WebHare for development

This manual will explain how to setup a local WebHare installation for development and will guide you through building a website. We will be using Docker to manage the WebHare installation.

> ### Docker limitations
>
> Please note that Docker has some limitations (especially on Mac and Windows) in performance and ease of development that you wouldn't see on a production WebHare installation, but gives a much easier setup experience. All production WebHare setups are generally done in Docker on a Linux (virtual) machine.

If you have sufficient experience manually maintaining and compiling software, you can alternatively install WebHare from source.

Installing WebHare

First, install Docker from https://www.docker.com/get-started.

On Windows

Run the following commands:

```
mkdir %USERPROFILE%/whdata
docker run -ti --rm --name webhare -p 80:80 -p 443:443 -v %USERPR
```

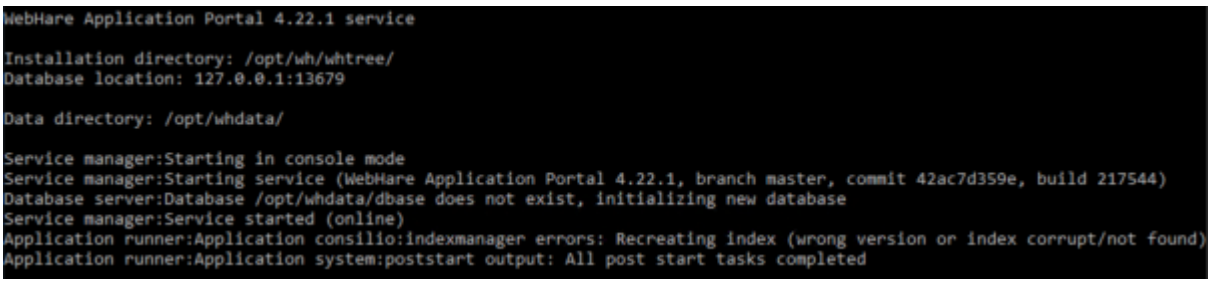
If Docker asks you whether you want to share your C: drive, you should probably do so.

On macOS or Linux

Run the following commands:

```
mkdir ~/whdata
docker run -ti --rm --name webhare -p 80:80 -p 443:443 -v ~/whdat
```

You'll see something similar to the following screen if WebHare started correctly:



WebHare ships with a CLI tool to control it named 'wh'. We can use docker to invoke this tool for us. One use is to check WebHare's status. Let's see if everything is working correctly:

```
docker exec webhare wh check
```

You will see something like:

```
Arnolds-MBP:~ arnold$ wh-moe3 check
WebHare 4.22.1 - 2 issues!
The email fallback FROM address has not been configured
Backend WebHare URL unknown
```

It's running, but we cannot connect to it yet . We need to be able to access what we often simply call the 'backend' - the WebHare backend interface.

Setting up the Webhare backend interface

On a production server we would generally setup a proxy server to host the interface, but here we'll let the built-in webserver handle it by itself. We need an open port for virtual hosting so we can run multiple websites on the same port (in our case: the WebHare backend and a future output website).

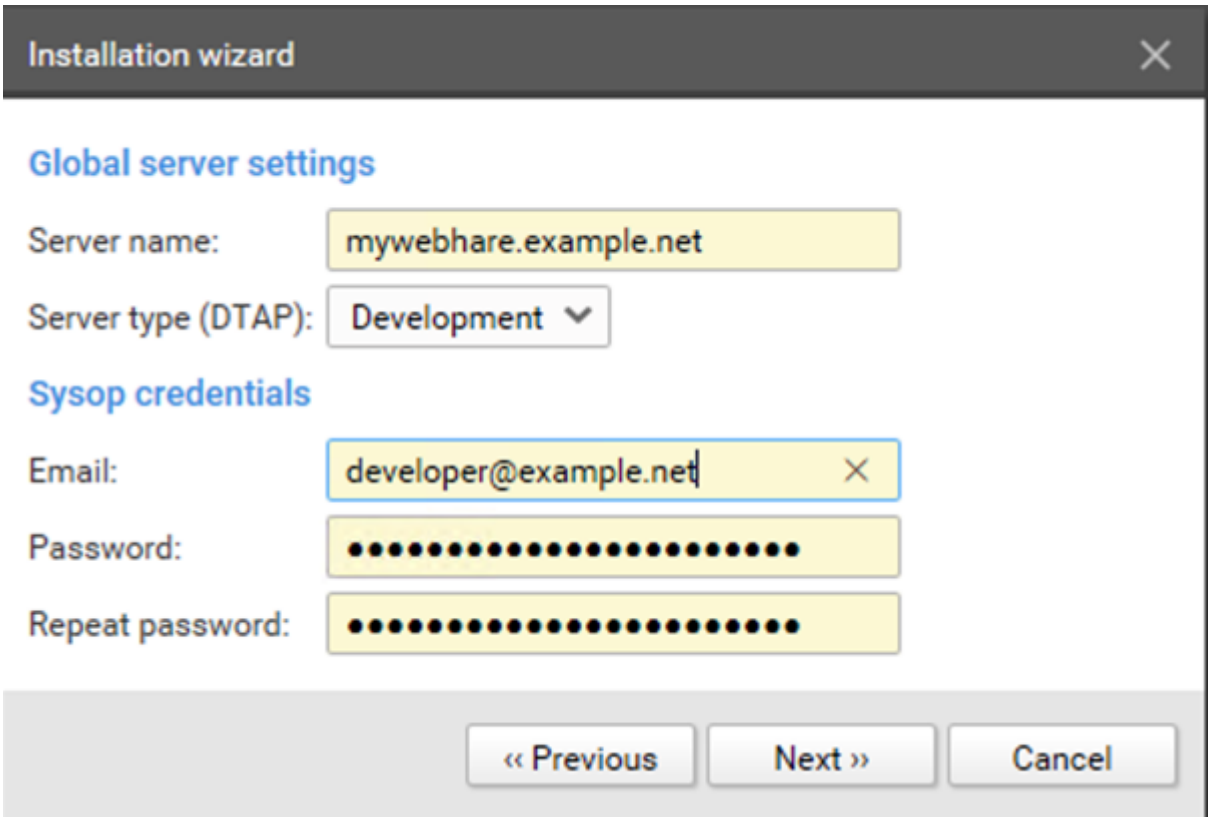
> ### Nonstandard port numbers
>
> We strongly recommend using the standard port 80/443 for your WebHare installation and using virtual hosting - some features may be more difficult to setup or have subtle issues with non standard port numbers. You will need to modify the docker command line used during installation to open up other ports than 80 and 443.

To create a virtual hosted port on port 80 and host a site named 'localhost' on it:

```
docker exec webhare wh webserver addport --virtual 80
docker exec webhare wh webserver addbackend http://localhost/
```

- Visit the URL you've just added - eg http://localhost/.

- You will be greeted by the installation wizard:



- Set up an account. Be sure to set the DTAP stage to 'Development'.

When you finish the installation wizard, you will need to actually login with the specified credentials.

Setting up output

To configure output please consult our Webservers setup manual

TODO!

- uitleggen hoe je een web server maakt
- verwijzen naar Tools > Set website debug settings en vervolgens 'enable output debug mode'.

Creating a webdesign

explain webdesign basics (what is it, how to use it)

prerequisites: running webhare, configured 'wh' command, output server

Creating a module

```
wh module create [path/]<modulename>
```

Create and initialize the webdesign

```
wh module createwebdesign <modulename>:<webdesignname> <domainnam
```

domainname is used to create the namespace used inside the webdesign.
If domainname is “example.com” the first part of the namespace used in the webdesign will become: http://example.com/

Creating the website

Creating a website that uses the webdesign is done in the Publisher application.

- Start the WebHare publisher from the Start Menu
- Select the folder (or create new folder and select it) where you want your website.
- Select in ‘Menu’ option File > New > Website…
- The “New site” dialog is opened. You can now set:
  - name;
  - output URL;
  - if the site is in a subfolder: subfolder in output URL. (default ‘/’)
- In the tab ‘Design’ select your previously created webdesign.
- Press ‘Ok’ to confirm settings.

Changing layout

> **Example site**
>
> Verwijzen naar hoofdstuk Example site

To edit the site template, goto your webhare installation folder

You find the files for the site layout in:

webhare/whtree/whdata/installedmodules/<modulename>/webdesigns/<webdesignname>/

Explain static files and changes template only visible after republish if not in preview mode

Webdesign files you can edit are:

Also there is a <webdesignname>/'web/img/' folder where you can put all static images used in the template. (accessible in template by [imgroot]imagefilename)

To change the basic layout you need to edit:
<webdesignname>.scss
<webdesignname>.witty
/shared/rtd/rtd.css

You can find basic styling for RTD content in '/shared/rtd/rtd.css'

If you want to use a custom font, add at top the rtd.css file the import rule like:

@import url(//fonts.googleapis.com/css?family=Open+Sans:400,700,400i,700i);

Inline images in the RTD files are set to a maximum width which is defined in the main siteprofile <webdesignname>.siteprl.xml by the attribute 'maxcontentwidth' in the 'webdesign' node.

The site language can be found in this siteprofile in element 'sitelanguage'.

You can activate stats from google by using/activate one of the settings in the siteprofile:

```
    <googleanalytics account="UA-XXXXXXX-X" />
    <gtm account="GTM-XXXXXX" />
```

If using googleanalytics, default 'anonymizeip' is enabled.

The main witty template contains two required components, a htmlhead component and a htmlbody component.

component htmlhead

The htmlhead is used for elements inside the head-element

You can add additional meta tags inside the htmlhead component like meta tags for the site icons.

Some basic elements and links to the compiled javascript and css for the webdesign are automatically generated inside this component.

Opengraph meta is automatically placed inside this component if opengraph plugin is activated in the siteprofile or by the GetPlugin function in the harescript library.

```
  <opengraph xmlns="http://www.webhare.net/xmlns/socialite"
             site_name="WebHare - Examplesite"
             type="website"
```

```
5               image="web/img/logo-big.png"
                />
```

(If using GetPlugin function the plugin '<opengraph xmlns="http://www.webhare.net/xmlns/socialite" />' must be defined in the siteprofile)

Robots meta (like '<meta name="robots" content="noindex">') is automatically placed inside this component if robots plugin is activated in the siteprofile or by the GetPlugin function in the harescript library.

```
1   <robots xmlns="http://www.webhare.net/xmlns/consilio"/>
```

(If using GetPlugin function the plugin '<robots xmlns="http://www.webhare.net/xmlns/consilio"/>' must be defined in the siteprofile)

component htmlbody

The htmlbody is used for elements inside the body-element.

It should at least have the macro [contents] which is used to render RTD content.

Basic forms

If you want to add a contact form to the site, create new file type 'Form'. You can style the form by altering the content of /shared/forms/forms.scss. The full form layout, with all basic fields, can be tested by selecting in the publisher a file in your website and then using the 'Open forms test' in the publisher menu under option 'Tools'.

Add custom widget in RTD with just html/js

You can add extra components to the RTD by adding additional rules to the siteprofiles xml and components to the witty file

For example if you want to offer a 'weather' widget option inside the RTD:

Create folder widgets/weather/

Add to this folder the file weather.siteprl.xml with the content:

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <siteprofile xmlns="http://www.webhare.net/xmlns/publisher/sitepr
3    <widgettype namespace="http://yourdomainname/xmlns/widgets/weat
4                title="Weather"
5                wittycomponent="weather.witty:weather">
6    </widgettype>
7  </siteprofile>
```

Now, for example, if you want to use the weather widget from https://weatherwidget.io/

Add file weather.witty to the folder with content:

```
[rawcomponent weather]
  <div class="widget widget-weather">
    <a class="weatherwidget-io" href="https://forecast7.com/en/52
    <script>
    !function(d,s,id){var js,fjs=d.getElementsByTagName(s)[0];if(
    </script>
  </div>
[/rawcomponent]
```

By using 'rawcomponent' the content of this component will not be altered or interpreted while rendered.

Add to the main siteprofile (<webdesignname>.siteprl.xml) directly after the existing applysiteprofile rule:

```
<applysiteprofile path="widgets/weather/weather.siteprl.xml" />
```

Add to RTD siteprofile definition (/shared/rtd/rtd.siteprl.xml) inside the <widgets /> element:

```
<allowtype type="http://yourdomainname/xmlns/widgets/weather" />
```

Create in the publisher, in your website, a new richdocument file.

Edit this file by double clicking and then you can add, besides text and images, the weather widget to your page.

After publishing the edited RTD file the contents of this file is generated in the template

Adding navigation

To add navigation to the template you first need to edit the harescript library <webdesignname>.whlib in your editor

Inside this file you find a public objecttype <webdesignname>Design

There you find in comments the record function GetPageConfig()

All fields in the return record of this function are accessible in the witty template.

Uncomment the function and add inside it:

```
RETURN [ mainnav := this->GetMainNav()
       ];
```

And after the pageconfig function add a getmainnav function.

The query for the main navigation is done by only selecting folders with title directly within the site root from the 'system' database in the 'fs_objects' table.
The 'system.fs_objects' table contains all publisher files and folders.
There are three objects available which you can use within queries to build your navigation.

```
RECORD ARRAY FUNCTION GetMainNav()
{
  RETURN (SELECT *
                , isselected := this->targetobject->whfspath LIKE
            FROM system.fs_objects
           WHERE parent = this->targetsite->root // Items in site
             AND link != ""      // Published pages
             AND title != ""     // Must have title
             AND isfolder        // Only folders
         ORDER BY ordering, ToUpperCase(title), name);
}
```

After this you can add the navigation to the witty template inside the htmlbody component

```
<header>
  <nav id="mainnav">
    <a href="[siteroot]" [if ishomepage]class="active"[/if]>Home<
    [forevery mainnav]
      <a href="[link]" [if isselected]class="active"[/if]>[title]
    [/forevery]
  </nav>
</header>
```

To see the navigation working you have to add folders (with title!) and a published RTD file in the site root as index (home).

Subnavigation

To add subnavigation for current active page, alter the pageconfig return record to:

```
RETURN [ mainnav := this->GetMainNav()
       , subnav  := this->GetSubNav(this->targetfolder->id)
       ];
```

And after the pageconfig a getsubnav function:

```
RECORD ARRAY FUNCTION GetSubNav( INTEGER folderid )
{
  RETURN (SELECT *
                , isselected := this->targetobject->whfspath LIKE
            FROM system.fs_objects
           WHERE parent = folderid // Items in current folder
             AND link != ""        // Published pages
             AND title != ""       // Must have title
             AND id != this->targetfolder->indexdoc //Ignore inde
             AND (parent != this->targetsite->root OR NOT isfolde
        ORDER BY ordering, ToUpperCase(title), name);
}
```

After this add to the witty template inside the htmlbody component:

```
<aside>
  [if subnav]
    <nav id="subnav">
      [forevery subnav]
        <a href="[link]" [if isselected]class="active"[/if]>[titl
      [/forevery]
    </nav>
  [/if]
</aside>
```

Adding path/crumbtrail.

To add path navigation for current active page, alter the pageconfig return record to:

```
RETURN [ mainnav := this->GetMainNav()
       , subnav  := this->GetSubNav(this->targetfolder->id)
       , pathnav := this->GetPathNav()
       ];
```

And after the pageconfig the getpathnav function:

```
  RECORD ARRAY FUNCTION GetPathNav()
  {
    //Get all ids from site root until current folder
    // First item is the site root, last the current folder id
    INTEGER ARRAY foldertree_ids := GetFolderTreeIDs(this->target

    RECORD ARRAY pathnav := SELECT *, title := title ?? name
                              FROM system.fs_objects
                             WHERE id IN foldertree_ids
                               AND link != ""  // only with publi
                               AND isfolder
                         ORDER BY SearchElement(foldertree_ids,

    IF( Length(pathnav) > 0 )
    {
      pathnav[0].title := "Home"; //Overwrite first title (home)

      //If current file is not the index, add current file to pat
      IF( this->targetobject->id != this->targetfolder->indexdoc
      {
        INSERT [ link  := this->targetobject->link
               , title := this->targetobject->title ?? this->targ
               ] INTO pathnav AT END;
      }

      IF( Length(pathnav) = 1 )
        RETURN DEFAULT RECORD ARRAY;//If just one item (home), th
    }

```

```
    RETURN pathnav;
  }
```

And in the witty add:

```
[if pathnav]
  <ol id="pathnav">
    [forevery pathnav]
      <li><a href="[link]">[title]</a></li>
    [/forevery]
  </ol>
[/if]
```

For styling the navigation edit the scss file (<webdesignname>.scss).

Add extra properties to page

You can add extra properties to a page by adding rules to the siteprofile xml

For example adding an SEO title option which overrides the title in html head element.

Add following to the siteprofile:

```
  <contenttype namespace="http://yourdomainname/xmlns/page" clone
    <member type="string" name="seotitle" />
  </contenttype>
  <tabsextension xmlns="http://www.webhare.net/xmlns/tollium/scre
                 name="pagesettings"
                 implementation="none"
                 >
    <insert position="title" where="after">
      <textedit composition="contentdata" cellname="seotitle" wid
    </insert>
  </tabsextension>
  <apply>
    <to type="file" />
    <extendproperties extension=".pagesettings" contenttype="http
  </apply>
```

After adding this part to the siteprofile you have an extra field ‘SEO title’ in the file properties just after the title field.

To use this field in the template, you have to add some harescript to the pageconfig function.

You can use this->pagetitle for altering the page title set in head/title, default the pagetitle is the current file title if set. If not set, it will be the folder title or name if folder title is not set.

For changing the pagetitle add to the pageconfig function:

```
RECORD pagesettings := this->targetobject->GetInstanceData("http:

//Format html/browser-tab/window title (pagetitle - sitetitle) if
IF( this->pagesettings.seotitle != "" )
  this->pagetitle := pagesettings.seotitle;
ELSE
{
  this->pagetitle := this->siteconfig.sitetitle;
  IF( this->targetobject->title != "" ) //If page has title, add
    this->pagetitle := this->targetobject->title || " - " || this
  ELSE IF( this->targetfolder->id != this->targetsite->root AND t
    this->pagetitle := this->targetfolder->title || " - " || this
}
```

Adding date option to news page

If you want an extra field ‘date’ specific for pages in news folder, change siteprofile xml for the contenttype with the page properties to:

```
<contenttype namespace="http://yourdomainname/xmlns/page" cloneor
  <member type="file" name="image" />
  <member type="datetime" name="date" />
</contenttype>
```

And add to siteprofile a new custom ‘news’ folder type:

```
<!-- News folder -->
<contenttype namespace="http://examplesite.webhare.com/xmlns/fold
<foldertype typedef="http://examplesite.webhare.com/xmlns/folders
```

```
            title="News"
            tolliumicon="tollium:folders/news" />
<apply>
  <to type="folder" />
  <allowfoldertype typedef="http://examplesite.webhare.com/xmlns/
</apply>
```

Then add to siteprofle the screen definition for editing the date for each news page

```
<tabsextension xmlns="http://www.webhare.net/xmlns/tollium/screer
               name="newspagesettings"
               implementation="none"
               >
  <insert position="description" where="after">
    <datetime composition="contentdata" cellname="date" required=
  </insert>
</tabsextension>
<apply>
  <and>
    <to type="file" parenttype="http://examplesite.webhare.com/xm
    <not><to type="index" /></not>
  </and>
  <extendproperties extension=".newspagesettings" contenttype="ht
</apply>
```

After this you can create a news type folder and every file (except the index) in this folder has extra date field just after the description field in the file properties.

To display the date in the template first add

```
LOADLIB "wh::datetime.whlib";
```

at top of the harescript library [webdesignname].whlib so you can use the date formatting function.

Then add date to the pageconfig function the return record. We use FormatDateTime to present the date in a readable format.

```
RETURN [ mainnav := this->GetMainNav()
       , subnav  := this->GetSubNav(this->targetfolder->id)
       , pathnav := this->GetPathNav()
       , date    := FormatDateTime("%d %B %Y", pagesettings.date,
       ];
```

After this you can add next line to the witty template

```
[if date]<div class="pagedate">[date]</div>[/if]
```

Adding headerimage to page

If you want a pageimage for every page, you need to add the following.

Change xml for the page contenttype properties to:

```
<contenttype namespace="http://yourdomainname/xmlns/page" cloneor
  <member type="string" name="seotitle" />
  <member type="datetime" name="date" />
  <member type="file" name="headerimage" />
</contenttype>
```

Then add to tabextention named ‘pagesettings’

```
<newtab title="Header image">
  <imgedit composition="contentdata"
           cellname="headerimage"
           width="1pr"
           height="350px"
           title="Header image"
           allowedactions="all refpoint" />
</newtab>
```

After this you should see an extra tab ‘Header image’ in the properties of every file in the site.

To display the date in the template first add

```
LOADLIB "mod::system/lib/cache.whlib";
```

in top of the harescript library [webdesignname].whlib so you can use the image cache and resize function WrapCachedImage.

Now you can add

```
, headerimage := WrapCachedImage( pagesettings.headerimage, [ met
                                                            , set
                                                            , set
                                                            ])
```

inside the pageconfig return record.

In the witty template you change the header element to

```
<header>
  [if headerimage]
    <img class="headerimage" src="[link]" alt=""/>
  [/if]
  <nav id=”mainnav”>
    [forevery mainnav]
      <a href=”[link]” [if isselected]class=”active”[/if]>[title]
    [/forevery]
  </nav>
</header>
```

and add the additional appropriate css styling to [webdesignname].scss

Add extra properties to site

You can add extra properties to the site folder by adding rules to the siteprofile xml

For example adding custom selectable footer navigation.

Add following to the siteprofile:

```
  <contenttype namespace="http://yourdomainname/xmlns/site">
    <member type="array" name="footerlinks">
      <member type="string" name="title" />
      <member type="intextlink" name="link" />
    </member>
```

```
  </contenttype>

  <tabsextension xmlns="http://www.webhare.net/xmlns/tollium/scre
                 name="sitesettings"
                 implementation="none"
                 >
    <newtab title="Footer">
      <heading title="Footer links" />
      <arrayedit rowselect="true"
                 composition="contentdata"
                 cellname="footerlinks"
                 roweditscreen=".editfooterlinks"
                 width="1pr"
                 height="1pr"
                 orderable="true">
        <column name="title" title="Title"  type="text" width="1p
        <p:intextlinkcolumn name="link" title="Link" width="2pr"
      </arrayedit>
    </newtab>
  </tabsextension>
  <apply>
    <to type="folder" pathmask="/" />
    <extendproperties extension=".sitesettings" contenttype="http
  </apply>

  <screen name="editfooterlinks"
          title="Edit link"
          implementation="rowedit"
          minwidth="350px"
          xmlns="http://www.webhare.net/xmlns/tollium/screens">
    <compositions>
      <record name="row" />
    </compositions>
    <body>
      <textedit composition="row" cellname="title" title="Title"
      <p:intextlink composition="row" cellname="link" title="Link
    </body>
    <footer>
      <defaultformbuttons buttons="ok cancel" />

```

```
    </footer>
  </screen>
```

And add at top namespace for components ‘xmlns:p’ to the siteprofile

```
<siteprofile xmlns="http://www.webhare.net/xmlns/publisher/sitepr
             xmlns:m="http://www.webhare.net/xmlns/system/modulec
             xmlns:p="http://www.webhare.net/xmlns/publisher/comp
```

After this you see an extra tab ‘Footer’ in the properties of the site folder where you can add links for the footer.

To display the footer links in the template first add

```
LOADLIB "mod::publisher/lib/publisher.whlib";
```

in top of the harescript library [webdesignname].whlib so you can use the function GetIntextLinkTarget for link resolving.

Then add to the pageconfig function:

```
RECORD sitesettings := this->targetsite->rootobject->GetInstanceD
```

And add in the return record of the pageconfig function:

```
, footernav := (SELECT title, link := GetIntextLinkTarget(link) F
```

Now you can use footernav in the witty template like:

```
<footer>
  [if footernav]
    <nav id="footernav">
      [forevery footernav]
        <a href="[link]">[title]</a>
      [/forevery]
    </nav>

```

```
    [/if]
  </footer>
```

Adding custom widgets with extra properties in RTD

Adding a custom widget in the RTD with extra properties like a two columns widget

Create folder widgets/twocolumns/

Add to folder file: twocolumns.siteprl.xml with content:

```
<?xml version="1.0" encoding="UTF-8" ?>
<siteprofile xmlns="http://www.webhare.net/xmlns/publisher/sitepr
<widgettype namespace="http://yourdomainname/xmlns/widgets/twocol
              title="Two columns"
              editfragment=".edittwocolumns"
              renderlibrary="twocolumns.whlib"
              renderobjectname="EmbedTwoColumns"
              wittycomponent="twocolumns.witty:twocolumns"
              >
    <members>
      <member name="left" type="richdocument" />
      <member name="right" type="richdocument" />
    </members>
  </widgettype>

  <fragment name="edittwocolumns"
            xmlns="http://www.webhare.net/xmlns/tollium/screens"
            implementation="none">
    <contents>
      <grid>
        <col width="1pr" />
        <col width="1pr" />
        <row>
          <cell height="1pr">
            <richdocument cellname="left"
                          composition="contentdata"
                          errorlabeltid=".left"
                          required="true"
                          width="400px"
                          height="1pr"
```

```
                            minheight="330px"
                            />
          </cell>
          <cell height="1pr">
            <richdocument cellname="right"
                            composition="contentdata"
                            errorlabeltid=".right"
                            required="true"
                            width="400px"
                            height="1pr"
                            minheight="330px"
                            />
          </cell>
        </row>
      </grid>
    </contents>
  </fragment>

</siteprofile>
```

Create file twocolumns.witty with content:

```
[component twocolumns]
  <div class="widget-twocolumns">
    <div class="col">
      [left]
    </div>
    <div class="col">
      [right]
    </div>
  </div>
[/component]
```

Create file twocolumns.whlib with content:

```
<?wh
LOADLIB "mod::publisher/lib/widgets.whlib";

PUBLIC OBJECTTYPE EmbedTwoColumns EXTEND WidgetBase
```

```
<
  MACRO PTR left;
  MACRO PTR right;

  MACRO NEW()
  { //Pointer for RTD rendering always before Render function
    this->left := PTR this->context->OpenRTD(this->data.left)->Re
    this->right := PTR this->context->OpenRTD(this->data.right)->
  }

  UPDATE PUBLIC MACRO Render()
  {
      this->EmbedComponent([ left  := this->left
                           , right := this->right
                           ]);
  }
>;
```

Add to the main siteprofile (<webdesignname>.siteprl.xml) directly after the existing applysiteprofile rule:

```
<applysiteprofile path="widgets/twocolumns/twocolumns.siteprl.xml
```

For styling add file twocolumns.css to the folder with:

```
.widget-twocolumns
{
  display: flex;
}
.widget-twocolumns > .col
{
  flex: 0 1 50%;
  max-width: 50%;
  padding-right: 15px;
}
.widget-twocolumns > .col + .col
{
  padding-left: 15px;
```

```
    padding-right: 0;
}
@media(max-width: 600px)
{
    .widget-twocolumns
    {
        display: block;
    }
    .widget-twocolumns > .col
    {
        padding-right: 0;
    }
    .widget-twocolumns > .col + .col
    {
        padding-left: 0;
    }
}
```

Note: Use .css file when you want to use the styling in RTD-editor because editor preview does not handle .scss files

Add to main scss file definition (<webdesignname>.scss

```
@import "./widgets/twocolumns/twocolumns.css";
```

Add to RTD siteprofile definition (/shared/rtd/rtd.siteprl.xml)

after '<css path="rtd.css" />':

```
<css path="../../widgets/twocolumns/twocolumns.css" />
```

(This is used by the RTD-editor to preview the widget)

and inside the <widgets /> element:

```
<allowtype type="http://yourdomainname/xmlns/widgets/twocolumns"
```

Adding a search page

This uses the integrated WebHare search engine ‘Consilio’

Create folder pages/search/

Add to this folder the file search.siteprl.xml with the content:

```
<?xml version="1.0" encoding="UTF-8" ?>
<siteprofile xmlns="http://www.webhare.net/xmlns/publisher/sitepr
  <apply>
    <to type="file" />
    <prebuiltpage type="dynamic"
                  tag="modulename:mysearchpage"
                  library="search.whlib"
                  webpageobjectname="SearchPage" />
  </apply>
  <apply>
    <to type="file" filetype="http://www.webhare.net/xmlns/publis
    <robots xmlns="http://www.webhare.net/xmlns/consilio" noindex
  </apply>

  <!-- Consilio catalog -->
  <index xmlns="http://www.webhare.net/xmlns/consilio" name="modu
    <contentsource type="publisher:webhare" folder="site::website
  </index>
</siteprofile>
```

Add file search.whlib to the folder with content:

```
<?wh
LOADLIB "wh::witty.whlib";
LOADLIB "mod::consilio/lib/api.whlib";
LOADLIB "mod::publisher/lib/webdesign.whlib";
LOADLIB "mod::system/lib/webserver.whlib";

PUBLIC OBJECTTYPE SearchPage EXTEND DynamicPageBase
<
  UPDATE PUBLIC MACRO PrepareForRendering(OBJECT webcontext)
  {
    INSERT "searchpage" INTO webcontext->htmlclasses AT END;
```

```
  }

  UPDATE PUBLIC MACRO RunBody(OBJECT webcontext)
  {
    STRING words := GetWebVariable("words");
    RECORD results := RunConsilioSearch("webhare_com:example-site

    EmbedWittyComponent( this->pagefolder || "search.witty:search
  }

>;
```

Next add file search.witty to the folder with content:

```
[component search]
  <form action="./" method="get">
    <input type="search" value="[words]" name="words" placeholder
    <button type="submit">Search</button>
  </form>

  <div class="results">
    [if totalcount]
      [totalcount] item(s) found for '[words]'.
    [elseif words]
      No items found for '[words]'.
    [else]
      No search term given
    [/if]

    [if results]
      <ul class="searchresults">
        [forevery results]
          <li>
            <a href="[objectid]">
              <b class="title">[if title][title][else]<i>No title
              [if summary]
                <span class="summary">[summary]</span>
              [/if]
            </a>
```

```
            </li>
          [/forevery]
        </ul>
      [/if]
    </div>

[/component]

```

To make the consilio-search engine to create summaries of just the content of a page, add html comments, in the main witty template around the [contents] macro else the summary will also contain the navigation and footer content.

```
<!--wh_consilio_content-->
[contents]
<!--/wh_consilio_content-->
```

Add to the main siteprofile (<webdesignname>.siteprl.xml) directly after the existing applysiteprofile rule(s):

```
<applysiteprofile path="pages/search/search.siteprl.xml" />
```

Finally, create in the website a new file of type ‘prebuilt page’ and select in properties for ‘Prebuilt page’ the tag ‘mysearchpage’.

For styling add ‘search.css’ (or .scss) to the folder and add to main scss file definition (<webdesignname>.scss)

```
@import "./pages/search/search.css";
```

and put your styling for the search page in the search.css file.

Example site

An example site can be installed:

- Download the module with the example webdesign from gitlab.com to see the fully functional and optimized example webdesign.

- Download the archived version of the example website can be downloaded from https://www.webhare.dev/downloads/example-site.wharchive
- Open the Publisher application in WebHare
- Choose a folder for your website
- Upload the archive file and unpack the archive (double click).
- Select the extracted folder and choose “Sites -> Convert to site” from the Publisher menu.

(Working example site see: https://examplesite.webhare.dev/)

When using this example design for your own created website, you have change the search.siteprl.xml file with reference to the site name ‘site::example-site’ into reference to your own site like ‘site::<your-site-name>’

When using the example site as basis for a new moduledesign, you have to rename the first part of the used namespaces in the harescripts and siteprofiles to a new namespace used for your new moduledesign.