The MKWS manual: embedded metasearching with the MasterKey Widget Set

Mike Taylor

Introduction

There are lots of practical problems in building resource discovery solutions. One of the biggest, and most ubiquitous is incorporating metasearching functionality into existing web-sites – for example, content-management systems, library catalogues or intranets. In general, even when access to core metasearching functionality is provided by simple web-services such as Pazpar2, integration work is seen as a major part of most projects.

Index Data provides several different toolkits for communicating with its metasearching middleware, trading off varying degrees of flexibility against convenience:

- pz2.js – a low-level JavaScript library for interrogating the Service Proxy and Pazpar2. It allows the HTML/JavaScript programmer to create JavaScript applications to display facets, records, etc. that are fetched from the metasearching middleware.
- masterkey-ui-core – a higher-level, complex JavaScript library that uses pz2.js to provide the pieces needed for building a full-featured JavaScript application.
- MasterKey Demo UI – an example of a searching application built on top of masterkey-ui-core. Available as a public demo at `http://mk2.indexdata.com/`
- MKDru – a toolkit for embedding MasterKey-like searching into Drupal sites.

All but the last of these approaches require programming to a greater or lesser extent. Against this backdrop, we introduce MKWS (the MasterKey Widget Set) – a set of simple, very high-level HTML+CSS+JavaScript components that can be incorporated into any web-site to provide MasterKey searching facilities. By placing `<div>`s with well-known MKWS classes in any HTML page, the various components of an application can be embedded: search-boxes, results areas, target information, etc.

Simple example

The following is a complete MKWS-based searching application:

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>MKWS demo client</title>
    <script type="text/javascript" src="//mkws.indexdata.com/mkws-complete.js"></script>
    <link rel="stylesheet" href="//mkws.indexdata.com/mkws.css" />
  </head>
  <body>
    <div class="mkws-search"></div>
    <div class="mkws-results"></div>
  </body>
</html>
```

Go ahead, try it! Simply put the above in a file (e.g. index.html), drop it into a folder accessible with an ordinary web-server (e.g. Apache HTTP Server) and load it in your web browser. Just like that, you have working metasearching.

How the example works

If you know any HTML, the structure of the file will be familiar to you: the `<html>` element at the top level contains a `<head>` and a `<body>`. In addition to whatever else you might want to put on your page, you can add MKWS elements.

These fall into two categories. First, the prerequisites in the HTML header, which are loaded from the tool site `mkws.indexdata.com`:

- `mkws-complete.js` contains all the JavaScript needed by the widget-set, including a copy of the jQuery library.
- `mkws.css` provides the default CSS styling.

Second, within the HTML body, `<div>` elements with special classes that begin `mkws-` can be provided. These are filled in by the MKWS code, and provide the components of the searching UI. The very simple application above has only two such widgets: a search box and a results area. But more are supported.

Defining widget elements

Widget type

An HTML element is made into an MKWS widget by including an MKWS class-name. These names begin `mkws-` and what follows that prefix specifies the type of the widget. The type can be any sequence of alphanumeric characters and hyphens *except* something beginning `team` – see below.

The main widgets are:

- `mkws-search` – provides the search box and button.
- `mkws-results` – provides the results area, including a list of brief records (which open out into full versions when clicked), paging for large results sets, facets for refining a search, sorting facilities, etc.
- `mkws-progress` – shows a progress bar indicating how many of the targets have responded to the search request.
- `mkws-stat` – provides a status line summarising the statistics of the various targets.
- `mkws-switch` – provides links to switch between a view of the result records and of the targets that provide them. Only meaningful when `mkws-targets` is also provided.
- `mkws-targets` – the area where per-target information will appear when selected by the link in the `mkws-switch` area. Of interest mostly for fault diagnosis rather than for end-users.
- `mkws-lang` – provides links to switch between one of several different UI languages. By default, English, Danish and German are provided.

To see all of these working together, just put them all into the HTML `<body>` like so:

```
<div class="mkws-switch"></div>
<div class="mkws-lang"></div>
<div class="mkws-progress"></div>
<div class="mkws-search"></div>
<div class="mkws-results"></div>
<div class="mkws-targets"></div>
<div class="mkws-stat"></div>
```

The full set of supported widgets is described in the reference guide below.

Widget team

In general a set of widgets work together in a team: in the example above, the search-term that the user enters in the `mkws-search` widget is used to generate the set of records that are displayed in the `mkws-results` widget.

Sometimes, it's desirable to have multiple teams in a single page. A widget can be placed in a named team by giving it (in addition to its main class) a class that begins with `mkws-team-` and what follows that prefix specifies the team that the widget is part of. For example, `<div class="mkws-search mkws-team-aux">` creates a search widget that is part of the `aux` team.

Widgets that do not have a team specified (as in the examples above) are placed in the team called `AUTO`.

Configuring widgets

Global configuration

Many aspects of the behaviour of MKWS can be modified by setting parameters into the `mkws_config` object. So the HTML header looks like this:

```
<script type="text/javascript">
  var mkws_config = {
    lang_options: [ "en", "da" ]
    lang: "da",
    sort_default: "title",
  };
</script>
<script type="text/javascript" src="http://mkws.indexdata.com/mkws-complete.js"></script>
```

This configuration restricts the set of available UI languages English and Danish (omitting German), sets the default to Danish (rather than the English), and initially sorts search results by title rather than relevance (though as always this can be changed in the UI).

The full set of supported configuration settings is described in the reference guide below.

Per-widget configuration

In addition to the global configuration provided by the `mkws_config` object, individual widgets' behaviour can be configured by providing configuration settings as attributes on their HTML elements. For example, a `records` widget

might be restricted to displaying no more than three records by setting the `maxrecs` parameter as follows:

```
<div class="mkws-records" maxrecs="3">
```

Although this works well, HTML validators will consider this element unacceptable, since the `maxrecs` attribute is not part of the HTML schema. However, attributes beginning `data-` are always accepted as HTML extensions, much like email headers beginning with `X-`. Therefore, the widget set also recognises configuration attributes prefixed with `data-mkws-`, so:

```
<div class="mkws-records" data-mkws-maxrecs="3">
```

The first form is more convenient; the second is more correct.

Because some configuration settings take structured values rather than simple strings, they cannot be directly provided by inline attributes. To allow for this, the special attribute `data-mkws-config`, if provided, is parsed as JSON and its key-value pairs used as configuration settings for the widget in question. For example, the value of `lang_options` is an array of strings specifying which of the supported UI languages should be made available. The following invocation will limit this list to only English and Danish (omitting German):

```
<div class="mkws-lang" data-mkws-config='{ "lang_options": [ "en", "da" ] }'></div>
```

(Note that, as JSON requires double quotes around all strings, single quotes must be used to contain the entire attribute value.)

Control over HTML and CSS

More sophisticated applications will not simply place the widgets together, but position them carefully within an existing page framework – such as a Drupal template, an OPAC or a SharePoint page.

While it's convenient for simple applications to use a monolithic `mkws-results` area which contains record, facets, sorting options, etc., customised layouts may wish to treat each of these components separately. In this case, `mkws-results` can be omitted, and the following lower-level widgets provided instead:

- `mkws-facets` – provides the facets
- `mkws-ranking` – provides the options for how records are sorted and how many are included on each page of results.

- `mkws-pager` – provides the links for navigating back and forth through the pages of records.
- `mkws-navi` – when a search result has been narrowed by one or more facets, this area shows the names of those facets, and allows the selected values to be clicked in order to remove them.
- `mkws-records` – lists the actual result records.

Customisation of MKWS searching widgets can also be achieved by overriding the styles set in the toolkit's CSS stylesheet. The default styles can be inspected in mkws.css and overridden by any styles that appear later in the HTML. At the simplest level, this might just mean changing fonts, sizes and colours, but more fundamental changes are also possible.

To properly apply styles, it's necessary to understand how the HTML is structured, e.g. which elements are nested within which containers. The structures used by the widget-set are described in the reference guide below.

Customised display using Handlebars templates

A lot can be done by styling widgets in CSS and changing basic MKWS config options. For further customisation, MKWS allows you to change the markup it outputs for any widget. This is done by overriding the Handlebars template used to generate it. In general these consist of `{{things in double braces}}` that are replaced by values from the system. For details of Handlebars template syntax, see the online documentation.

The templates used by the core widgets can be viewed in our git repository. Parameters are documented in a comment at the top of each template so you can see what's going where. If all you want to do is add a CSS class to something or change a `span` to a `div` it's easy to just copy the existing template and make your edits.

Overriding templates

To override the template for a widget, include it inline in the document as a `<script>` tag marked with a class of `mkws-template-foo` where foo is the name of the template you want to override (typically the name of the widget). Inline Handlebars templates are distinguished from JavaScript via a `type="text/x-handlebars-template"` attribute. For example, to override the pager template you would include this in your document:

```
<script class="mkws-template-pager" type="text/x-handlebars-template">
  ...new Pager template
</script>
```

The Facet template has a special feature where you can override it on a per-facet basis by adding a dash and the facet name as a suffix e.g. `facet-subjects`. (So `class="mkws-template-facet-subjects"`.) When rendering a facet for which no specific template is defined, the code falls back to using the generic facet template, just called `facet`.

You can also explicitly specify a different template for a particular instance of a widget by providing the name of your alternative (e.g. `special-pager`) as the value of the `template` key in the MKWS config object for that widget: for example, `<div class="mkws-pager" template="special-pager"/>`.

Templates for MKWS can also be precompiled. If a precompiled template of the same name is found in the `Handlebars.templates` object, it will be used instead of the default.

Inspecting metadata for templating

MKWS makes requests to the Service Proxy or Pazpar2 that perform the actual searching. Depending on how these are configured and what is available from the targets you are searching, there may be more data available than what is presented by the default templates.

Handlebars offers a convenient log helper that will output the contents of a variable for you to inspect. This lets you look at exactly what is being returned by the back-end without needing to use a JavaScript debugger. For example, you might prepend `{{log hits}}` to the Records template in order to see what is being returned with each search result in the list. In order for this to work, you will need to enable verbose output from Handlebars which is done by including this line or similar:

```
<script>Handlebars.logger.level = 1;</script>
```

Internationalisation

If you would like your template to use the built-in translation functionality, output locale specific text via the mkws-translate helper like so: `{{{mkws-translate "a few words"}}}`.

Example

Rather than use the toolkit's included AJAX helpers to render record details inline, here's a summary template that will link directly to the source via the address provided in the metadata as the first element of `md-electronic-url`:

```
<script class="mkws-template-summary" type="text/x-handlebars-template">
  <a href="{{md-electronic-url.[0]}}">
    <b>{{md-title}}</b>
  </a>
  {{#if md-title-remainder}}
    <span>{{md-title-remainder}}</span>
  {{/if}}
  {{#if md-title-responsibility}}
    <span><i>{{md-title-responsibility}}</i></span>
  {{/if}}
</script>
```

For a more involved example where markup for multiple widgets is decorated with Bootstrap classes and a custom Handlebars helper is employed, take a look at the source of topic.html.

Some Refinements

Message of the day

Some applications might like to open with content in the area that will subsequently be filled with result-records – a message of the day, a welcome message or a help page. This can be done by placing an `mkws-motd` division anywhere on the page. It will initially be moved into the `mkws-results` area and displayed, but will be hidden as soon as the first search is made.

Popup results with jQuery UI

The jQuery UI library can be used to construct MKWS applications in which the only widget generally visible on the page is a search box, and the results appear in a popup. The key part of such an application is this invocation of the MKWS jQuery plugin:

```
<div class="mkws-search"></div>
<div class="mkws-popup" popup_width="1024" popup_height="650">
  <div class="mkws-results"></div>
</div>
```

The necessary scaffolding can be seen in an example application, popup.html.

The relevant properties (`popup_width`, etc.) are documented below in the reference section.

MKWS target selection

Introduction

MKWS accesses targets using the Pazpar2 metasearching engine. Although Pazpar2 can be used directly, using a statically configured set of targets, this usage is unusual. More often, Pazpar2 is fronted by the Service Proxy (SP), which manages authentication, sessions, target selection, etc. This document assumes the SP is used, and explains how to go about making a set of targets (a "library") available, how to connect your MKWS application to that library, and how to choose which of the available targets to use.

By default MKWS configures itself to use an account on a service hosted by `sp-mkws.indexdata.com`. By default, it sends no authentication credentials, allowing the appropriate account to be selected on the basis of referring URL or IP address.

If no account has been set up to recognise the referring URL of the application or the IP address of the client, then a default "MKWS Demo" account is used. This account (which can also be explicitly chosen by using the username `mkws`, password `mkws`) provides access to about a dozen free data sources.

In order to search in a customised set of targets, including subscription resources, it's necessary to create an account with Index Data's hosted Service Proxy, and protect that account with authentication tokens (to prevent unauthorised use of subscription resources).

Maintaining the library

The Service Proxy accesses sets of targets that are known as "libraries". In general, each customer will have their own library, though some standard libraries may be shared between many customers – for example, a library containing all open-access academic journals. A library can also contain other configuration information, including the set of categories by which targets are classified for the library.

Libraries are maintained using MKAdmin (MasterKey Admin). Specifically, those used by MKWS are generally maintained on the "MKX Admin" installation at `http://mkx-admin.indexdata.com/console/` In general, Index Data will create a library for each customer, then give the customer a username/password pair that they can use to enter MKAdmin and administrate that library.

Once logged in, customers can select which targets to include (from the list of several thousand that MKAdmin knows about), and make customer-specific modifications to the target profiles – e.g. overriding the titles of the targets.

Most importantly, customers' administrators can add authentication credentials that the Service Proxy will use on their behalf when accessing subscription

resources – username/password pairs or proxies to use for IP-based authentication. Note that **it is then crucial to secure the library from use by unauthorised clients**, otherwise the customer's paid subscriptions will be exploited.

Access to libraries is managed by creating one or more "User Access" records in MKAdmin, under the tab of that name. Each of these records provides a combination of credentials and other data that allow an incoming MKWS client to be identified as having legitimate access to the library. The authentication process, described below, works by searching for a matching User Access record.

Authenticating your MWKS application onto the library

Some MKWS applications will be content to use the default library with its selection of targets. Most, though, will want to define their own library providing a different range of available targets. An important case is that of applications that authenticate onto subscription resources by means of back-end site credentials stored in MKAdmin: precautions must be taken so that such library accounts do not allow unauthorised access.

Setting up such a library is a process of several stages.

Create the User Access account

Log in to MKAdmin to add a User Access account for your library:

- Go to `http://mkx-admin.indexdata.com/console/`
- Enter the administrative username/password
- Go to the User Access tab
- Create an end-user account
- Depending on what authentication method is to be used, set the User Access account's username and password, or referring URL, or IP-address range.

If your MWKS application runs at a well-known, permanent address – `http://yourname.com/app.html`, say – you can set the User Access record so that this originating URL is recognised by setting it into the "Referring URL" field. Then the application will always use the library that this User Access record is associated with (unless it sends a username/password pair to override this default).

Or if your application's users are coming from a well-known range of IP-address space, you can enter the range in the "IP Ranges" field. The format of this

field is as follows: it can contain any number of ranges, separated by commas; each range is either a single IP address or two addresses separated by a hyphen; each IP address is four small integers separated by periods. For example, `80.229.143.255-80.229.143.255, 5.57.0.0-5.57.255.255, 127.0.0.1`.

Alternatively, your application can authenticate by username and password credentials. This is a useful approach in several situations, including when you need to specify the use of a different library from the usual one. To arrange for this, set the username and password as a single string separated by a slash – e.g. `mike/swordfish` – into the User Access record's Authentication field.

You can set multiple fields into a single User Access record, or create multiple User Access records. For example, a single User Access record can specify both a Referring URL and a username/password pair that can be used when running an application from a different URL. But if multiple Referring URLs are needed, then each must be specified in its own User Access record.

(Optional): embed credentials for access to the library

When credential-based authentication is in use (username and password), it's necessary to pass these credentials into the Service Proxy when establishing the session. This is done by providing the `sp_auth_credentials` configuration setting as a string containing the username and password separated by a slash:

```
mkws_config = { sp_auth_credentials: "mike/swordfish" };
```

(Optional): conceal credentials from HTML source

Using credential-based authentication settings such as those above reveals the credentials to public view – to anyone who does View Source on the MKWS application. This may be acceptable for some libraries, but is intolerable for those which provide authenticated access to subscription resources.

In these circumstances, a different approach is necessary. Referer-based or IP-based authentication may be appropriate. But if these are not possible, then a more elaborate approach can be used to hide the credentials in a web-server configuration that is not visible to users.

The idea is to make a Service Proxy authentication URL local to the customer, hiding the credentials in a rewrite rule in the local web-server's configuration. Then local mechanisms can be used to limit access to that local authentication URL. Here is one way to do it when Apache2 is the application's web-server, which we will call `yourname.com`:

Step 1: add a rewriting authentication alias to the configuration:

```
RewriteEngine on
RewriteRule /spauth/ http://sp-mkws.indexdata.com/service-proxy/\
    ?command=auth&action=check,login&username=U&password=PW [P]
```

Step 2: set the MKWS configuration setting `service_proxy_auth` to `http://yourname.com/spauth/`.

Step 3: protect access to the local path `http://yourname.com/spauth/` (e.g. using a `.htaccess` file).

Choosing targets from the library

MKWS applications can choose what subset of the library's targets to use, by means of several alternative settings on individual widgets or in the `mkws_config` structure:

- `targets` – contains a Pazpar2 targets string, typically of the form "pz:id=" or "pz:id~" followed by a pipe-separated list of low-level target IDs. At present, these IDs can take one of two forms, depending on the configuration of the Service Proxy being used: they may be based on ZURLs (so a typical value would be something like `pz:id=josiah.brown.edu:210/innopac|lui.indexdata.com:8080/solr4/select?fq=database:4902` or they may be UDBs (so a typical value would be something like `pz:id=brown|artstor`)
- `targetfilter` – contains a CQL query which is used to find relevant targets from the relvant library. For example, `udb==Google_Images` or `categories=news`
- `target` – contains a single UDB: that of the sole target to be used. For example, `Google_Images`. This is merely syntactic sugar for "targetfilter" with the query `udb==NAME`

For example, a `Records` widget can be limited to searching only in targets that have been categorised as news sources by providing an attribute as follows:

```
<div class="mkws-records" targetfilter='categories=news'/>
```

Reference guide

Widgets

The following widgets are provided in the core set. (Others can be added: see the MKWS developers' guide.)

Name	Description
`auth-name`	Initially empty, it updates itself to shows the name of the library that the application is logged in as when authentication is complete.
`builder`	A button which, when pressed, analyses the current settings of the team that it is a part of, and generates the HTML for an auto-searching element that will replicate the present search. This HTML is displayed in an alert box: it is intended that this widget be subclassed to store the generated widget definitions in more useful places.
`button`	The search button. Usually generated by a

Configuration settings

Configuration settings may be provided at the level of a individual widget, or a team, or globally. Per-widget configuration is described above; per-team settings can be placed in a `config` widget belonging to the relevant team, and will be applied to that team as a whole; and global settings are provided in the global variable `mkws_config`. This structure is a key-value lookup table, and may specify the values of many settings.

Some settings apply only to specific widgets; others to the behaviour of the toolkit as a whole. When a widget does not itself have a value specified for a particular configuration setting, its team is consulted; and if that also does not have a value, the global settings are consulted. Only if this, too, is unspecified, is the default value used.

The supported configuration settings are described in the table below. For those settings that apply only to particular widgets, the relevant widgets are listed. All entries are optional, but if specified must be given values of the specified type. Long default values are in footnotes to keep the table reasonably narrow.

Setting	Widget	Type	Default	Description
autosearch	facet, facets, record, records, results	string		If provided, this setting contains a query which is immediately run on behalf of the team. Often used with an indirect setting.
facet	facet	string		For a `facet` widget, this setting is mandatory, and indicates which field to list terms for. Three fields are supported: `subject`, `author`, and `xtargets` – the latter is a special case which treats the target (that is providing a record) as a facet. Any other field may also be used, but the default caption and maximum term-count may not be appropriate, needing to be overridden by `facet_caption_*` and `facet_max_*` settings.
facet_caption_*	facet	string		Specifies what on-screen caption is to be used for

The `show_lang`, `show_perpage`, `show_sort`, and `show_switch` configuration settings are technically redundant, as the relevant widgets, like all widgets, are displayed only when they are provided. But they are retained as an easier route to lightly customise the display, than by providing a full HTML structure.

Notes

1. The default for `facets` is `["xtargets", "subject", "author"]`
2. The default for `perpage_options` is `[10, 20, 30, 50]`
3. The default for `pp2_hostname` is `"sp-mkws.indexdata.com"`
4. The default for `pp2_path` is `"service-proxy/"`
5. The default for `sort_options` is `[["relevance"], ["title:1", "title"], ["date:0", "newest"], ["date:1", "oldest"]]`
6. The default for `sp_auth_query` is `"command=auth&action=perconfig"`

Indirect settings

The values of any setting are generally interpreted literally. However, it is possible to specify a value indirectly – for example, by reference to a query parameter – and this is often useful in contexts such as specifying an autosearch query. Settings of this kind have values beginning with an exclamation mark, and take the form `!`*type*`!`*value*.

The currently supported types are:

- `param` – uses the value of the specified query parameter for the URL. For example `<div class="mkws-results" autosearch="!param!term">` will auto-search for the word "sushi" if the page containing that widget is invoked from the URL `http://example.com/magic/example.html?term=sushi`
- `path` – uses the value of the __n__th component of the URL path, as specified by the value. For example `!path!3` will auto-search for the word "dinosaur" if the page containing that widget is invoked from the URL `http://example.com/magic/lookup/dinosaur`
- `var` – uses the value of the named JavaScript global variable. This is a very powerful and general mechanism. For example, to search for the reversed value of the query parameter called `reverseTerm`, you might write a JavaScript function to do the extraction and reversing, then use the following HTML:

```
<script>var _reversedParam = extractAndReverse("term");</script>
<div class="mkws-results" autosearch="!var!_reversedParam">
```

Assembling Pazpar2 URLs

Most of MKWS's functionality is achieved by use of the Pazpar2 middleware. This is accessed on an endpoint URL which is usually assembled from the two configuration settings `pp2_hostname` and `pp2_path`. However, if for some reason an unusual Pazpar2 endpoint must be used, that endpoint can be specified in the `pazpar2_url` setting, and that will be used instead.

In the common case where Pazpar2 is accessed via the Service Proxy, an authentication call is made during initialisation. The call is generally made to the same endpoint as the other requests. However, the hostname used for authentication may if necessary be overridden using the `sp_auth_hostname` setting, and the path overridden by `sp_auth_path`. In any case, the value of `sp_auth_query` is appended; and if `sp_auth_credentials` is set, then it is used to add username and password parameters.

So in the absence of any configuration added by an application, the Service Proxy authentication URL is made up of `pp2_hostname` (sp-mkws.indexdata.com) since `sp_auth_hostname` is undefined; and `pp2_path` (service-proxy/) since `sp_auth_path` is undefined; and `sp_auth_query` (command=auth&action=perconfig); and no credentials, since `sp_auth_credentials` is undefined. Therefore the URL `http://sp-mkws.indexdata.com/service-proxy/?command=auth&action=p` is generated.

Language specification

Support for another UI language can be added by providing an entry in the `mkws_config` object whose name is `language_` followed by the name of the language: for example, `language_French` to support French. The value of this entry must be a key-value lookup table, mapping the English-language strings of the UI into their equivalents in the specified language. For example:

```
var mkws_config = {
  language_French: {
    "Authors": "Auteurs",
    "Subjects": "Sujets",
    // ... and others ...
  }
}
```

The following strings occurring in the UI can be translated:

- `Search complete:  found`
- `records`

- Displaying
- to
- of
- found
- Prev
- Next
- Sort by
- and show
- per page
- Search
- Active clients
- Retrieved records
- Records
- Targets
- Target ID
- Hits
- Diags
- Records
- State

In addition, facet names can be translated:

- Authors
- Sources
- Subjects

and whatever field captions are defined by `facet_caption_*` settings.
And sort-orders:

- relevance
- title

- `newest`
- `oldest`

and whatever sort-orders are defined by the `sort_options` setting.

Finally, the names of fields in the full-record display can be translated. These include, but may not be limited to:

- `Title`
- `Date`
- `Author`
- `Links`
- `Subject`
- `Locations`

jQuery UI popup invocation

The MasterKey Widget Set can be invoked in a popup window at the top of the page.

Note that the `popup` widget uses facilities from the jQuery UI, so it's necessary to include both CSS and JavaScript from that toolkit. The relevant lines are:

```
<script src="http://code.jquery.com/ui/1.10.3/jquery-ui.min.js"></script>
<link rel="stylesheet" type="text/css"
      href="http://code.jquery.com/ui/1.10.3/themes/smoothness/jquery-ui.css" />

<div class="mkws-search"></div>
<div class="mkws-popup" popup_width="1024" popup_height="650" popup_autoOpen="0">
  <div class="mkws-switch"></div>
  <div class="mkws-lang"></div>
  <div class="mkws-results"></div>
  <div class="mkws-targets"></div>
  <div class="mkws-stat"></div>
</div>
```

Popup windows can contain any HTML, not just MKWS widgets.

The properties of the `popup` widget are as follows:

Setting	Type	Default	Description
popup_width	int	880	Width of the popup window, in pixels.
popup_height	int	760	Height of the popup window, in pixels.
popup_button	string	`input.mkwsButton`	A jQuery selector identifying the element which, which clicked, pops up the window.
popup_modal	bool	0	Indicates whether the popup is modal (blocks access to the background page until dismissed). Set to 1 if a modal popup is required.
popup_autoOpen	bool	1	Indictaes whether to pop up the window automatically when the page is loaded.

Multiple `popup` widgets can co-exist on a page. In this case, different `popup_button` values must be used for each.

Structure of HTML generated by widgets

In order to override the default CSS styles provided by the MasterKey Widget Set, it's necessary to understand the structure of the HTML elements that are generated within the widgets. The HTML structure is as follows. As in CSS *.class* indicates an instance of a class. A trailing `*` indicates zero or more instances; a trailing `?` indicates zero or one instance.

```
.mkws-progress
  span.mkws-done
  span.mkws-waiting

.mkws-search
  form.mkws-search-form
    input.mkws-query
    input.mkws-button

.mkws-results
  table
    tbody
      tr
        td.mkws-facets-container-wide
          div.mkws-facets
            div.mkws-facet*
              div.mkws-facet-title
              div.mkws-term*
                a
                span
        td.mkws-motd-container
          div.mkws-ranking
            form
              select.mkws-sort
                option*
              select.mkws-perpage
                option*
          div.mkws-pager
            div
            div
              span.mkws-prev
              span.mkws-current-page
              a*
              span.mkws-next
          div.mkws-navi
          div.mkws-records
            div.mkws-summary*
              div.mkws-field-data
```

```
              span.mkws-field-NAME*
            div.mkws-details?
              table
                tbody
                  tr*
                    th
                    td
      tr
        td
          div.mkws-facets-container-narrow

.mkws-targets
  table
    thead
      tr
        td*
    tbody
      tr*
        td*
```

Appendix: compatibility roadmap

Wherever possible, we ensure that all functional changes in MKWS are backwards-compatible, so that applications written against old versions of the toolkit will continue to work when running against newer versions.

However, a few aspects of functionality may unavoidably change in backwards incompatible ways. We ensure that **this only happens with new major versions** – so it should *always* be safe to upgrade to a new minor version. As an aid to porting old applications, we here note the specific backwards-incompatible changes in the various major releases, and those planned for future major releases:

Major version 1.x

Versions of MKWS before v1.0 (including the only prior release, v0.9.1) used camel-case class-names: without hyphens and with second and subsequent words capitalised. So instead of `mkws-search`, it used to be `mkwsSearch`. And the classes used to specify team names used an `mkwsTeam_` prefix (with an underscore). So instead of `mkws-team-foo`, it used to be `mkwsTeam_foo`.

The 1.x series of MKWS releases recognise these old-style class-names as well as the canonical ones, as a facility for backwards compatibility. However, **these old class-names are deprecated, and support will be removed in v2.0**. Existing applications that use them should be upgraded to the new-style class names as soon as is convenient.