Wikibase Ecosystem Tutorial Workshop

Contents

Getting set up

You'll need:

- Linux, Mac or Windows with WSL operating system, running on AMD64/x86
- A working install of Docker that can run, say, the "hello world" example
  - Instructions for docker
  - hello world: `docker run hello-world`

Docker images

Download these docker images:

```
docker pull defreitas/dns-proxy-server:2.19.0
docker pull jwilder/nginx-proxy:0.10
docker pull docker-registry.wikimedia.org/dev/stretch-php74-
```

```
fpm:3.0.0
docker pull docker-registry.wikimedia.org/dev/buster-apache2:2.0.0
docker pull mariadb:10.8
```

IDE

To write some code, you'll need an IDE.

What's an IDE?: It stands for "integrated development environment". It's a tool that helps you write code.

- See also: https://www.howtogeek.com/837214/what-is-an-ide/

Why should I use an IDE?

- Can manage tasks for you: checkouts, commits, compiles, debugging, tests, etc.
- Syntax highlighting & auto-completion
- Easy code cleanup
- Easy navigation of code and documentation

Of course you can use the IDE of your choice, but we recommend these two:

- https://www.mediawiki.org/wiki/Visual_Studio_Code
- https://www.mediawiki.org/wiki/JetBrains_IDEs
    - For PHP: https://www.jetbrains.com/phpstorm/

This presentation uses VSCode for its examples.

Install mwcli

We'll be using mwcli, whiich implements a Docker-based development environment for MediaWiki. To install, follow this guide.

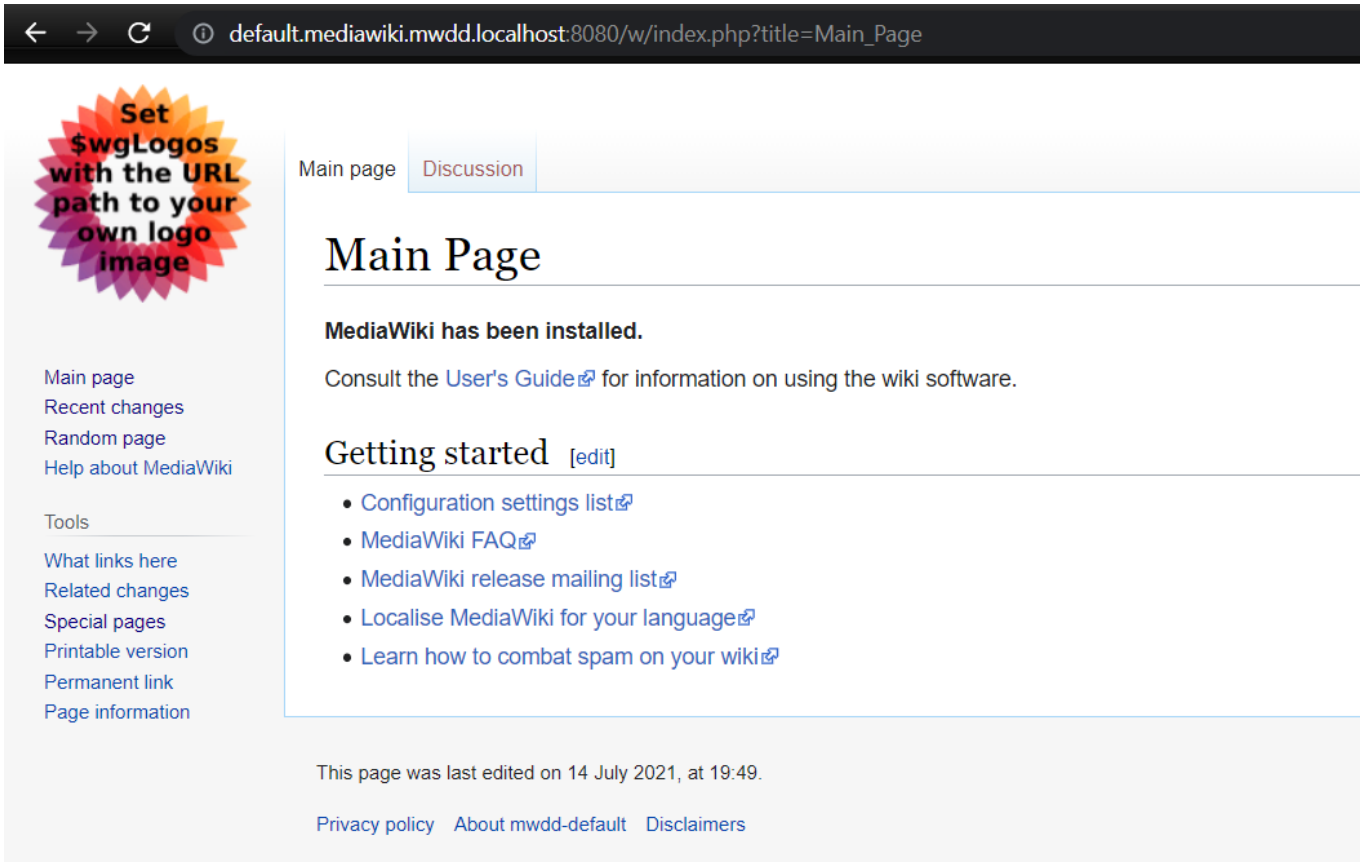
Using mwcli to create a development environment

Once mwcli's installed, you can set up your development environment.
This will also:

- clone MediaWiki
- install needed dependencies
- install an initial development site

To set up your development environment, follow this guide.

Now you'll have a local MediaWiki installation running in your development environment, accessible at http://default.mediawiki.mwdd.localhost:8080 (or whatever other port you chose during setup).



More on mwcli

Your development environment contains a lot more commands and tools that we haven't covered. Find out more by running `mw --help`. Some examples:

- `mw docker stop`: Stops all currently running containers
- `mw docker start`: Starts the containers that were previously running
- `mw docker destroy`: Deletes all containers and associated data (such as databases)
- `mw docker mailhog create`: Creates a mailhog service for receiving email
- `mw docker phpmyadmin create`: Creates a phpmyadmin service for viewing SQL databases

You can find the full online reference for the CLI on this wiki page.

Environment check

Let's make sure the development environment is running.

```
mw docker docker-compose ps
```

You should see that most services show a state of "Up". One or two may show the state "Exit 0". If they instead show as exited with any other code, you should restart them.

```
mw docker resume
```

If your list of services is empty, it probably means that at some point you ran `mw docker destroy`. In that case, you'll need to reinitialize your development environment:

```
mw docker mediawiki create
mw docker mysql create
mw docker mediawiki install --dbtype=mysql
```

Now you should be able to see your wiki at the URL mentioned above. If you chose a custom port (other than 8080) but forgot what it was, just run:

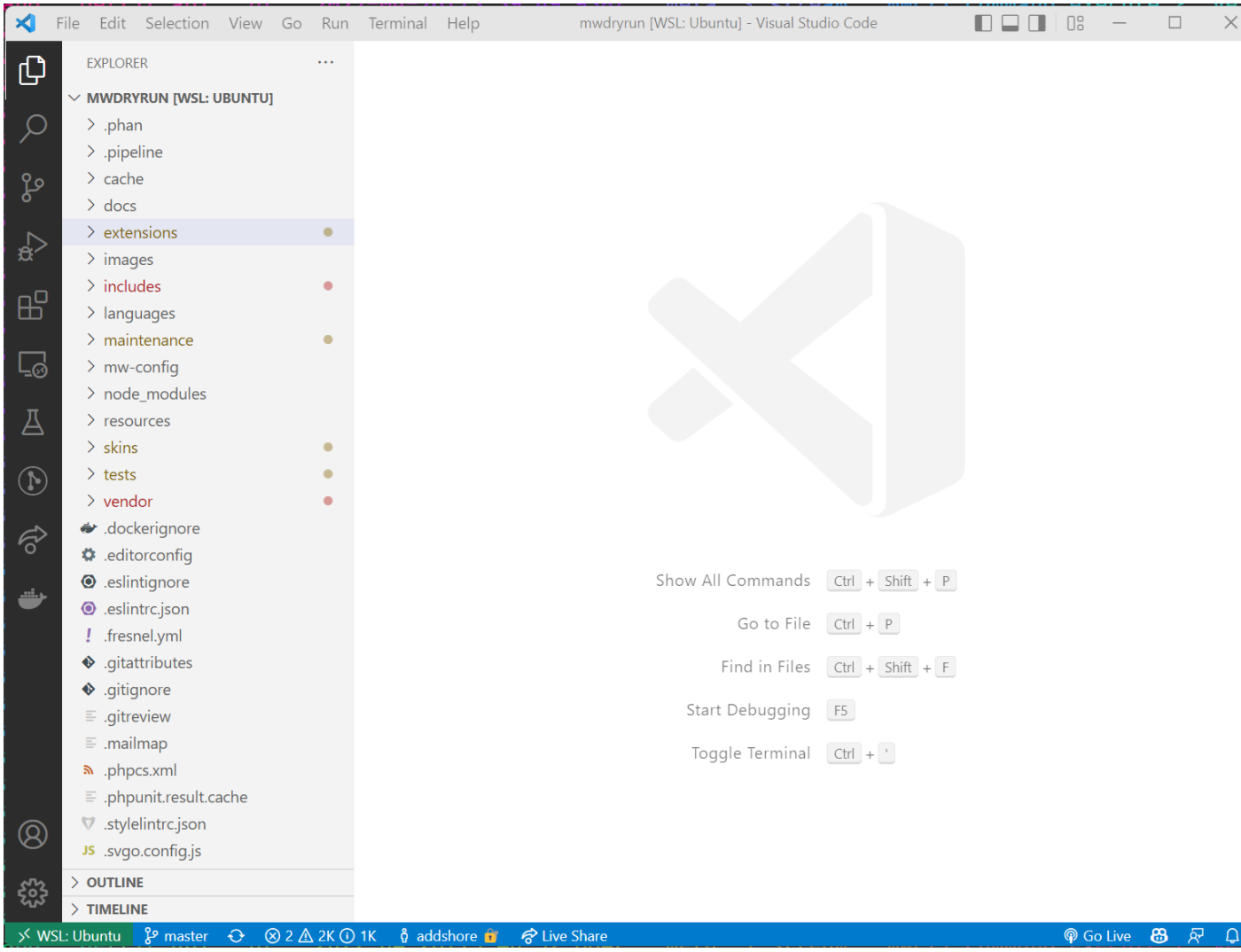
```
mw docker env get PORT
```

Writing some code

Load the BoilerPlate extension

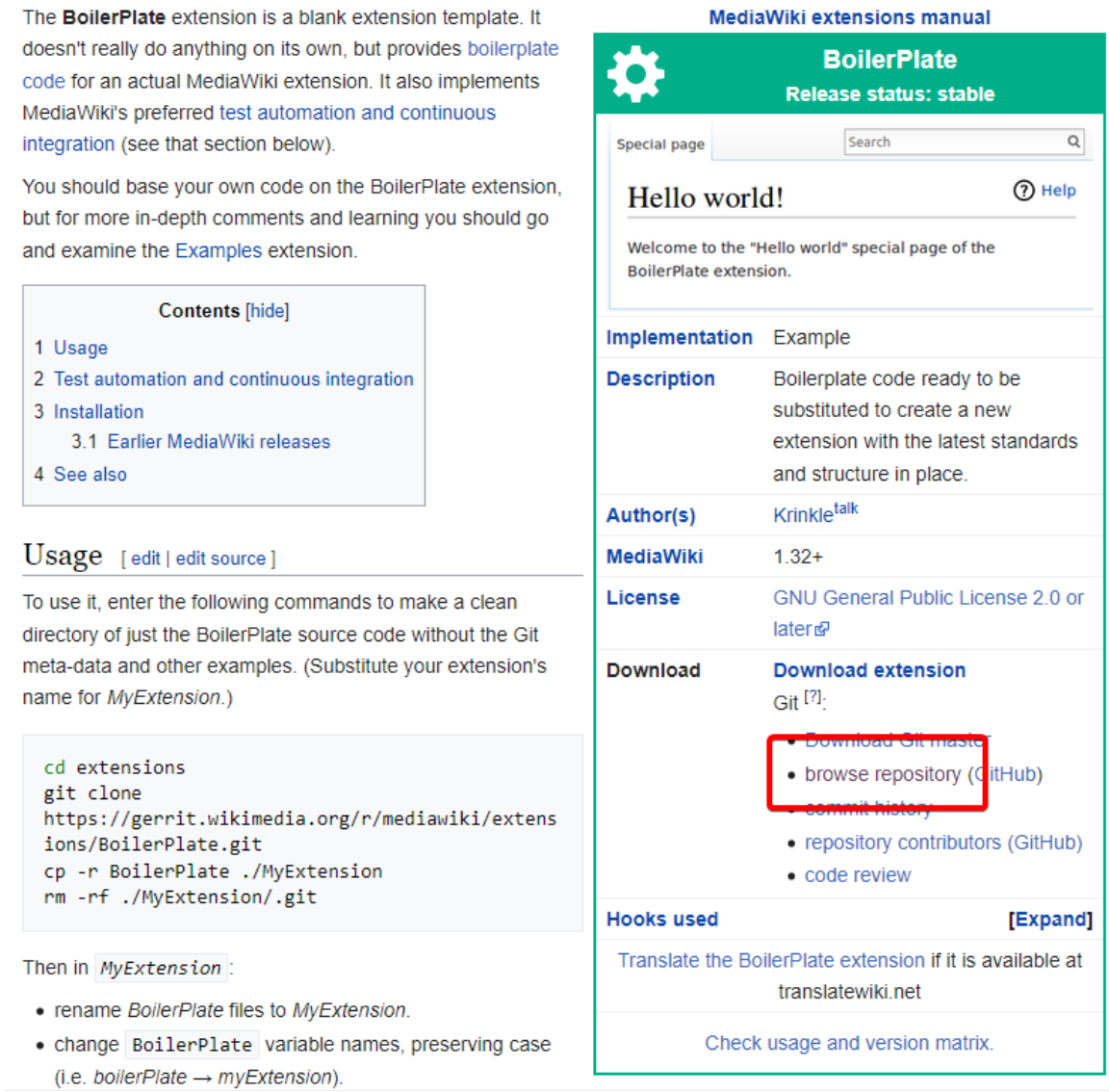
You now have MediaWiki code on your machine in the directory you chose during setup.

Open your IDE to see this code. In VSCode it'll look something like this:



Now add the BoilerPlate extension. It's a mostly blank extension template that displays example text.

Click "browse repository" in the sidebar to navigate to the git repository: https://gerrit.wikimedia.org/g/mediawiki/extensions/BoilerPlate



Extension:BoilerPlate

The **BoilerPlate** extension is a blank extension template. It doesn't really do anything on its own, but provides boilerplate code for an actual MediaWiki extension. It also implements MediaWiki's preferred test automation and continuous integration (see that section below).

You should base your own code on the BoilerPlate extension, but for more in-depth comments and learning you should go and examine the Examples extension.

Contents [hide]

1 Usage
2 Test automation and continuous integration
3 Installation
3.1 Earlier MediaWiki releases
4 See also

Usage [edit | edit source]

To use it, enter the following commands to make a clean directory of just the BoilerPlate source code without the Git meta-data and other examples. (Substitute your extension's name for *MyExtension*.)

```
cd extensions
git clone
https://gerrit.wikimedia.org/r/mediawiki/extens
ions/BoilerPlate.git
cp -r BoilerPlate ./MyExtension
rm -rf ./MyExtension/.git
```

Then in `MyExtension`:

- rename *BoilerPlate* files to *MyExtension*.
- change `BoilerPlate` variable names, preserving case (i.e. *boilerPlate* → *myExtension*).

MediaWiki extensions manual

BoilerPlate
Release status: stable

Special page | Search

Hello world! Help

Welcome to the "Hello world" special page of the BoilerPlate extension.

Implementation	Example
Description	Boilerplate code ready to be substituted to create a new extension with the latest standards and structure in place.
Author(s)	Krinkle talk
MediaWiki	1.32+
License	GNU General Public License 2.0 or later
Download	Download extension Git [?]: • Download Git master • browse repository (GitHub) • commit history • repository contributors (GitHub) • code review
Hooks used	[Expand]

Translate the BoilerPlate extension if it is available at translatewiki.net

Check usage and version matrix.

Now clone the repository into the MediaWiki extensions directory.

```
cd $(mw docker mediawiki where)/extensions
git clone
"https://gerrit.wikimedia.org/r/mediawiki/extensions/BoilerPlate"
```

You should see something like this:

```
~/git/gerrit/mediawiki/core/extensions (master) $ cd $(mw docker mediawiki where)/extensions
~/git/gerrit/mediawiki/mwcli-core/extensions (master) $ git clone ssh://gerrit.wikimedia.org:29418/mediawiki/extensions/BoilerPlate
Cloning into 'BoilerPlate'...

remote: Total 1598 (delta 0), reused 1598 (delta 0)
Receiving objects: 100% (1598/1598), 586.55 KiB | 1.08 MiB/s, done.
Resolving deltas: 100% (1045/1045), done.
```

Finally, enable the extension in MediaWiki by editing your `LocalSettings.php` file (in the root directory of your MediaWiki checkout) and adding the following line:

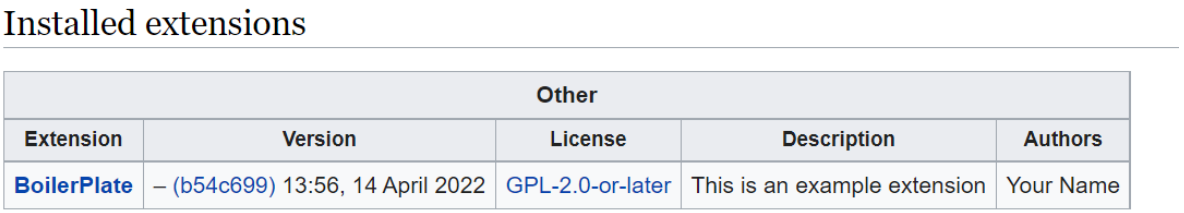
```
wfLoadExtension( 'BoilerPlate' );
```

At this point your `LocalSettings.php` should look something like this.

```
<?php
//require_once "$IP/includes/PlatformSettings.php";
require_once '/mwdd/MwddSettings.php';

wfLoadSkin('Vector');
wfLoadExtension( 'BoilerPlate' );
```

Navigate to the MediaWiki page Special:Version to see that the BoilerPlate extension is loaded.



Installed extensions

Other				
Extension	Version	License	Description	Authors
BoilerPlate	– (b54c699) 13:56, 14 April 2022	GPL-2.0-or-later	This is an example extension	Your Name

Success! You're now running MediaWiki in a development environment with an additional extension.

Create your own extension

General guide: https://www.mediawiki.org/wiki/Manual:Developing_extensions

In this section you'll make a custom extension called "TutorialWorld".

1. Create a "TutorialWorld" directory in the extensions directory of your MediaWiki checkout.
2. Create a file named `extension.json` that contains some basic information, as explained here: https://www.mediawiki.org/wiki/Manual:Developing_extensions#Registering_features_with_MediaWiki
3. Set `manifest_version` to 2 and `type` to "other".

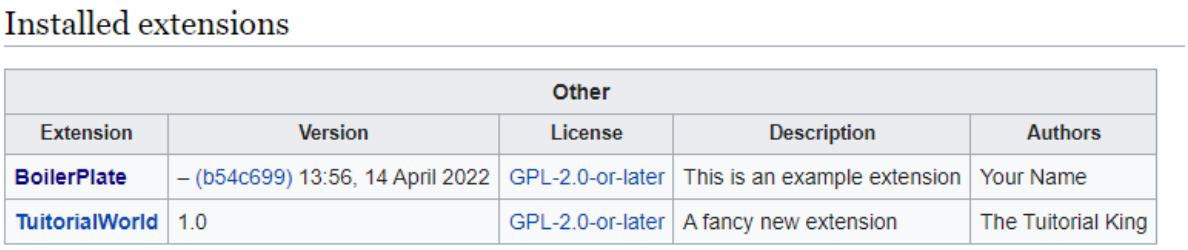
You should end up with something that looks like this:

```
{
	"name": "TutorialWorld",
	"author": "Your name here",
	"url":
"https://www.mediawiki.org/wiki/Extension:TutorialWorld",
	"description": "My amazing TutorialWorld extension",
	"version": "0.1",
	"license-name": "GPL-2.0-or-later",
	"type": "other",
	"manifest_version": 2
}
```

Now load your extension by adding this line to `LocalSettings.php`:

```
wfLoadExtension( 'TutorialWorld' );
```

Navigate to Special:Version and you'll see your new extension in the installed extensions list.



Installed extensions

Other				
Extension	Version	License	Description	Authors
BoilerPlate	– (b54c699) 13:56, 14 April 2022	GPL-2.0-or-later	This is an example extension	Your Name
TuitorialWorld	1.0	GPL-2.0-or-later	A fancy new extension	The Tuitorial King

Inside BoilerPlate

The `extension.json` file for BoilerPlate specifies what the extension is, how it works, and how it connects to MediaWiki. Learn more about the specification at https://www.mediawiki.org/wiki/Manual:Extension.json/Schema.

Metadata

We see this data on the Special:Version page, among other places:

```
{
    "name": "BoilerPlate",
    "author": [
        "Your Name"
    ],
    "url": "https://www.mediawiki.org/wiki/Extension:BoilerPlate",
    "descriptionmsg": "boilerplate-desc",
    "license-name": "GPL-2.0-or-later",
    "type": "other",
    "requires": {
        "MediaWiki": ">= 1.35.0"
    },
```

Location of code to be loaded

MediaWIki will instruct PHP to load classes from the directory you specify:

```
    "AutoloadNamespaces": {
        "MediaWiki\\Extension\\BoilerPlate\\": "src/"
    },
```

Configuring the extension

The BoilerPlate extension exposes two feature toggles:

```
    "config": {
        "BoilerPlateEnableFoo": {
            "description": "Enable the Foo feature.",
            "value": true
        },
        "BoilerPlateVandalizeEachPage": {
            "description": "Write 'BoilerPlate was here' on each page",
            "value": false
        }
    },
```

Hook handlers and hook usage

Hooks allow custom code to be executed when a specific event happens. BoilerPlate uses `BeforePageDisplay`, which allows the extension to make last-minute changes to the output page, such as adding CSS or JavaScript.

```
    "HookHandlers": {
        "BoilerPlateHooks": {
            "class": "MediaWiki\\Extension\\BoilerPlate\\Hooks"
        }
    },
    "Hooks": {
        "BeforePageDisplay": "BoilerPlateHooks"
    },
```

Internationalization (i18n) files

This section specifies translations of the messages used by the extension, for use by MediaWiki's internationalization system:

```
5 "MessagesDirs": {
    "BoilerPlate": [
      "i18n"
    ]
  },
```

Front-end resources

This section defines UI resources, for use by MediaWiki's resource system:

```
6 "ResourceModules": {
    "ext.boilerPlate": {
      "localBasePath": "resources/ext.boilerPlate",
      "remoteExtPath": "BoilerPlate/resources/ext.boilerPlate",
      "dependencies": [],
      "styles": [],
      "packageFiles": [
        "init.js"
      ],
      "messages": []
    }
  },
```

Registration manifest version

```
7 "manifest_version": 2
}
```

Adding the `BeforePageDisplay` hook

One of the features of the BoilerPlate extension is the ability to add text to the bottom of every page on the wiki (thus “vandalizing” them).

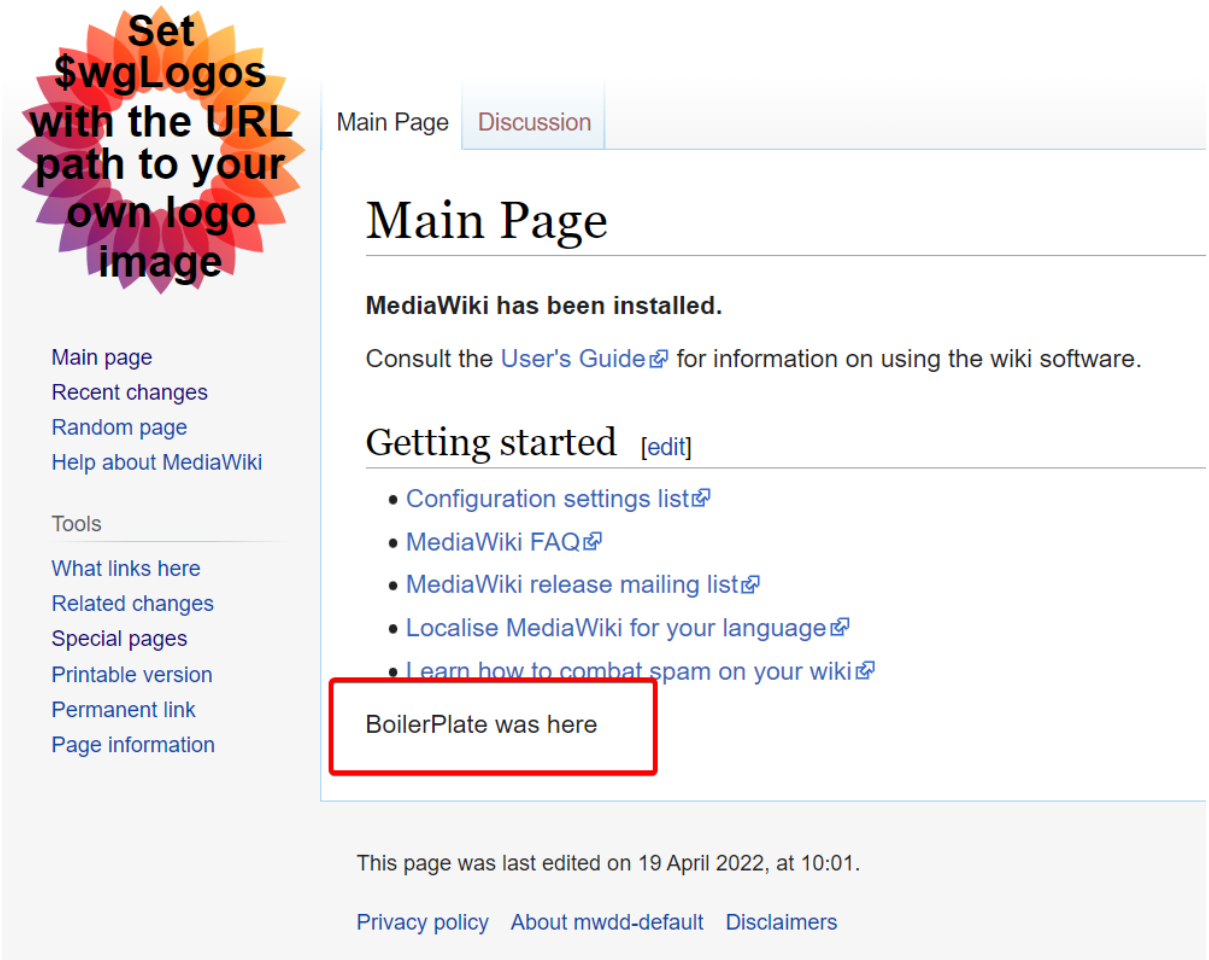
Enable this by changing the `wgBoilerPlateVandalizeEachPage` setting to true in your `LocalSettings.php`:

```
$wgBoilerPlateVandalizeEachPage = true;
```

You can see this setting defined in BoilerPlate’s `extension.json` file.

Now reload your wiki’s main page to observe how the extension added text to the end of the page.

Note: If you don’t see the text right away, consider disabling request caching in your browser, via either your browser’s preferences or the developer console (F12: Network: check “Disable caching”).



Hooks

You can see the code that makes this happen by looking at `extensions/BoilerPlate/src/Hooks.php`.

This code defines a namespaced class called `Hooks`; its full name, including namespace, is `MediaWiki\Extension\BoilerPlate\Hooks`. The class implements a Hook interface from MediaWiki called `BeforePageDisplayHook`, which defines an interface for the single function implemented in the class called `onBeforePageDisplay`.

When called, the hook checks an extension configuration variable. If true, the hook adds the `oojs-ui-core` module defined by MediaWiki and adds some HTML to the bottom of the page.

```php
<?php
/** ...

namespace MediaWiki\Extension\BoilerPlate;

Timo Tijhof, 3 months ago | 5 authors (Kosta Harlan and others) | 3 references | 0 implementations
class Hooks implements \MediaWiki\Hook\BeforePageDisplayHook {

    /**
     * @see https://www.mediawiki.org/wiki/Manual:Hooks/BeforePageDisplay
     * @param \OutputPage $out
     * @param \Skin $skin
     */
    3 references | 0 overrides
    public function onBeforePageDisplay( $out, $skin ): void {
        $config = $out->getConfig();
        if ( $config->get( 'BoilerPlateVandalizeEachPage' ) ) {
            $out->addModules( 'oojs-ui-core' );
            $out->addHTML( \Html::element( 'p', [], 'BoilerPlate was here'
        }
    }

}
```

Jumping back to `extension.json`, you can see how this function connects to MediaWiki and the hook system.

1. A HookHandler is registered with the name `BoilerPlateHooks` that points at the Hooks class mentioned above.

2. The function registers the specific hook `BeforePageDisplay` which points to the hook handler `BoilerPlateHooks`:

```
25      },
26  4   "HookHandlers": {
27          "BoilerPlateHooks": {
28              "class": "MediaWiki\\Extension\\BoilerPlate\\Hooks"
29          }
30      },
31      "Hooks": {
32          "BeforePageDisplay": "BoilerPlateHooks"
33      },
```

The flow looks something like this:

MediaWiki code ->
named hook ->
hook handler ->
Hooks class ->
hook interface method ->
Extension code

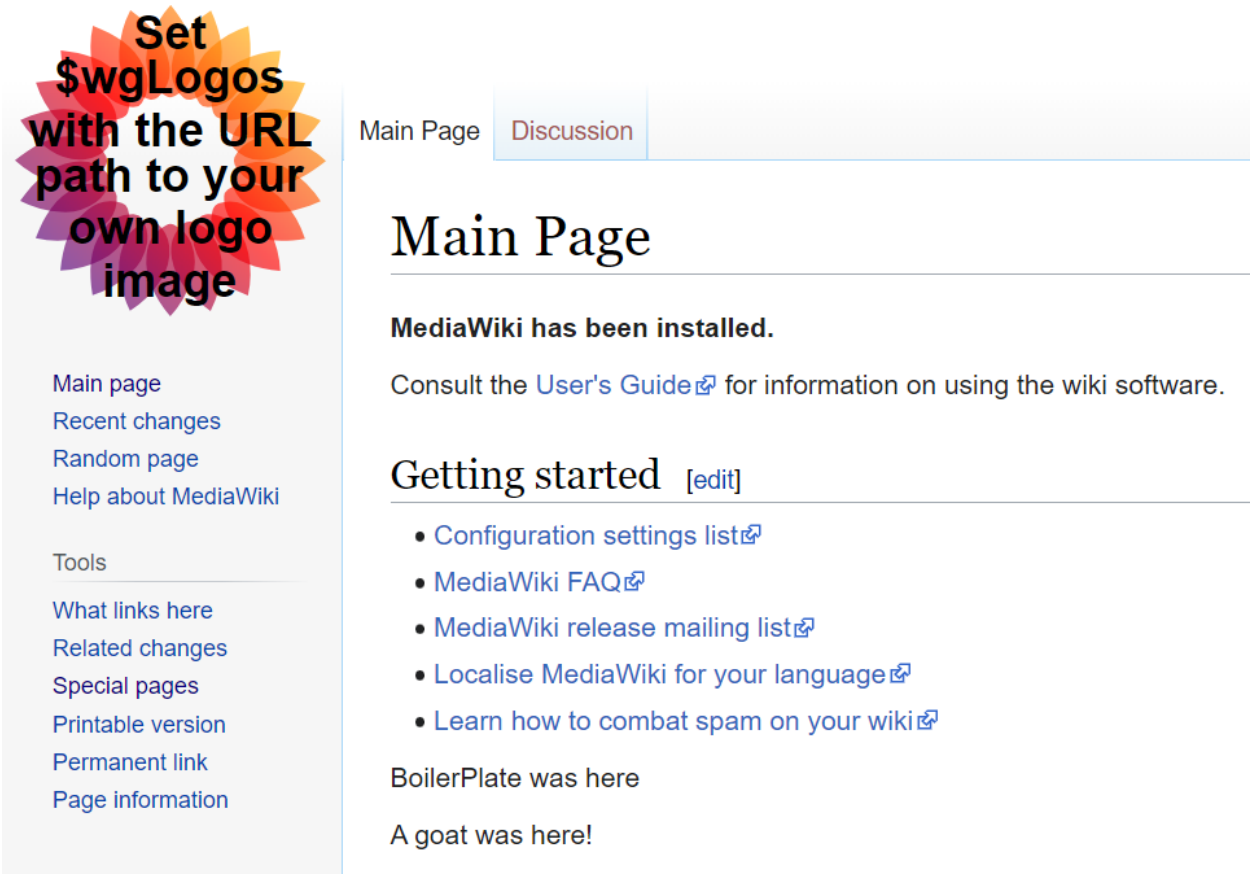
Implementing `BeforePageDisplay`

Now that you've seen an example of the `BeforePageDisplay` hook in use, it's time to use it to modify your own TutorialWorld extension and have it add text of your own choosing to every page.

Use the BoilerPlate extension files for inspiration.

1. Create a `src` directory in the TutorialWorld extension directory.
2. Register the `AutoloadNamespaces` for the extension in `extension.json`.
3. Create a new class implementing the `BeforePageDisplayHook` interface in the `src` directory. (Use different text!)
4. Register the class as a `HookHandler` in `extension.json`.
5. Register the `Hook` in `extension.json`.

As an extra challenge, try to make the text easily configurable. Your resulting page might look something like this (note the goat):



Alternate ways to extend MediaWiki

Hooks are one of the major ways to extend MediaWiki. MediaWiki itself offers many hooks https://www.mediawiki.org/wiki/Manual:Hooks#Available_hooks, with many more hooks implemented in extensions.

Other common extension methods include:

- API endpoints
    - Action API: https://www.mediawiki.org/wiki/API:Extensions
    - REST API: https://www.mediawiki.org/wiki/API:REST_API/Extensions
- Special pages https://www.mediawiki.org/wiki/Manual:Special_pages

Using these alternate methods involves a similar process to using hooks:

- Read the MediaWiki documentation on the extension mechanism.
- Register the mechanism in `extension.json` in the appropriate place.
- Create code that works with the given extension mechanism.

General notes

- Some docker images don't work well with ARM CPUs.
- Installing the docker engine doesn't install docker-compose by default (thus `hello world` will work, but starting with `mwcli` will complain)
- At the time of this writing, finding the right hooks is more of an art than a science.

Wikibase for local development

At the end of this workshop, your challenge will be to develop some code that extends Wikibase. The first step is to install Wikibase into your local development environment.

You'll likely find this very similar to the process of fetching the BoilerPlate extension above.

```
cd $(mw docker mediawiki where)/extensions
git clone
"https://gerrit.wikimedia.org/r/mediawiki/extensions/Wikibase"
```

Wikibase has composer dependencies that you can download with composer-merge-plugin.

Create a file in the mediawiki/core directory called `composer.local.json` with the following contents:

```
{
    "extra": {
        "merge-plugin": {
            "include": [
                "extensions/*/composer.json",
                "skins/*/composer.json"
            ]
        }
    }
}
```

These lines are already in a sample file, so you can simply copy it into place:

```
cd $(mw docker mediawiki where)
cp composer.local.json-sample composer.local.json
```

Now update composer's dependencies.

```
mw docker mediawiki composer update
```

```
> ComposerHookHandler::onPreUpdate
Loading composer repositories with package information
Warning from https://repo.packagist.org: Support for Composer 1 is deprecated and some packages will not
deprecating-composer-1-support/
Info from https://repo.packagist.org: #StandWithUkraine
Updating dependencies (including require-dev)
Package operations: 12 installs, 0 updates, 0 removals
  - Installing composer/installers (v1.12.0): Downloading (100%)
  - Installing data-values/interfaces (1.0.0): Downloading (100%)
  - Installing data-values/data-values (3.0.0): Downloading (100%)
  - Installing data-values/common (1.0.0): Downloading (100%)
  - Installing data-values/time (1.0.4): Downloading (100%)
  - Installing diff/diff (3.2.0): Downloading (100%)
  - Installing psr/simple-cache (1.0.1): Downloading (100%)
  - Installing serialization/serialization (4.0.0): Downloading (100%)
  - Installing data-values/serialization (1.2.4): Downloading (100%)
  - Installing data-values/geo (4.3.0): Downloading (100%)
  - Installing data-values/number (0.11.1): Downloading (100%)
  - Installing onoi/message-reporter (1.4.2): Downloading (100%)
Package phpunit/php-token-stream is abandoned, you should avoid using it. No replacement was suggested.
Writing lock file
Generating optimized autoload files
composer/package-versions-deprecated: Generating version class...
composer/package-versions-deprecated: ...done generating version class
46 packages you are using are looking for funding.
Use the `composer fund` command to find out more!
> ComposerVendorHtaccessCreator::onEvent
```

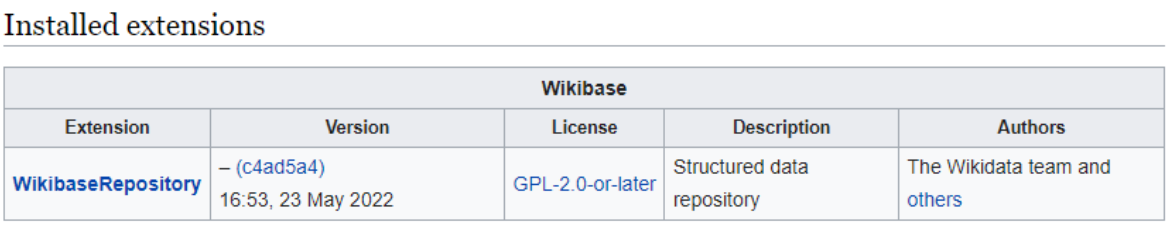
Once the dependencies are updated, load the extension in `LocalSettings.php` file by adding the following lines (which load only the repository portion of Wikibase):

```
## Wikibase Repository
wfLoadExtension( 'WikibaseRepository',
"$IP/extensions/Wikibase/extension-repo.json" );
require_once "$IP/extensions/Wikibase/repo/ExampleSettings.php";
```

Now run `update.php`:

```
mw docker mediawiki exec -- php maintenance/update.php --quick
```

You can now navigate to the Special:Version page to see Wikibase installed there:



Installed extensions

Wikibase				
Extension	Version	License	Description	Authors
WikibaseRepository	– (c4ad5a4) 16:53, 23 May 2022	GPL-2.0-or-later	Structured data repository	The Wikidata team and others

Create a new item using Special:NewItem to ensure everything is working correctly.

Stable interface policies

Several of Wikidata's stable interface policies are relevant to MediaWiki and Wikibase development. (The Wikidata stable interface policy was written primarily with Wikidata in mind and does not make much explicit mention Wikibase.)

Key points:

- Wikidata/Wikibase stable interface policy
    - Changes of stable interfaces (such as public APIs) shall be communicated
    - Things that are considered stable:
        - JSON and RDF output
        - Wikibase Web API (`api.php`)
        - Linked data interface (Special:EntityData)
        - Wikidata Query Service
        - Wikibase LUA Library
        - JavaScript hooks, documented in hooks-js.md
    - Some things that are considered unstable:
        - Raw Wikibase content stored by MediaWiki and returned by the MediaWiki core API
        - Wikibase PHP code
        - Wikibase JavaScript code
        - The HTML DOM structure
- MediaWiki stable interface policy
    - Using code
        - It is generally stable to call public methods on a class instance.
        - It is generally not stable to construct (instantiate) a class.
        - It is generally not stable to extend a class (subclass), nor to implement an interface.
    - A documented deprecation and removal process exists.
    - A documented, recognized definition of an extension "ecosystem" exists.

Wikibase PHP hooks

Consult the developer documentation for Wikibase PHP hooks.

Current repository PHP hooks:

- Addition of DataTypes and EntityTypes
    - **WikibaseRepoDataTypes**
        - Example usages:
            - Math
            - Score

                - WikibaseEdtf
                - WikibaseLocalMedia
        - **WikibaseRepoEntityTypes**
            - Example usages adding entity types:
                - WikibaseLexeme
                - WikibaseMediaInfo
            - Example usages modifying entities (altering search behavior):
                - WikibaseCirrusSearch
                - WikibaseLexemeCirrusSearch
- Generally, additional EntityTypes specify a default namespace/slot for their content.
    - **WikibaseRepoEntityNamespaces**
        - Examples:
            - WikibaseMediaInfo (in a dedicated slot in the file namespace)
            - WikibaseLexeme (in the configured LexemeNamespace, if enabled)
- **WikibaseChangeNotification**
    - Example usage:
        - WikibaseQualityConstraints (schedules jobs after entity edits)
- **WikibaseContentLanguages**
    - Example usage:
        - WikibaseLexeme
- **GetEntityContentModelForTitle**
    - Example usage:
        - WikibaseMediaInfo
- **WikibaseRepoOnParserOutputUpdaterConstruction**
    - Allows extensions to register extra EntityParserOutputUpdater implementations.
    - Example usage:
        - WikibaseLexeme
- **GetEntityByLinkedTitleLookup**
    - Allows extensions to add custom EntityByLinkedTitleLookup services.
    - Example usage:
        - WikibaseMediaInfo

Walkthrough: WikibaseManifest

WikibaseManifest is an extension that summarizes metadata about a Wikibase installation and exposes it as a simple API. The goal is to help toolmakers write tools that can target any Wikibase.

Taking a look at `extension.json`:

The extension makes use of:

- Three configuration options
- Code is located in the `includes` directory and `MediaWiki\Extension\WikibaseManifest` namespace

- A single REST API route is registered at `/wikibase-manifest/v0/manifest`
- MediaWiki's i18n/localization system
- MediaWiki's service registration system

The extension consists of over 20 files that contain code.

Some Wikibase services and configurations are used:
- RdfVocabulary
    - Allows reporting the RDF URIs used, via the API
- LocalEntitySource
    - Allows reporting of the namespaces that entities exist in locally

You can see some example API output here:
https://addshore-alpha.wiki.opencura.com/w/rest.php/wikibase-manifest/v0/manifest

Walkthrough: WikibaseCirrusSearch

This extension implements Elasticsearch-based search functionality for default Wikibase entities (Items and Properties). It replaces the default SearchHelper with a SearchHelper that consults an Elasticsearch instance.

Looking at `extension.json`:

This extension makes use of:
- Twelve configuration options
- Code is located in the `src` directory and `Wikibase\Search\Elastic` namespace
- Nine hooks are used (one from Wikibase)
- MediaWiki i18n/localization system

The single Wikibase hook that's used overrides some registered entities' fields. Altered fields can be found in WikibaseSearch.entitytypes.php.
- Def::ENTITY_SEARCH_CALLBACK
- Def::SEARCH_FIELD_DEFINITIONS
- Def::FULLTEXT_SEARCH_CONTEXT

These fields relate to the hook used and are documented in the entity types documentation.

Walkthrough: AutomatedValues

This is a Wikibase extension through which a user can define rules that automatically set labels or aliases based on Statement values.

Looking at `extension.json`:

The extension makes use of three configuration options and five hooks. The code is located in the `src` directory and the `ProfessionalWiki\\AutomatedValues` namespace, and it employs the MediaWiki resource loader system.

The hook that enables the main functionality acts onMultiContentSave and gets the content being saved. If that content is an entity with Statements, it applies the given rules to the content being saved.

Your turn!

In the live version of this workshop, the developers led participants through peer/mob programming a bulk entity creation REST API. This is a process that needs to be experienced live, but here are the participant notes for that section if you'd like to continue on your own.

Documentation:

- https://www.mediawiki.org/wiki/API:REST_API/Extensions#Defining_routes
- https://www.mediawiki.org/wiki/API:REST_API/Extensions#Handling_requests
- https://www.mediawiki.org/wiki/Manual:Extension.json/Schema#AvailableRights
- https://www.mediawiki.org/wiki/Manual:Extension.json/Schema#GroupPermissions

Relevant Wikibase code points:

- EntityStore
  - A storage service that will work for ALL registered entity types
  - Retrieval: Wikibase\Repo\WikibaseRepo::getEntityStore
  - Service: WikibaseRepo.EntityStore
  - Interface: EntityStore
- EntityDeserializer
  - A deserializer object (JSON -> PHP object) that will work for ALL registered entity types
  - Retrieval: Wikibase\Repo\WikibaseRepo::getAllTypesEntityDeserializer
  - Service: WikibaseRepo.AllTypesEntityDeserializer
  - Interface: DispatchableDeserializer & Deserializer

Note: This is PHP code, so it is unstable per the current stable interface policy (see above).

Planned user input

The user should be able to submit a list of entities to be created by the API in the BODY of a POST request to a `batchcreate` route.

The request body might look something like this:

```
{
  "entities":[
    {
      "type": "item",
      "labels": {
```

```
        "en": { "language": "en", "value": "hello goat" }
      }
    },
    {
      "type": "item",
      "labels": {
        "en": { "language": "en", "value": "potato land" }
      }
    }
  ]
}
```

(Entity format is the same as for `wbeditentity`)

A response might look something like this:

```
[
    {
      "id": "Q1",
      "rev": 2
    },
    {
      "id": "Q2",
      "rev": 3
    }
]
```

(In case of error, return an error key for that entity instead)

Goals

- Register a route.
- Implement code for the route that responds with some hardcoded JSON.
- Handle input from the user (echo it back).
- Deserialize a list of entities submitted by the user in the body of the request and respond with how many were received.
- Create the entities the user submits.

And for extra credit:

- Require permissions for API usage.
- Limit the number of entities that can be created in a request to 100.
- Use a factory for the registration of the handler.