form an issue key.

If none of the attempts worked then null is returned.

getSearchProject

getSelectedProjectObject on the super class JiraWebActionSupport is tried first. That method uses the UserProjectHistoryManager to attempt to find the most recently used project that the user still has permission to browse.

If the first attempt fails then getBrowsableProjects on ProjectActionSupport is called which uses the permission manager to list all of the projects that the user has permission to browse. If the user only has one project that they could be using then that project is selected.

If neither of these work then null is returned.

JiraKeyUtils

This class contains methods that can be used to parsing a string to determine if it's format is correct to be used as an Issue key.

isValidIssueKey

QuickSearch calls this method from its getKey method. The method tests:

- if the key is null then null is returned
- if the key contains a hyphen (xxxx-yyyy) then
    - The first part of the key is tested to see if it is a valid project code, using JiraKeyUtils.validProjectKey

    - The second part of the key is tested to see if it has at least 1 character and that all of the characters are numbers.
- if the key passes both tests then it is returned, otherwise null is returned.

validProjectKey

This delegates to a KeyMatcher calling the isValidProjectKey method

ProductionKeyMatcher?

isValidProjectKey

The key is tested to ensure it is not null or empty.

The key is then tested against the regular expression defined in the application properties' property jira.projectkey.pattern. The default for this is ([A-Z][A-Z]+) which ensures that the key contains only capital letters and has at least two characters.

DefaultQueryCreator

This class is an implementation of QueryCreator. It appears to have been registered with the dependency injection framework as it appears as part of the constructor for the Quick Search Action. Its purpose appears to be to convert the string entered into the quick search box into a series of parameters that are passed on to Issue Navigator's Action handler. I expect the format is mimicking that used by the 'custom' searcher's in the Issue Navigator's page.

QuickSearchHandlers

Contains a hard-coded collection of QuickSearchHandler objects. The search string is passed to these handlers in turn, with each removing any of the terms that it is able to convert into part of the QuickSeachResult (that will later become the query string for the search URL). The handlers are all extensions of other, abstract, handlers. The name in brackets after the searcher's class indicates which searcher it extends.

- ProjectQuickSearchHandler (Single Word)
    - getProjectByKey using <word>
    - getProjectByName using <word>
    - adds "pid" = project.id
- IssueTypeQuickSearchHandler (Single Word)
    - get IssueType from <word>
    - try with 'S'/'s' removed from <word>
    - adds "type" = type.id
- StatusQuickSearchHandler (Single Word)
    - get Status
    - adds "status" = status.id
- FixForQuickSearchHandler (Version)
    - get versions prefixed by ff:
    - adds "fixfor" = version.id
- RaisedInVersionQuickSearchHandler (Version)
    - get versions prefixed by v:
    - adds "version" = version.id
- ComponentQuickSearchHandler (Prefixed Single Word)
    - get components prefixed by c:
    - adds "component" = component.id
- MyIssuesQuickSearchHandler (Single Word)
    - #can't see source
- OverdueQuickSearchHandler (Single Word)
    - #can't see source

- ResolutionQuickSearchHandler (Single Word)
  - getResoultionsByName using <word>
  - adds "resolution" = resolution.id
  - if <word> = 'unresolved"
  - adds "resolution" = -1
- PriorityQuickSearchHandler (Single Word)
  - getPriorities and filter by <word>
  - adds "priority" = status.id
- CreatedQuickSearchHandler (Date)
  - dates prefixed by "created:"
  - adds "created" = date
- UpdatedQuickSearchHandler (Date)
  - dates prefixed by "updated:"
  - adds "updated" = date
- DueDateQuickSearchHandler (Date)
  - dates prefixed by "due:"
  - adds "duedate" = date

The abstract classes that these are based on described further below.

The query handlers are private, final and lack any accessor methods. If they could be modified they would provide a very good method of extending the functionality of the quick search.

createQuery

This is the method that creates the QuickSearchResult object, passes it and the query to the QuickSearchHandlers and produces the final URL. Beyond the handler based conversion of the string it also:

Returns a pre-set URL if there is no query (IssueNavigator.jspa?mode=show)

If there are still words in the query string that no handler has removed then these are added to the URL as the parameter 'query'

If one or more words were processed and removed by the handlers two extra parameters are added: 'usedQuickSearch' which is always set to 'true' and 'originalQuickSearchQuery' which is set to the search string as it was before the handlers modified it. These are used to provide the option in the navigator of using the whole string that was put in the quick search box for a text search.

QuickSearchResult

This is a container for two items, the current state of the query string while it is being modified by the handlers and the URL parameters that have been generated by the handlers.

It also contains a method 'getQueryString' which converts the map of URL parameters into a query string, for inclusion by DefaultQueryCreator.createQuery

SingleWordQuickSearchHandler

Splits the string based on spaces. passes the word to the implementation's 'handleWord' method. If that method returns a Map then:

- The word is removed from the search
- The QuickSearchResults object is updated with the contents of the returned map.

PrefixedSingleWordQuickSearchHandler (Single Word)

Implements handleWord, passing a subset of the words to the implementation's handleWordSuffix method. The filtering is done by testing if the words prefix matches that returned by the implementations getPrefix method. Before passing the word to the handleWordSuffix method the

first prefix.length characters are removed.

VersionQuickSearchHander (Prefixed Single Word)

- Uses a ProjectAwareQuickSearchHandler to get any project values from the QuickSearchResult object.
- For each project the versions are retrieved.
- For each version its name is compared to the suffix.
    - If it matches then the value from the implementations getSearchParamName is used to add version.id to the SearchResults.

DateQuickSearchHandler (Single Word)

Checks for the prefix from the implementation's getPrefix method. Strips the prefix. Maps

today -> <param name>:after:=<today's date>
<param name>:before:=<tomorrow's date>
yesterday -> <param name>:after:=<yesterday's date>
<param name>:before:=<today's date>
tomorrow -> <param name>:after:=<tomorrow's date>
<param name>:before:=<(tomorrow + 1)'s date>

If it contains a comma then

xxxx,yyyy -> <param name>:previous:=xxxx
<param name>:next:=yyyy

otherwise

xxxx -> <param name>:previous:=xxxx
(param name comes from the implementation's getSearchParamName method.)

Notes:

A class that replaces QuickSearch will have to duplicate quite a bit of the code owing to hard-coding and private methods. Seemingly the most sensible place to make a modification would be by adding to the QuickSearch handlers but this would require replicating much of DefaultQueryCreator as the list of handlers is hard-coded and cannot be modified from outside the class.

The main problem is in the implementation of getKey which returns a value if the search string can be parsed as a number even if no valid project is associated with it.

Passing the number through to a full text search would allow bac-id's to be used in the search box

If no project was specified and the most recent project was Sample Tracking then se a full text search.

Adding a quick search handler for bac:xxxxx would be familiar and unlikely to affect any other user group.

Modifying the search form to submit to a new action

The javascript to modify the url a form submits to is below:

```
<script type="text/javascript">
jQuery(document).ready(function() {
      var theForm=document.getElementById("theForm");
      theForm.action = newtarget;
});
</script>
```

The HTML comes from WEB-INF/classes/templates/plugins/topnavigation/topnav.vm
(The parts in italics are the values that appear in the rendered HTML).

```
 #if ($hasAnyProjects)
<form id="quicksearch" action="${baseurl}/secure/QuickSearch.jspa" method="post">
    <label for="quickSearchInput" class="overlabel">
        $i18n.getText('alt.text.quicksearch')
        Quick Search
        <span id="quicksearchhelp" class="informationBox">
            #set ($helpURL = "<a href='$quickSearchHelpPath.url' target='_jirahelp'>")
                        $i18n.getText('alt.text.quicksearch.learn.more', $helpURL, "</a>")
            Learn more about
            <a target = '_jirahelp'
                 href = 'http://docs.atlassian.com/jira/docs-040/Using+Quick+Search?clicked=jirahelp'>
                            Quick Search
            </a>
        </span>
    </label>
    <input    id = "quickSearchInput"
            class = "quickSearchInput"
            title = "$i18n.getText('alt.text.quicksearch.title')"
                    "Go directly to an issue by typing a valid issue key, or run a free-text search."
             type = "text"
             size = "25"
             name = "searchString"
        accessKey = "q"
           valign = "absmiddle" />
</form>
#end
```

Bold = in both the template and the HTML produced
Plain = only in the template
Italic = only in the resultant HTML

The form's ID is 'quicksearch', which seems likely to be fairly stable through version changes. None of the Javascript in the page references it.

includes/navigator/quick-search-reverse.jsp - refers to quick search. It is included in 'includes/navigator/table.jsp' and 'secure/views/navigator/advancedsearch.jsp'. It appears to be used to place a message if the search has been modified by quick search and to offer to carry out the original search.

includes/js/jira-global.js - includes adding a tool tip to quicksearch

com.atlassian.auiplugin:ajs.js contains a function called 'autocomplete' which may be associated with the field.

Searching from within a Plug-in

```
public Collection getRecentlyUpdatedIssues(Browser browser) {
    List issues = null;
    try {
        // Create a Search Request to get the list of issues
        SearchRequest searchRequest = new SearchRequest(browser.getRemoteUser());

        // Only get issues for the current project
        searchRequest.addParameter(new ProjectParameter(browser.getSelectedProject().getLong("id")));

        // Only get issues updated in the last week
        Date startDate = new Date(System.currentTimeMillis() - ONE_WEEK);
        searchRequest.addParameter(
               new AbsoluteDateRangeParameter(DocumentConstants.ISSUE_UPDATED, startDate, null));

        // Sort the issues by issue updated time
        searchRequest.addSearchSort(
               new SearchSort(NavigableField.ORDER_DESCENDING, IssueFieldConstants.UPDATED));

        // Do the search, return the list of issues.
        issues = searchProvider.search(searchRequest, browser.getRemoteUser(),PAGER_FILTER).getIssues();

    } catch (Exception e) {
        log.error("Error finding recently updated issues: " + e, e);
    }

    return issues;
}
```

The above way of searching is limited to the current user. For a plug-in that waits for all Issues in a batch to reach a certain stage the simplest search would be "are their any Issues in this batch and not at the required step?" The above method of querying would actually query for "are their any Issues in this batch and not at the required step and that I can edit?" We can't tell what state the Issues the user can't edit are in.

To avoid it being restricted by the user's permissions the query needs to be carried out at a lower level:

```
//Create a query builder
JqlQueryBuilder queryBuilder = JqlQueryBuilder.newBuilder();

//get a clause builder
//the defaultAnd method gets a builder that doesn't need to be
//passed query seperators, as though they are being parsed.
//It just and's everything
JqlClauseBuilder whereBuilder = queryBuilder.where().defaultAnd();

whereBuilder.issueType(issueType.getString("id"));
whereBuilder.project(getProjectId());

//avoid being limited by the user, as we want to know if any exist, not if the user can see them.
searchProvider.searchCountOverrideSecurity(queryBuilder.buildQuery(), getRemoteUser());
```