Typed Data API

Let’s try to understand it. Finally …

Piotr Pakulski

Piotr Pakulski

FLORIDA DRUPAL CAMP

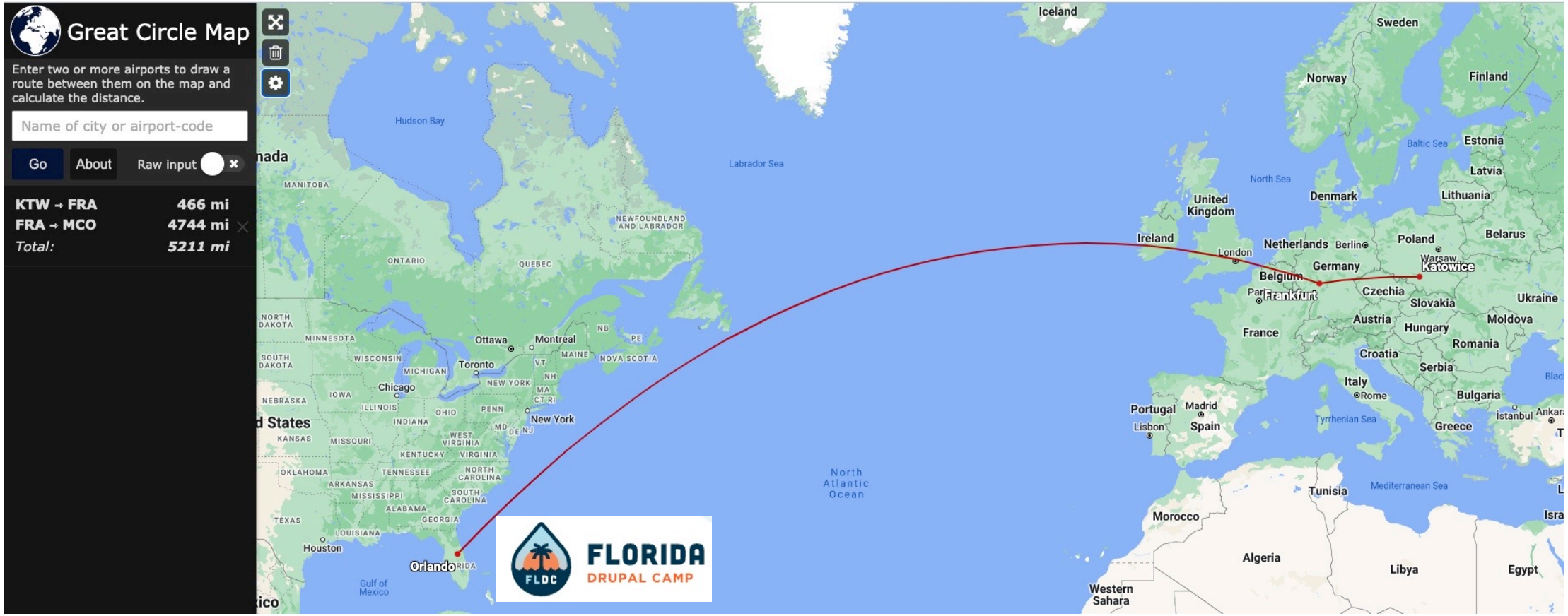
- I am from Poland
- Drupal Developer at BOX.com
- I am PHP developer for around 12+ years now
- Working with Drupal starting from late version 7
- Before Drupal used to be Symfony 2.x developer

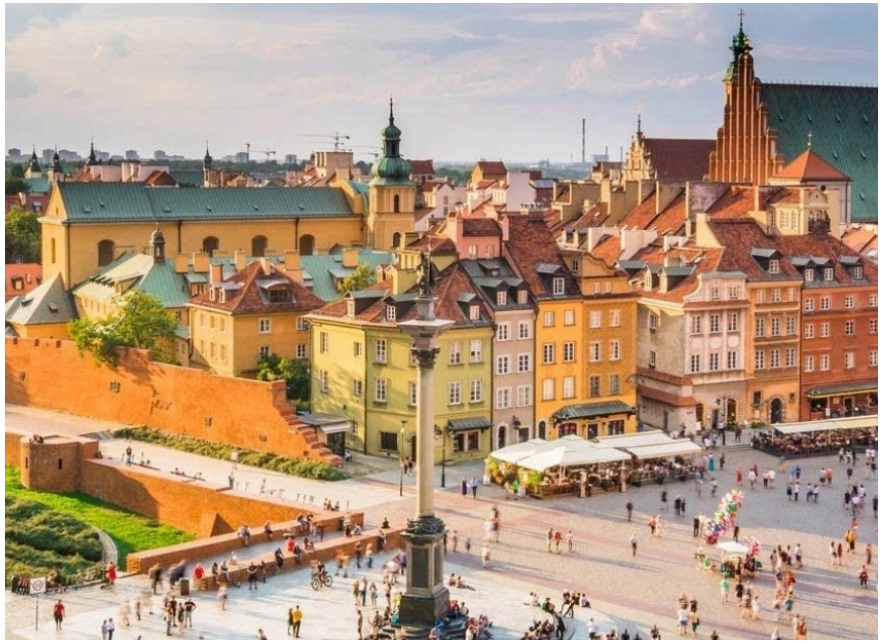
https://www.drupal.org/u/piotr-pakulski
https://www.linkedin.com/in/pakulski-piotr/

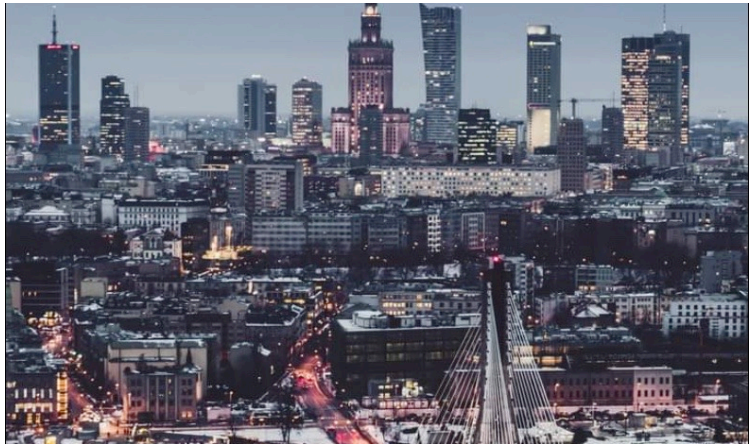
Developer under the Car
@developerunderthecar1479
81 subskrybentów

Drupal Association Individual Member

Thank you FLDC !

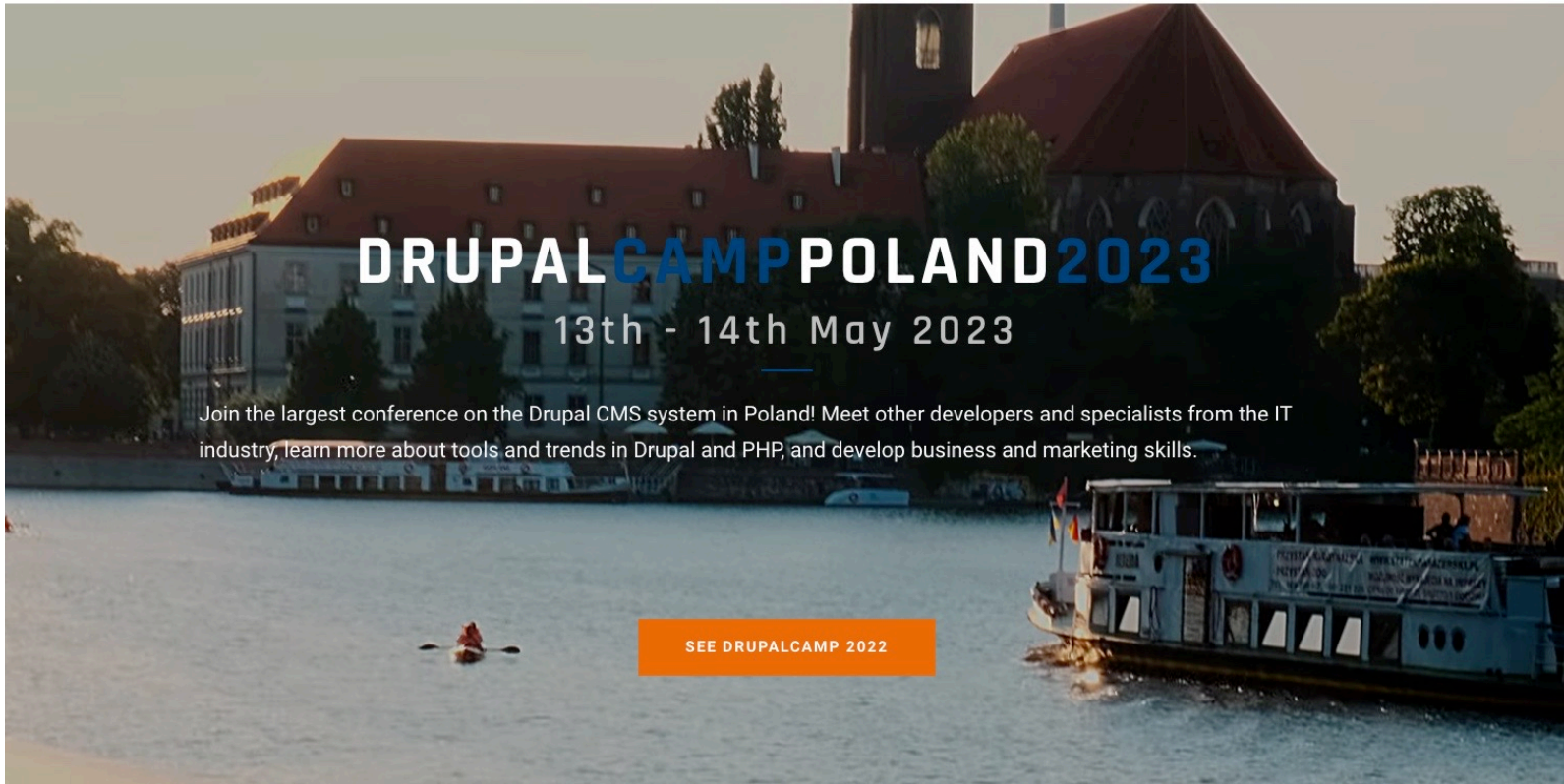






DRUPALCAMP POLAND/2022

Tickets Venue Proposed Sessions Sponsors EN



The agenda

Why?

How?

Content entities

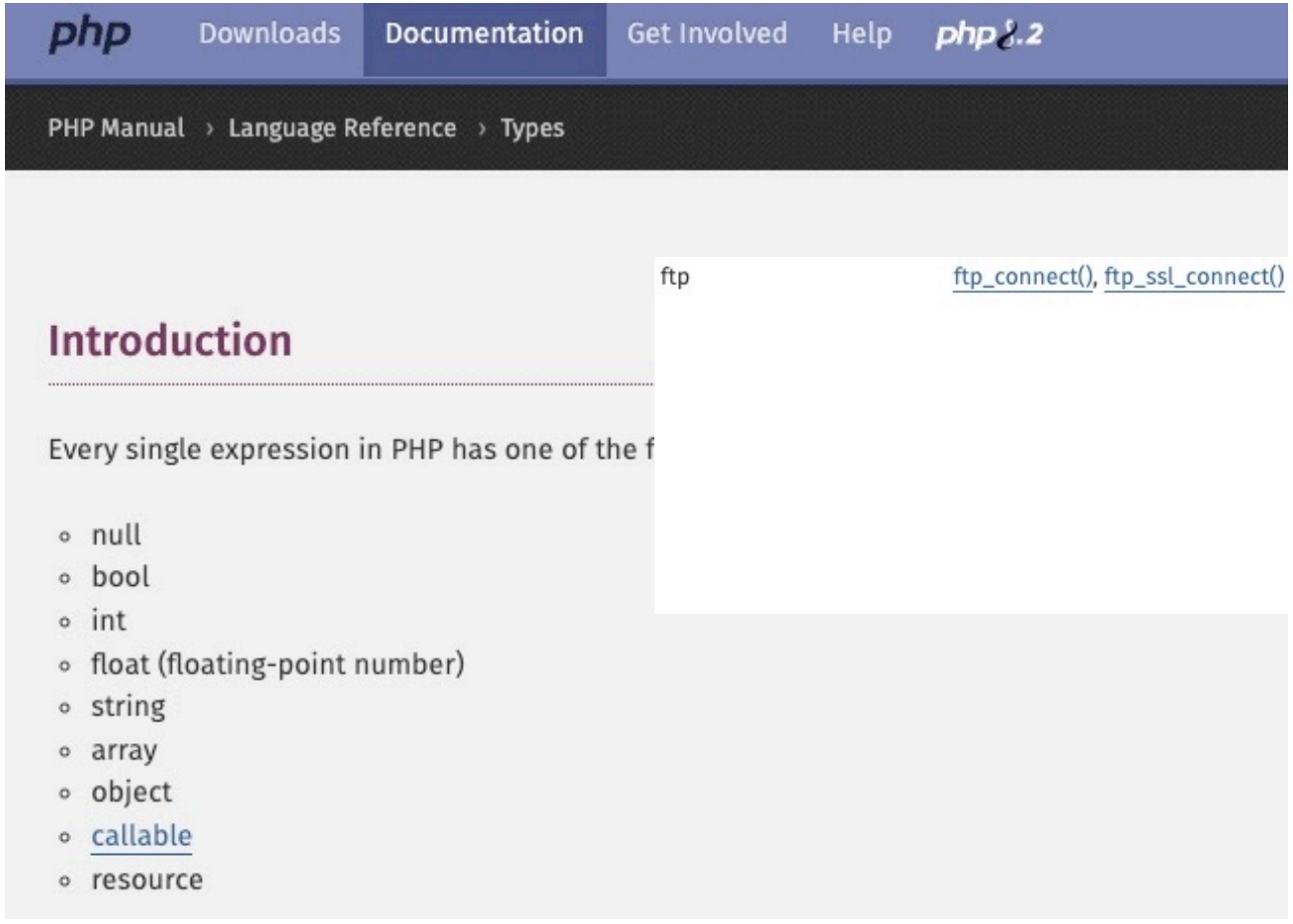
Use cases

Questions

How good you know PHP ?

FLORIDA DRUPAL CAMP

Anyone knows how many variable build-in types PHP has?



ftp	ftp_connect(), ftp_ssl_connect()	ftp_close()	FTP connection (prior to PHP 8.1.0)

ftp_login(), ftp_pwd(), ftp_cdup(), ftp_chdir(), ftp_mkdir(), ftp_rmdir(), ftp_nlist(), ftp_rawlist(), ftp_systype(), ftp_pasv(), ftp_get(), ftp_fget(), ftp_put(), ftp_fput(), ftp_size(), ftp_mdtm(), ftp_rename(), ftp_delete(), ftp_site(), ftp_alloc(), ftp_chmod(), ftp_exec(), ftp_get_option(), ftp_nb_continue(), ftp_nb_fget(), ftp_nb_fput(), ftp_nb_get(), ftp_nb_put(), ftp_raw(), ftp_set_option()

```php
// An example callback function
function my_callback_function() {
    echo 'hello world!';
}

call_user_func('my_callback_function');

// Type 2: Static class method call
call_user_func(array('MyClass', 'myCallbackMethod'));
```

How good you know PHP ?

FLORIDA DRUPAL CAMP

PHP is a dynamically typed language

Strict comparisons with ===

	true	false	1	0	-1	"1"	"0"	"-1"	null	[]	"php"	""
true	true	false	false	false	false	false	false	false	false	false	false	false
false	false	true	false	false	false	false	false	false	false	false	false	false
1	false	false	true	false	false	false	false	false	false	false	false	false
0	false	false	false	true	false	false	false	false	false	false	false	false
-1	false	false	false	false	true	false	false	false	false	false	false	false
"1"	false	false	false	false	false	true	false	false	false	false	false	false
"0"	false	false	false	false	false	false	true	false	false	false	false	false
"-1"	false	false	false	false	false	false	false	true	false	false	false	false
null	false	false	false	false	false	false	false	false	true	false	false	false
[]	false	false	false	false	false	false	false	false	false	true	false	false
"php"	false	false	false	false	false	false	false	false	false	false	true	false
""	false	false	false	false	false	false	false	false	false	false	false	true

Why we need TypedData?

FLORIDA
DRUPAL CAMP

- We can not realy of the type of certain data
- 1 === “1” is not TRUE
- so we either use == which is not good
- Let’s examine **1495875076** and **24958770776**
- However, in some cases, it is useful to be able to define an abstract type

History - the problem with Drupal 7

- No consistent way defining data type
- Can not easly say if field is translatable
- Access levels hard to define
- Validaton problems – famous hook_filed_validate
- Hard to build machine readable API for data access

More history background

https://www.drupal.org/project/drupal/issues/1696640

FLORIDA DRUPAL CAMP

Implement API to unify entity properties and fields

Overview

This is the first issue for implementing an entity property API as discussed during the WSCCI Web Services Format Sprint (report) and first proposed at the core conversations in London. In huge parts its about decoupling Field API systems and taking them to entity properties - so one could call it also “re-factor Field API”.

Why another API??

While introducing a separate lower-level API for the sake of describing entity properties might look like overkill, it turns out to have many more uses than that. The following use-cases have been identified:

fago German Vienna Credit *commented 10 years ago* #32

After more discussions me and dixon_ came up with the following terminology suggestions:

- Name the lower-level API "TypedData" instead of Property/Metadata/Data API.
- We realized that the objects we are working with really always are data wrappers, even in the case of properties as they wrap the plain property values. Thus, we are thinking that for the generic "PropertyInterface/ItemInterface" we better wanna go with DataWrapperInterface. All together, that makes the following suggestion:

Property\PropertyListInterface -> TypedData\DataListInterface
Property\PropertyContainerInterface -> TypedData\DataContainerInterface
Property\PropertyInterface -> TypedData\DataWrapperInterface

PropertyEntity -> EntityWrapper
PropertyString -> String
PropertyInteger -> Integer

Issues

Add a TypedData API for allowing metadata to be associated with data

Spin-off from ~~#1696640: Implement API to unify entity properties and fields~~. This is just the TypedData portion of that issue's work, presented here on its own, because it is useful for ~~#1648930: Introduce configuration schema and use for translation~~ and #1696302: [meta] Blocks/Layouts everywhere in addition to that issue.

Closed (fixed)	
Project:	Drupal core
Version:	8.0.x-dev

Typed data API

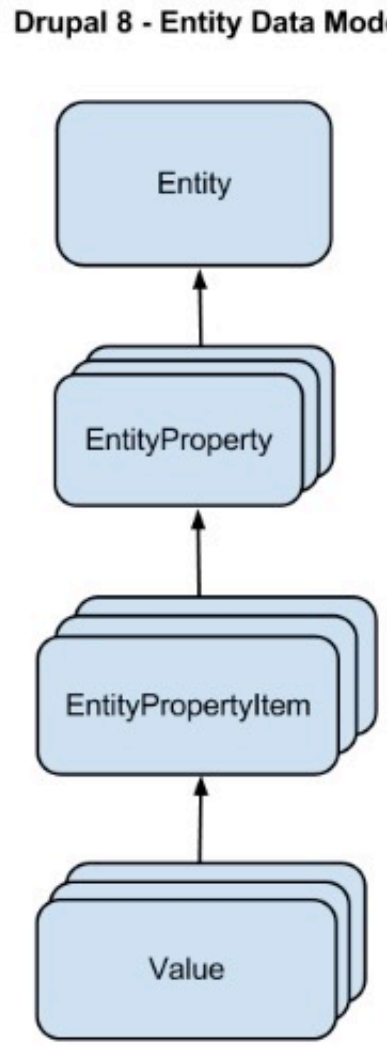
The TypedData API data model consists of data lists, complex data and data primitives, whereas a list is a sequential list of data and complex data contains any number of data properties. Data definitions describe data based upon defined data types, which may be primitives or any module defined data type (as e.g. a field type or an entity). Added data types may be mapped to the built-in primitives which are predefined, e.g. an e-mail type would be mapped to the string primitive (but could add further validation logic). This allows modules to operate with any kind of typed data just by supporting complex data, data lists and the built-in primitives, such as needed when serializing entities to certain formats. In this case, it saves us from having to do n*m integrations between module-added data types and serialization formats or other "data consumers".

The list of pre-defined primitives has been created by looking at other systems like PHPCR, XML-Schema, the d7 entity property api and SDL.

Built-in primitive data types

Data type	Machine name	Plain value	Further value formats (may be used when setting)
String	string	PHP string	any PHP variable that casts well to a string
Boolean	boolean	PHP boolean	any PHP variable that casts well to a boolean
Integer	integer	PHP integer	any PHP variable that casts well to an integer
Float	float	PHP float	any PHP variable that casts well to a float
Date	date	DateTime object	any string supported by DateTime::__construct(), or a timestamp as integer
Duration	duration	DateInterval object	a ISO8601 string as supported by DateInterval::__construct, or an integer in seconds
URI	uri	PHP string (absolute URI)	-
Binary	binary	PHP file resource	absolute stream resource URI as string

Wellcome Typed Data API in D8



Comparison to Drupal 7

$node

$node->field_image,
$node->field_body,
$node->title,
...

$node->field_image[en][0],
$node->field_image[en][1],
...

$node->field_image[en][0]['fid'],
$node->field_image[en][0]['alt'],
...

TypedData on Drupal.org

FLORIDA
DRUPAL CAMP

Typed Data API

Edit

Typed Data API in Drupal 8.

Typed Data API overview

Overview of Typed Data API in Drupal 8.

Data type plugins

Describes the concept of a Data Type in Drupal

Data definitions

Describes the Data Definition concept in Drupal

Computed properties (TBD)

Initial stub page, just to outline the different child-pages we need.

Report as spam

TypedData on Drupal.org

FLORIDA DRUPAL CAMP

Typed Data API overview

Last updated on 14 April 2020

Edit

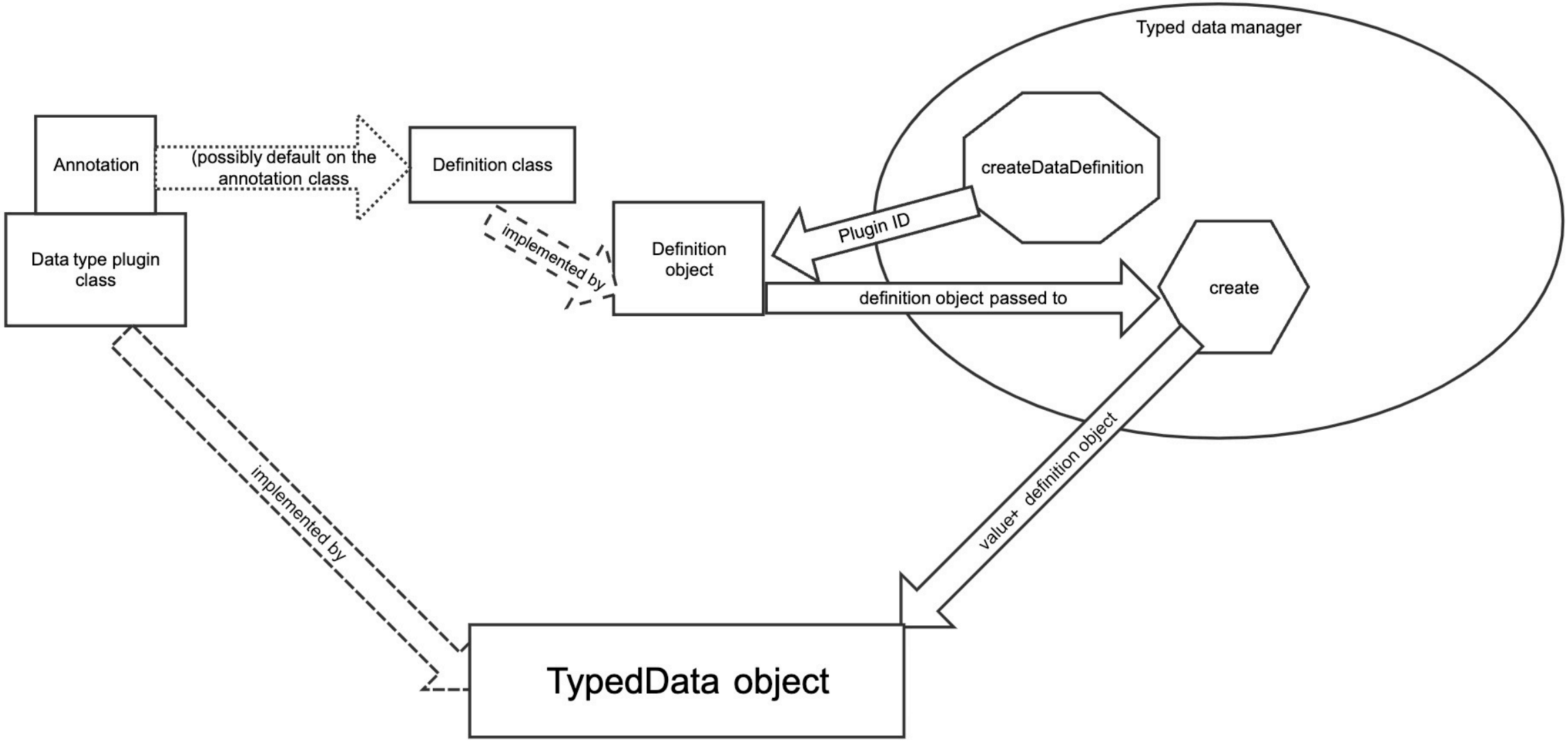
Overview

The Typed Data API was created to provide developers with a consistent way of interacting with data in different ways. Not only does the API allow you to interact with the actual data, it also provides means of fetching more information, or metadata, about the actual data.

The Typed Data API is a low level, generic and reusable object oriented API that appears at multiple levels of the Drupal 8 architecture. Take for example, the EntityAdapter, which extends TypedData and acts as a wrapper for an Entity. Or FieldItemBase, which is an unwrapped extension of TypedData.

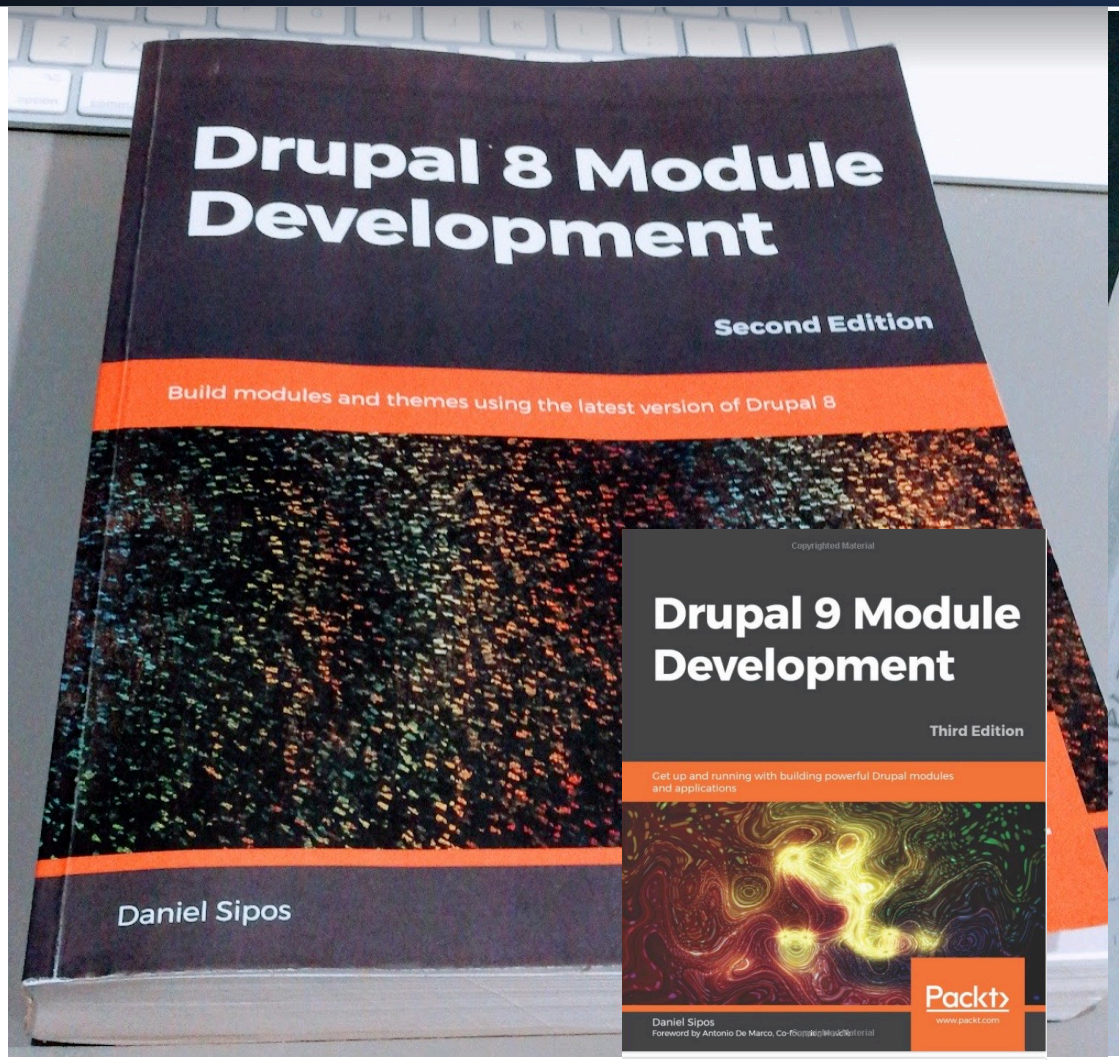
On this page

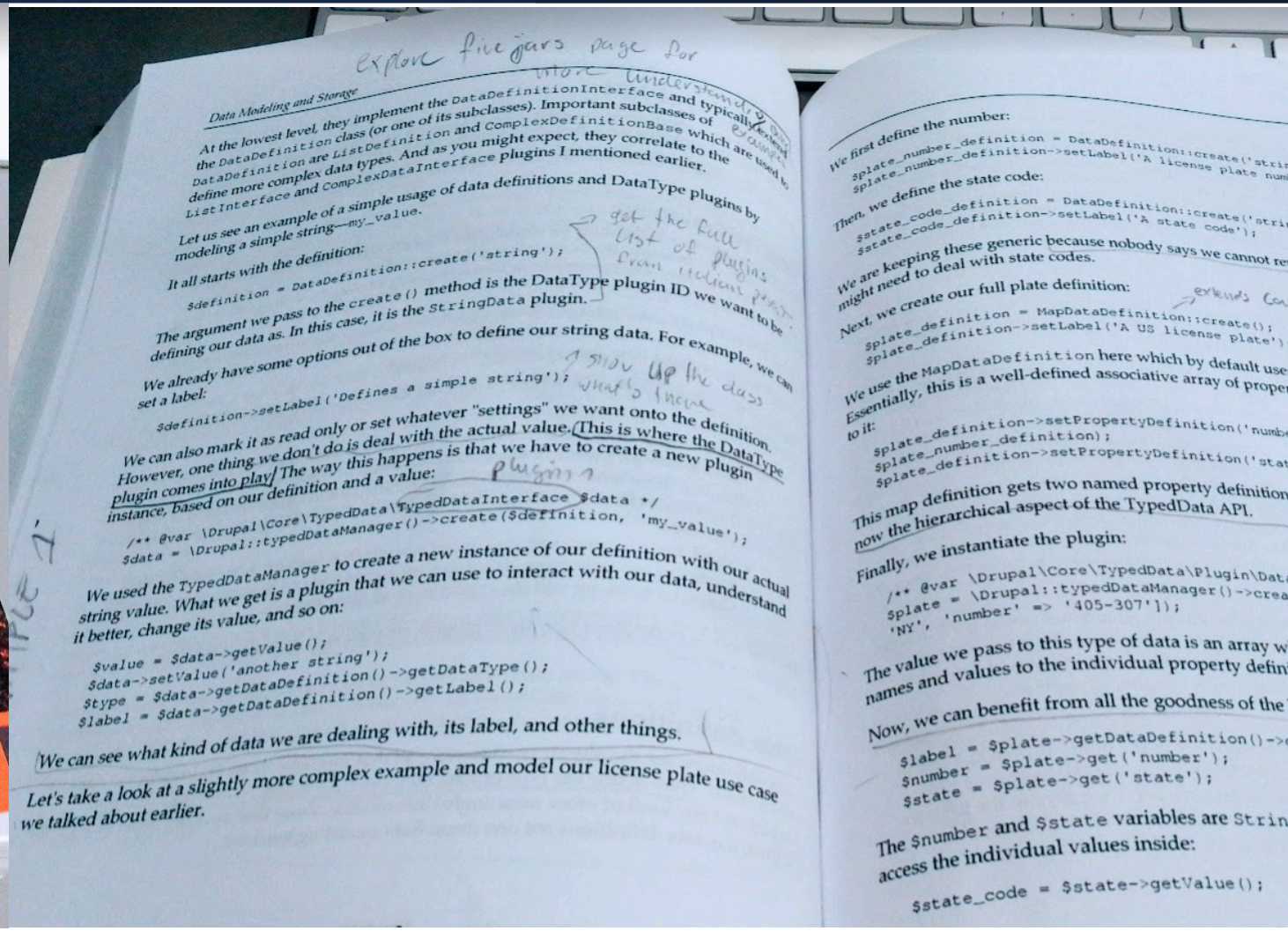
TypedData on Drupal.org



TypedData by Daniel Sipos

FLORIDA DRUPAL CAMP





Data Modeling and Storage

At the lowest level, they implement the `DataDefinitionInterface` *and typically extend the* `DataDefinition` *class (or one of its subclasses). Important subclasses of* `DataDefinition` *are* `ListDefinition` *and* `ComplexDefinitionBase` *which are used to define more complex data types. And as you might expect, they correlate to the* `ListInterface` *and* `ComplexDataInterface` *plugins I mentioned earlier.*

Let us see an example of a simple usage of data definitions and DataType plugins by modeling a simple string—`my_value`*.*

It all starts with the definition:

```
$definition = DataDefinition::create('string');
```

The argument we pass to the `create()` *method is the DataType plugin ID we want to be defining our data as. In this case, it is the* `StringData` *plugin.*

We already have some options out of the box to define our string data. For example, we can set a label:

```
$definition->getLabel('Defines a simple string');
```

We can also mark it as read only or set whatever "settings" we want onto the definition. However, one thing we don't do is deal with the actual value. This is where the DataType plugin comes into play. The way this happens is that we have to create a new plugin instance, based on our definition and a value:

```
/** @var \Drupal\Core\TypedData\TypedDataInterface $data */
$data = \Drupal::typedDataManager()->create($definition, 'my_value');
```

We used the `TypedDataManager` *to create a new instance of our definition with our actual string value. What we get is a plugin that we can use to interact with our data, understand it better, change its value, and so on:*

```
$value = $data->getValue();
$data->setValue('another string');
$type = $data->getDataDefinition()->getDataType();
$label = $data->getDataDefinition()->getLabel();
```

We can see what kind of data we are dealing with, its label, and other things.

Let's take a look at a slightly more complex example and model our license plate use case we talked about earlier.

We first define the number:

```
$plate_number_definition = DataDefinition::create('strin
$plate_number_definition->setLabel('A license plate num
```

Then, we define the state code:

```
$state_code_definition = DataDefinition::create('strin
$state_code_definition->setLabel('A state code');
```

*We are keeping these generic because nobody says we cannot re
might need to deal with state codes.*

Next we create our full plate definition:

```
$plate_definition = MapDataDefinition::create();
$plate_definition->setLabel('A US license plate');
```

We use the `MapDataDefinition` *here which by default use
Essentially, this is a well-defined associative array of prope
to it:*

```
$plate_definition->setPropertyDefinition('numb
$plate_number_definition);
$plate_definition->setPropertyDefinition('stat
$plate_definition);
```

*This map definition gets two named property definition
now the hierarchical aspect of the TypedData API.*

Finally, we instantiate the plugin:

```
/** @var \Drupal\Core\TypedData\Plugin\Dat
$plate = \Drupal::typedDataManager()->crea
'NY', 'number' => '405-307']);
```

*The value we pass to this type of data is an array w
names and values to the individual property defin*

Now, we can benefit from all the goodness of the

```
$label = $plate->getDataDefinition()->
$number = $plate->get('number');
$state = $plate->get('state');
```

The `$number` *and* `$state` *variables are* `Strin
access the individual values inside:*

```
$state_code = $state->getValue();
```

What is TypedData ?

FLORIDA
DRUPAL CAMP

- First: it wraps values of any kind of complexity
- Second: it gives meaning to the data it wraps

Example

Nutrition Facts

1 servings per container

Serving size 1 wra

Amount per serving

Calories

% Da

Total Fat 8g

Saturated Fat 2g

Trans Fat 0g

Cholesterol 20mg

Sodium 870mg

Total Carbohydrate 30g

Dietary Fiber 0g

Total Sugars 1g

Includes 0g Added Sugars

Protein 14g

Vitamin D 0mcg

Calcium 62mg

Iron 1mg

Potassium 150mg 4%

*The % Daily Value tells you how much a nutrient in a serving of food contributes to a daily diet. 2,000 calories a day is used for general nutrition advice.

Nutrition Facts

1 servings per container

Serving size

Amount per serving

Calories

Total Fat 8g

Saturated Fat 2g

Trans Fat 0g

Cholesterol 20mg

Sodium 870mg

Total Carbohydrate

Dietary Fiber 0g

Total Sugars 1g

Includes 0g Adde

Protein 14g

Vitamin D 0mcg

Calcium 62mg

Iron 1mg

Potassium 150mg

*The % Daily Value tells you
serving of food contributes to
day is used for general nutriti

Nutrition Facts

1 servings per container

Serving size 1 wrap(147g)

Amount per serving

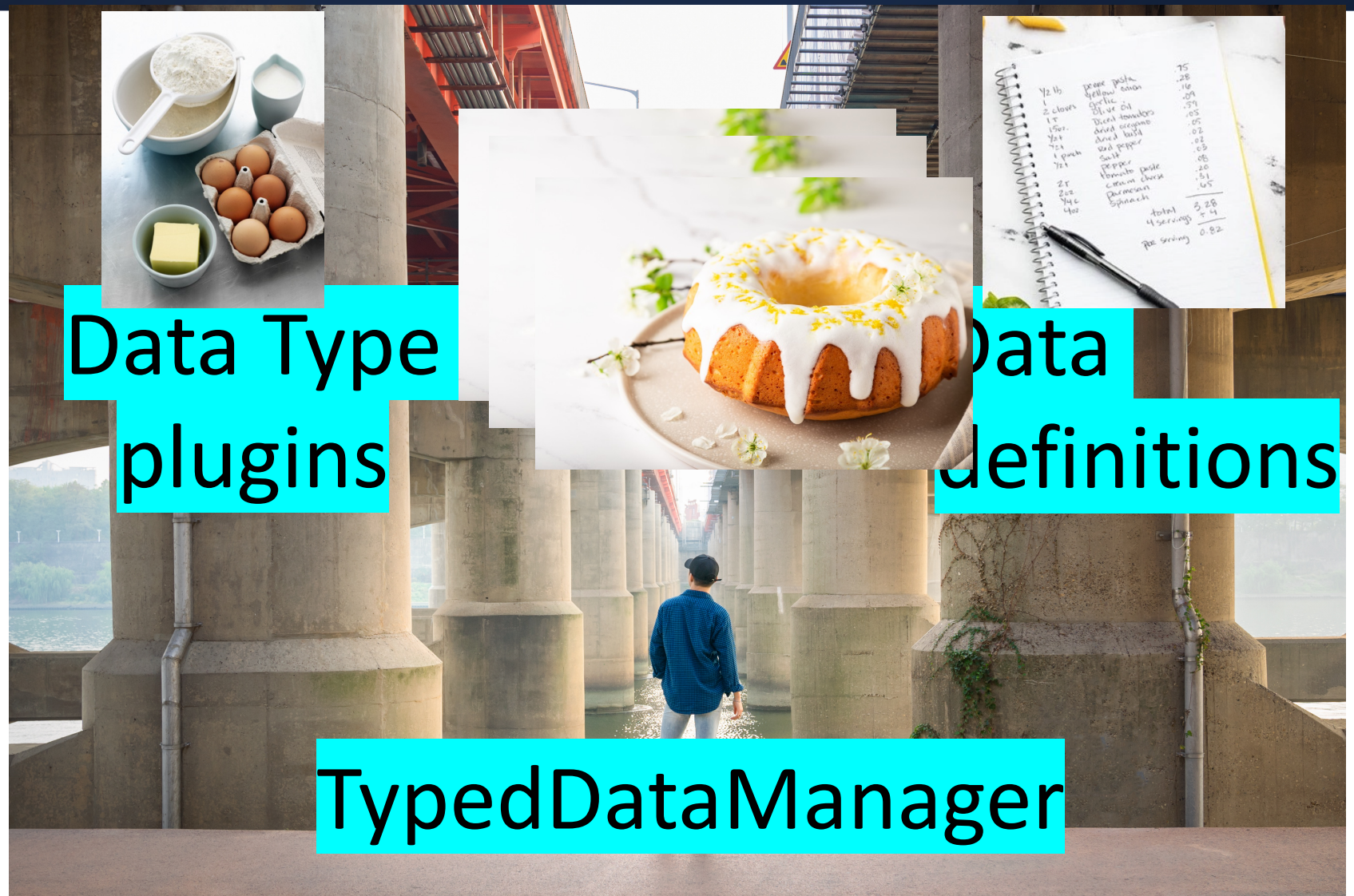
Calories 240

% Daily Value*

Total Fat 8g	10%
Saturated Fat 2g	10%
Trans Fat 0g	
Cholesterol 20mg	7%
Sodium 870mg	38%
Total Carbohydrate 30g	11%
Dietary Fiber 0g	0%
Total Sugars 1g	
Includes 0g Added Sugars	0%
Protein 14g	
Vitamin D 0mcg	0%
Calcium 62mg	4%
Iron 1mg	6%
Potassium 150mg	4%

*The % Daily Value tells you how much a nutrient in a serving of food contributes to a daily diet. 2,000 calories a day is used for general nutrition advice.

TypedData two main pillars



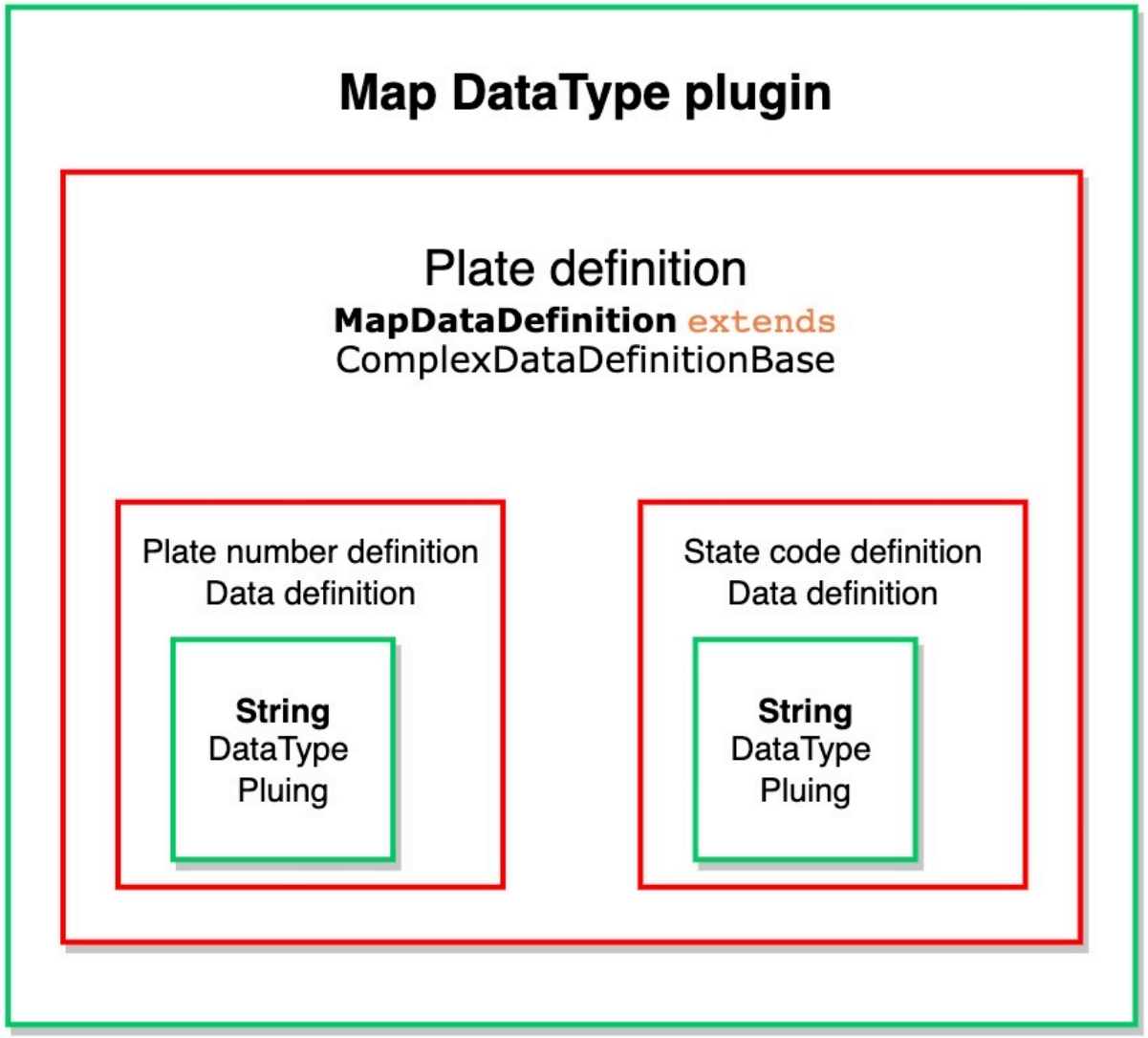
Code example


```php
$definition = \Drupal\Core\TypedData\DataDefinition::create(type: 'string');

$definition->setLabel(label: 'Defines a simple string');

/** @var \Drupal\Core\TypedData\TypedDataInterface $data */
$data = \Drupal::typedDataManager()->create($definition, value: 'my_value');

$value = $data->getValue(); // 'my_value'
$data->setValue('another string');
$type = $data->getDataDefinition()->getDataType();
$label = $data->getDataDefinition()->getLabel();
```

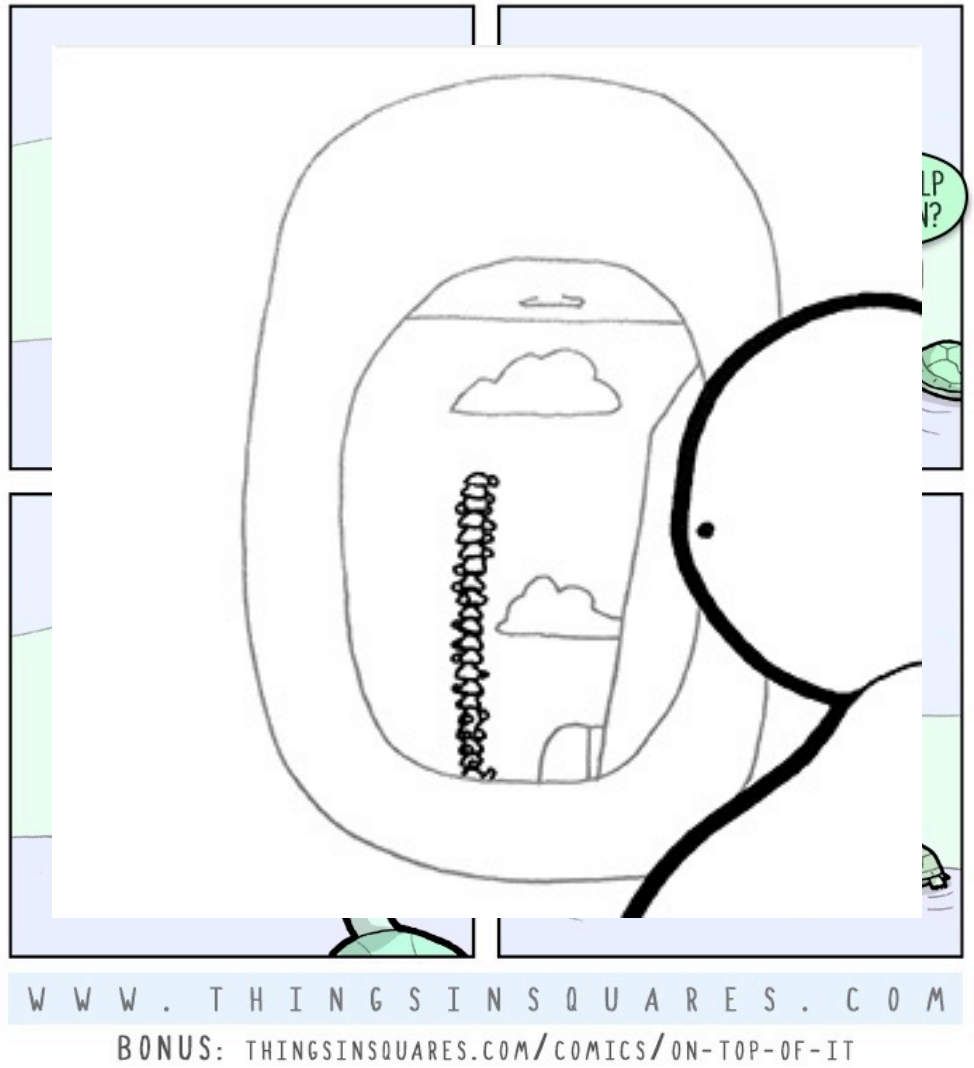
Code example


```php
\Drupal\Core\TypedData\MapDataDefinition::
$data = [
  'state' => 'NY',
  'number' => '405-307',
];
/** @var \Drupal\Core\TypedData\Plugin\DataType\Map $plate */
$plate = \Drupal::typedDataManager()->create($plate_definition, $data);

$label  = $plate->getDataDefinition()->getLabel();
$number = $plate->get('number')->getValue();
$state  = $plate->get('state')->getValue();
```

Code example as diagram



Entity API on top of TypedData API



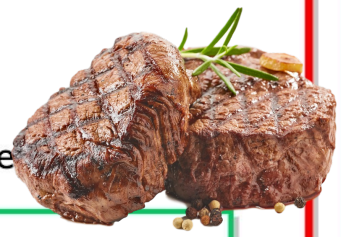
Content entities

FLORIDA DRUPAL CAMP

EntityAdapter extends **TypedData**
DataType plugin - defines "Entity" data type

EntityDataDefinition
extends
ComplexDataDefinitionBase

ase fields



configurable

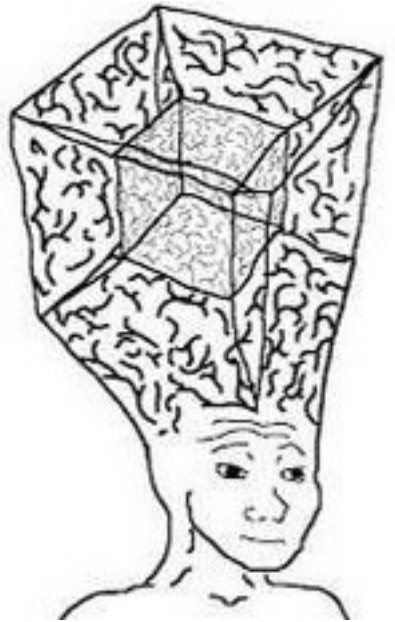
FieldItemList

BaseFieldDefinition
or **BaseFieldOverride**

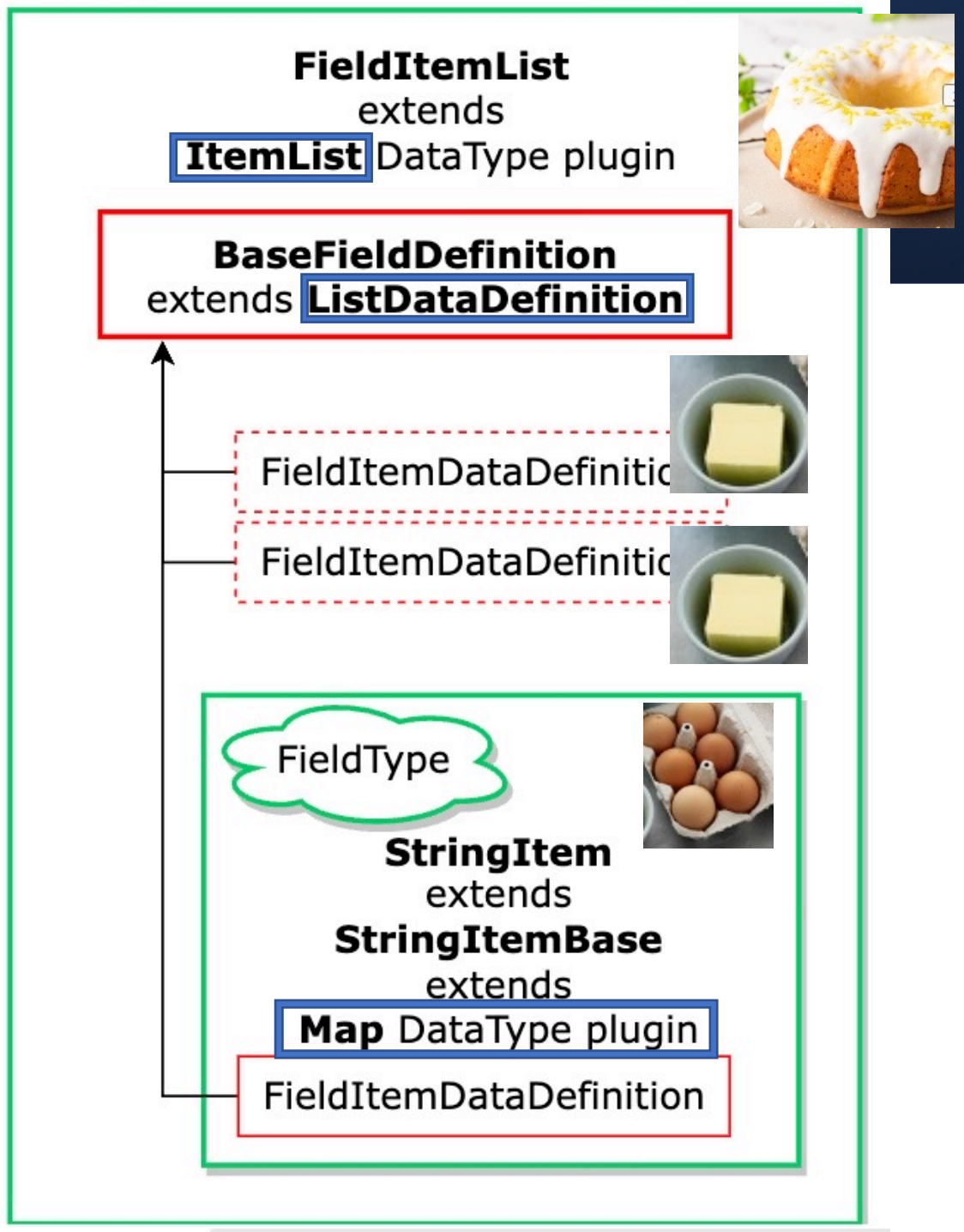
FieldItemList

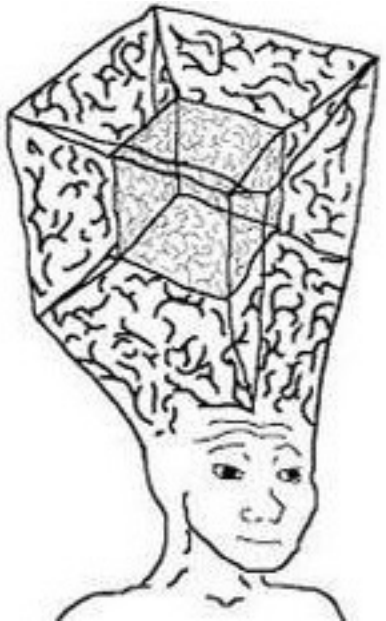
FieldConfig
Configuration Entity

implements
DataDefinitionInterface

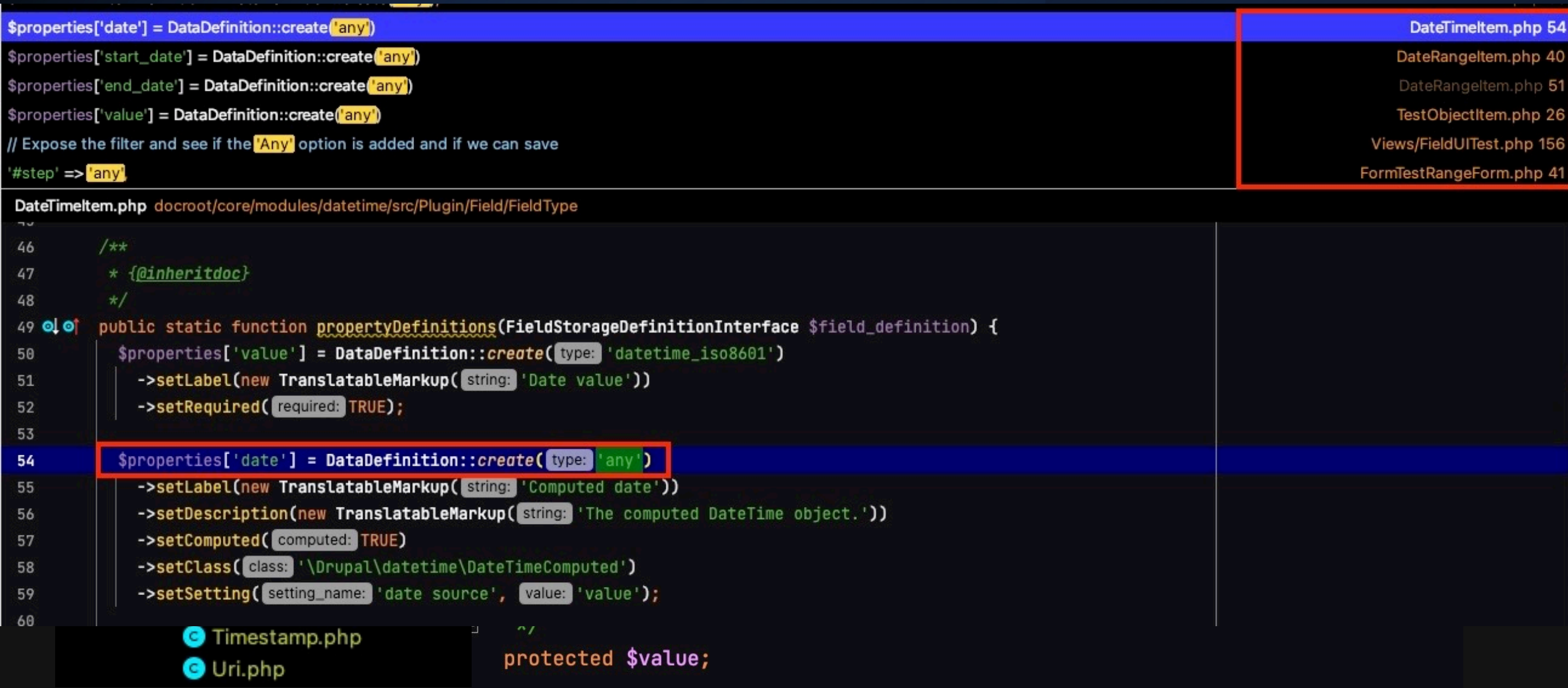


Base field


FieldItemList
extends
ItemList DataType plugin
BaseFieldDefinition
extends ListDataDefinition
FieldItemDataDefinitic
FieldItemDataDefinitic
FieldType
StringItem
extends
StringItemBase
extends
Map DataType plugin
FieldItemDataDefinition



Core Data Type plugins



TypedData by example

FLORIDA DRUPAL CAMP

mglaman/**drupal-typed-data-by-example**

Drupal's Typed Data API by example

2 Contributors | 7 Issues | 2 Discussions | 34 Stars | 6 Forks

Structure

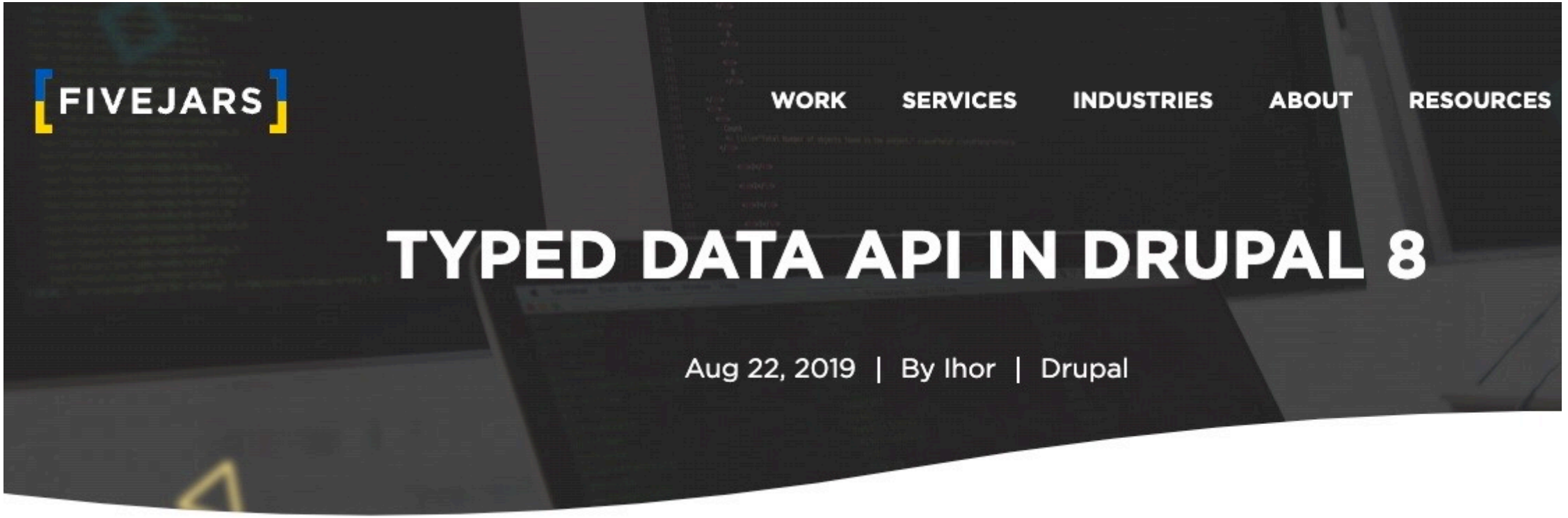
The examples are broken into different groups, each with their

- `primitives` : each example covers the data types provide simplistic examples of using a value and wrapping it in Typ representation.
- `lists` : each example covers using lists (arrays of a specif
- `maps` : each example covers using the Map data type, whic describe an object's shape.
- `serialization` : each example shows how the Typed Data which leverages the Serializer component from Symfony.

```php
$price_definition = MapDataDefinition::create()
  // There is an interesting Drupal problem, here. Drupal has a Decimal field
  // that uses a `string` data property for the value. It's schema for the
  // database is "numeric". In MySQL this maps to DECIMAL but PGSQL and SQLite
  // it is the NUMERIC. This is treated differently than floats.
  //
  // Decimal fields handle precision, but Float fields do not.
  // Remember: The field system deals with databases and storage, typed data
  // does not.
  // @todo It'd be neat to dig up the history of the DecimalItem field type
  //   why wasn't a decimal data type created as well?
  ->setPropertyDefinition(
    'number',
    DataDefinition::create('string')
      ->setRequired(TRUE)
  )
  ->setPropertyDefinition(
    'currency_code',
    DataDefinition::create('string')
      ->setRequired(TRUE)
      ->addConstraint('AllowedValues', ['USD', 'CAD'])
  );

// Now, we'll use the typed data manager to create a typed data value.
$value = $typed_data_manager->create($price_definition, [
  'number' => '10.99',
  'currency_code' => 'USD',
]);
assert($value instanceof Map);
```

TypedData by Fivejars

https://fivejars.com/blog/typed-data-api-drupal-8



TypedData by Fivejars

FLORIDA
DRUPAL CAMP

```php
/**
 * Demo Api Response Definition.
 */
class ResponseDefinition extends ComplexDataDefinitionBase {

  /**
   * {@inheritdoc}
   */
  public function getPropertyDefinitions() {
    if (!isset($this->propertyDefinitions)) {
      $info = &$this->propertyDefinitions;

      $info['status'] = DataDefinition::create('string')
        ->setRequired(TRUE)
        ->setLabel('status')
        ->addConstraint('AllowedValues', ['OK']);
      $info['command'] = DataDefinition::create('string')
        ->setRequired(TRUE)
        ->setLabel('command');
      $info['results'] = ListDataDefinition::create('demo_results')
        ->setLabel('results')
        ->addConstraint('NotNull');
    }
    return $this->propertyDefinitions;
```

Thank you!

FLORIDA DRUPAL CAMP