HEPTAconnect

HEPTACOM GmbH

HEPTACOM solution for medium and large enterprises

Table of contents

1. Portal Developer

1.1 How to be a HEPTAconnect portal developer

This is all about the guidelines to structure a portal or portal extensions.

Be sure to know the general thoughts and requirements to be a HEPTAconnect developer and have a basic understanding what a dataset is and what it means to develop one.

1.1.1 Composer

It is mandatory to add the keyword `heptaconnect-portal` to the composer package that provides one or more portals. This way HEPTAconnect can find your portal and register portal nodes for it. Also more people can easily find your portal on packagist. A common `composer.json` for a portal providing package may look like this:

```
{
    "name": "acme/heptaconnect-portal-bottle",
    "description": "HEPTAconnect portal to provide bottles",
    "type": "library",
    "keywords": [
        "heptaconnect-portal"
    ],
    "require": {
        "php": ">=7.4",
        "acme/heptaconnect-dataset-bottle": ">=1",
        "heptacom/heptaconnect-portal-base": ">=1"
    },
    "autoload": {
        "psr-4": {
            "Acme\\Portal\\Bottle\\": "src/"
        }
    },
    "extra": {
        "heptaconnect": {
            "portals": [
                "Acme\\Portal\\Bottle\\BottlePortal"
            ]
        }
    }
}
```

1.1.2 Structure

The entry point of your portal is an implementation of the `PortalContract`. It can implement a method to provide a configuration template or some custom methods to use in your flow components. None of these methods are mandatory, therefore the minimum valid portal class would look like this:

```
namespace Acme\Portal\Bottle;

use Heptacom\HeptaConnect\Portal\Base\Portal\Contract\PortalContract;

class BottlePortal extends PortalContract
{
}
```

A receiver that gets data from HEPTAconnect is to be told to communicate towards the API it wraps. A common implementation is to use a custom API client and let the receiver do the translation work from dataset structures to API structures:

```
namespace Acme\Portal\Bottle\Receiver;

use Acme\Portal\Bottle\Http\ApiClient;
use Heptacom\HeptaConnect\Dataset\Base\Contract\DatasetEntityContract;
use Heptacom\HeptaConnect\Playground\Dataset\Bottle;
use Heptacom\HeptaConnect\Portal\Base\Reception\Contract\ReceiveContextInterface;
use Heptacom\HeptaConnect\Portal\Base\Reception\Contract\ReceiverContract;
use Ramsey\Uuid\Uuid;

class BottleReceiver extends ReceiverContract
{
    /**
     * @param Bottle $entity
     */
    protected function run(DatasetEntityContract $entity, ReceiveContextInterface $context): void
```

```
    {
        // get API client to send the data to
        $apiClient = new ApiClient();

        // either the entity already has an ID or we create a new one
        $id = $entity->getPrimaryKey() ?? Uuid::uuid4()->toString();

        // translate entity to arbitrary structure and send it to the API
        $apiClient->upsert([
            'id' => $id,
            'cap' => $entity->getCap()->getType(),
            'volume' => $entity->getCapacity()->getAmount(),
        ]);

        // save the primary key, so a mapping is created
        $entity->setPrimaryKey($id);
    }

    public function supports(): string
    {
        // tells HEPTAconnect to use this receiver for bottles
        return Bottle::class;
    }
}
```

As we just read how a receiver is reduced to the case of communication we can compare it to an emitter that loads data from an API and feeds it into HEPTAconnect.

```
namespace Acme\Portal\Bottle\Emitter;

use Acme\Portal\Bottle\Http\ApiClient;
use Heptacom\HeptaConnect\Dataset\Base\Contract\DatasetEntityContract;
use Heptacom\HeptaConnect\Playground\Dataset\Bottle;
use Heptacom\HeptaConnect\Playground\Dataset\Cap;
use Heptacom\HeptaConnect\Playground\Dataset\Volume;
use Heptacom\HeptaConnect\Portal\Base\Emission\Contract\EmitContextInterface;
use Heptacom\HeptaConnect\Portal\Base\Emission\Contract\EmitterContract;

class BottleEmitter extends EmitterContract
{
    protected function run(string $externalId, EmitContextInterface $context): ?DatasetEntityContract
    {
        // get API client to read the data from
        $apiClient = new ApiClient();

        // read data from API client
        $data = $apiClient->select($externalId);

        if (\count($data) === 0) {
            return null;
        }

        // translate arbitrary data structure to entity
        return (new Bottle())
            ->setCap((new Cap())->setType($data['cap']))
            ->setCapacity((new Volume())
                ->setAmount($data['volume'])
                ->setUnit(Volume::UNIT_LITER)
            )
        ;
    }

    public function supports(): string
    {
        // tells HEPTAconnect to use this emitter for bottles
        return Bottle::class;
    }
}
```

1.1.3 Expose status for administration

As the portal node is about to get setup or is in usage an administrator needs to find out about its status regarding a correct configuration or the connectivity state of the underlying data source. A status reporter is meant to get information about a certain topic. Every portal should expose a health status reporter when a data source is used that depends on I/O operations like file or network access.

```
namespace Acme\Portal\Bottle\StatusReporter;

use Acme\Portal\Bottle\Http\ApiClient;
use Heptacom\HeptaConnect\Portal\Base\StatusReporting\Contract\StatusReporterContract;
use Heptacom\HeptaConnect\Portal\Base\StatusReporting\Contract\StatusReportingContextInterface;

class BottleHealthStatusReporter extends StatusReporterContract
{
    public function supportsTopic(): string
    {
```

```
        return self::TOPIC_HEALTH;
    }

    public function run(StatusReportingContextInterface $context): array
    {
        // get API client
        $apiClient = new ApiClient();

        return [
            $this->supportsTopic() => true,
            'bottleCount' => $apiClient->count(),
        ];
    }
}
```

1.1.4 Extend portals via attachments

A dataset sometimes is not able to hold data that is needed for an integration to work. The dataset author might have not thought of this case or evaluated it as an edge case. In these situations you are about to create an emitter decorator via a portal extension. A portal extension is published similar to a portal via the extra section in a composer package.

```
{
    "name": "acme/heptaconnect-portal-bottles-with-content",
    "description": "HEPTAconnect portal extension to provide content for bottles",
    "type": "library",
    "keywords": [
        "heptaconnect-portal-extension"
    ],
    "require": {
        "php": ">=7.4",
        "acme/heptaconnect-portal-bottle": ">=1",
        "acme/heptaconnect-dataset-bottle": ">=1",
        "heptacom/heptaconnect-portal-base": ">=1"
    },
    "autoload": {
        "psr-4": {
            "Acme\\PortalExtension\\Bottle\\": "src/"
        }
    },
    "extra": {
        "heptaconnect": {
            "portalExtensions": [
                "Acme\\PortalExtension\\Bottle\\BottlesWithContentPortal"
            ]
        }
    }
}
```

The portal extension has to specify which portal it extends:

```
namespace Acme\PortalExtension\Bottle;

use Acme\Portal\Bottle\BottlePortal;
use Heptacom\HeptaConnect\Portal\Base\Portal\Contract\PortalExtensionContract;

class BottlesWithContentPortal extends PortalExtensionContract
{
    public function supports(): string
    {
        return BottlePortal::class;
    }
}
```

The emitter decorator will be injected into the call chain and can now alter the mappings to be read from the original and add new data.

```
namespace Acme\PortalExtension\Bottle\Emitter;

use Acme\Portal\Bottle\Http\ApiClient;
use Heptacom\HeptaConnect\Dataset\Base\Contract\DatasetEntityContract;
use Heptacom\HeptaConnect\Playground\Dataset\Bottle;
use Heptacom\HeptaConnect\Playground\Dataset\Volume;
use Heptacom\HeptaConnect\Playground\PortalExtension\Dataset\BottleContent;
use Heptacom\HeptaConnect\Portal\Base\Emission\Contract\EmitContextInterface;
use Heptacom\HeptaConnect\Portal\Base\Emission\Contract\EmitterContract;

class BottleWithContentEmitter extends EmitterContract
{
    protected function extend(
        DatasetEntityContract $entity,
        EmitContextInterface $context
    ) : DatasetEntityContract {
        // get API client
        $apiClient = new ApiClient();
```

```
        // read extra data from the API client
        $data = $apiClient->selectContentData($entity->getPrimaryKey());

        if (\count($data) > 0) {
            // assign extra data to the already emitted entity
            $content = (new BottleContent())
                ->setContent(
                    (new Volume)
                        ->setAmount($data['content'])
                        ->setUnit(Volume::UNIT_LITER)
                );
            $entity->attach($content);
        }

        return $entity;
    }

    public function supports(): string
    {
        // tells HEPTAconnect to use this emitter for bottles
        return Bottle::class;
    }
}
```

1.2 Datasets

This is all about the guidelines to structure a dataset. Be sure to know then general thoughts and requirements to be a HEPTAconnect developer.

1.2.1 Composer

It is recommended to add the keyword `heptaconnect-dataset` to the composer package that provides a dataset. This way more people can easily find your dataset on packagist. A common `composer.json` for a dataset providing package may look like this:

```
{
    "name": "acme/heptaconnect-dataset-bottle",
    "description": "HEPTAconnect dataset package to provide bottles",
    "type": "library",
    "keywords": [
        "heptaconnect-dataset"
    ],
    "require": {
        "php": ">=7.4",
        "heptacom/heptaconnect-dataset-base": ">=1"
    },
    "autoload": {
        "psr-4": {
            "Acme\\Dataset\\Bottle\\": "src/"
        }
    }
}
```

1.2.2 Structure

Datasets describe a collection of structures that express common properties of familiar complex structures. These are the building blocks for portals. A dataset should be as common as possible. You don't have to include all possibilities at once.

In case of describing data about bottles a single bottle can be described as the following:

```
namespace Acme\Dataset\Bottle;

use Heptacom\HeptaConnect\Dataset\Base\Contract\DatasetEntityContract;

class Bottle extends DatasetEntityContract
{
    protected Volume $capacity;

    protected LabelCollection $labels;

    protected Cap $cap;

    protected BottleShape $shape;

    /* getters and setters */
}
```

It is important to use the base class `DatasetEntity` and use `protected` fields for internal processing in HEPTAconnect to work.

There are supporting classes to build up structures to use throughout any dataset. As internationalization (i18n) faces everyone during a data transport we offer helpful types to make translatable fields easier to handle.

```
namespace Acme\Dataset\Bottle;

use Heptacom\HeptaConnect\Dataset\Base\Contract\DatasetEntityContract;
use Heptacom\HeptaConnect\Dataset\Base\Translatable\TranslatableString;

class Label extends DatasetEntityContract
{
    protected TranslatableString $text;

    protected string $color;
}
```

As php does not offer generics every collection is missing the information about the types managed within the list of data. To ensure correct data in arrays and add type hinting for IDEs we provide tooling around typed collections. They contain psalm

hints about types and just have to know the contents type to work. Types like `StringCollection`, `IntegerCollection` and `DateTimeCollection` are already shipped in the dataset base to help building up a custom dataset very quickly.

```
namespace Acme\Dataset\Bottle;

use Heptacom\HeptaConnect\Dataset\Base\DatasetEntityCollection;

class LabelCollection extends DatasetEntityCollection
{
    protected function getT(): string
    {
        return Label::class;
    }
}
```

The usage of typed enumerations is discouraged as these are very difficult to impossible to extend. We prefer to use constant strings. In best case it is just a UUID as value. This way the string value can receive new values if anyone needs to extend your dataset later on.

1.2.3 Extend datasets with attachments

A dataset is sometimes not able to hold data that is needed for an integration to work. The dataset author might have not thought of this case or evaluated it as an edge case. In these situations you are able to extend dataset entities. To provide additional data for the bottle entity you have to create a custom structure that holds the additional data you need. A data extension can be any class that implements the `AttachableInterface`. They can be attached to an existing entity using the `attach` method. The `DatasetEntityContract` already implements this interface, so existing entities can be plugged into another entity with just a few actions.

```
namespace Acme\Dataset\Bottle;

use Heptacom\HeptaConnect\Dataset\Base\Contract\AttachableInterface;

class BottleContent implements AttachableInterface
{
    protected Volume $capacity;
}

$bottle = new Bottle();
$bottle->attach(new BottleContent());
```

1.3 Portal

HEPTAconnect is focused on modularity. Different packages can be bundled together to adapt it to your needs. That is why a single adapter to an external system is organized in a dedicated package. These packages are called portals. A portal consists of three main components to comply with the HEPTAconnect ecosystem: Explorers, emitters and receivers.

1.3.1 Intention

Those three component types are explained in more detail on their respective documentation pages. Here is a brief explanation of the flow of data through HEPTAconnect:

All of these components connect to their respective portal's data source. The explorer publishes every object to HEPTAconnect (creating a mapping for each of them). In the next step HEPTAconnect will pass these mappings to an emitter for it to read the entire object and construct a data set entity. This object is then passed to the receiver of another portal where it is then written to the data source.

1.3.2 Usage

A portals job is to register its components and to provide services that are unique for it. Those services e.g. can be a custom API client or a service that can access data inside a static file. You can create a portal by implementing PortalContract and referencing your portal class in the extra section of your packages composer.json like this:

```
"extra": {
    "heptaconnect": {
        "portals": [
            "Foo\\Bar\\Portal"
        ]
    }
},
```

It is required for your package's composer.json to include the keyword `heptaconnect-portal`. This is the way to let HEPTAconnect know it should take a look at the extra section of your package.

HEPTAconnect is split into different packages to provide great modularity. As a result your portal package only needs a single composer dependency to be functional: `heptacom/heptaconnect-portal-base`. This package provides you with all the contracts, structs and services that are relevant for portals while maintaining full system agnosticism.

1.4 Explorer

A portal connects to a data source for read and write operations. To let HEPTAconnect know about objects in the data source, an explorer has to publish these objects' primary keys. Publishing a primary key means to check whether a mapping for it already exists and to create one if it doesn't.

1.4.1 Intention

An explorer is a flow component that is primarily used after a new portal node has been created. At this moment there are no mappings in HEPTAconnect (for that portal node) but objects are already present in the data source. To get all these objects into the system, an explorer iterates over all of their identifiers and publishes them.

1.4.2 Usage

Explorers must implement the `ExplorerContract`. Every explorer must define which data type it supports. In the following example we see an explorer that supports the data type `Bottle`.

```
public function supports(): string
{
    return Bottle::class;
}
```

The `run` method iterates over primary keys in your data source and yield them.

```
protected function run(ExploreContextInterface $context): iterable
{
    $credentials = $context->getConfig()['credentials'];
    $client = new ApiClient($credentials);

    yield from $client->getBottleIds();
}
```

The explorer will iterate over the result of `$client->getBottleIds()` and yield the ids.

1.5 Emitter

An emitter is a flow component that has the job to read data from its portal's data source, convert it into a data set entity and hand that entity over to HEPTAconnect.

1.5.1 Intention

When an object from a data source is published to HEPTAconnect, a mapping will be created (if it doesn't exist yet). A publication also sends a message to the job queue telling HEPTAconnect to emit the object. This approach has the benefit (as opposed to direct transfer) that publications can be done quickly and don't take up a lot of computing time. This enables publications during time critical processes like e.g. a web request.

The actual reading of data is handled by a consumer process of the job queue, while the publication can have various origins.

1.5.2 Usage

Emitters must implement the `EmitterContract`. Every emitter must define which data type it supports. In the following example we see an emitter that supports the data type `Bottle`.

```
public function supports(): string
{
    return Bottle::class;
}
```

The `run` method receives a single id and should return a completely filled entity.

```
protected function run(string $externalId, EmitContextInterface $context): ?DatasetEntityContract
{
    $credentials = $context->getConfig()['credentials'];
    $client = new ApiClient($credentials);

    $result = new Bottle();
    $result->setPrimaryKey($externalId);
    $result->setCapacity($client->readBottleCapacity($externalId));
    $result->setShape($client->readBottleShape($externalId));
    $result->getLabels()->push($client->readBottleLabels($externalId));

    return $result;
}
```

To run the process in a batch pattern you can also implement `batch` instead.

```
protected function batch(iterable $externalIds, EmitContextInterface $context): iterable
{
    $credentials = $context->getConfig()['credentials'];
    $client = new ApiClient($credentials);

    $capacities = $client->readBottlesCapacity($externalIds);
    $shapes = $client->readBottlesShape($externalIds);
    $labels = $client->readBottlesLabels($externalIds);

    foreach ($externalIds as $externalId) {
        $result = new Bottle();
        $result->setPrimaryKey($externalId);
        $result->setCapacity($capacities[$externalId]);
        $result->setShape($shapes[$externalId]);
        $result->getLabels()->push($labels[$externalId]);

        yield $result;
    }
}
```

1.6 Receiver

A receiver is a flow component that has the job to take incoming entities from HEPTAconnect, convert them into API specific structures and write the API payload into its portal's data source.

1.6.1 Usage

Receivers must implement the `ReceiverContract`. Every receiver must define which data type it supports. In the following example we see a receiver that supports the data type `Bottle`.

```
public function supports(): string
{
    return Bottle::class;
}
```

The `run` method receives a completely filled entity and must set a primary key to the entity after successful writing it to the portal's data source.

```
protected function run(DatasetEntityContract $entity, ReceiveContextInterface $context): void
{
    $credentials = $context->getConfig()['credentials'];
    $client = new ApiClient($credentials);

    if ($entity->getPrimaryKey()) {
        $client->updateBottle($entity->getPrimaryKey(), [
            'capacity' => $entity->getCapactity()->getAmount(),
            'shape' => $entity->getShape()->getType(),
            'labels' => $entity->getLabels()->column('getText'),
        ]);
    } else {
        $id = $client->createBottle([
            'capacity' => $entity->getCapactity()->getAmount(),
            'shape' => $entity->getShape()->getType(),
            'labels' => $entity->getLabels()->column('getText'),
        ]);
        $entity->setPrimaryKey($id);
    }
}
```

To run the process in a batch pattern you can also implement `batch` instead.

```
protected function batch(TypedDatasetEntityCollection $entities, ReceiveContextInterface $context): void
{
    $commands = $entities->map(static fn (Bottle $b): array => [
        'id' => $b->getPrimaryKey(),
        'capacity' => $entity->getCapactity()->getAmount(),
        'shape' => $entity->getShape()->getType(),
        'labels' => $entity->getLabels()->column('getText'),
    ]);

    $credentials = $context->getConfig()['credentials'];
    $client = new ApiClient($credentials);
    $results = $client->upsertBottles($commands);

    foreach ($entities as $step => $bottle) {
        $bottle->setPrimaryKey($results[$step]);
    }
}
```

1.7 HTTP Client Middleware

Since heptacom/heptaconnect-portal-base: 0.9.2

A portal, that connects to an HTTP API regularly needs to add headers, session handling or customized response handling, for every request/response. To provide features like this for multiple HTTP requests, you can use these middlewares.

1.7.1 Intention

Shared code will occur with more and more different API calls an HTTP Client is used for. Authenticated session handling is a common case, that can be solved by wrapping each sending of a request and ensure to use an authenticated session identifier. This is where the HTTP Client Middlewares come into play.

Intercepting outbound HTTP requests is already simplified, when using the shipped HTTP Client, or defining a decorator for the `PSR-18` HTTP Client. With this HTTP Client Middleware it is simpler to build decorations around the HTTP Client as only a single file is needed with less potential to do it wrong. The underlying interface is similar to the `PSR-15` middleware interface but for outbound HTTP requests.

1.7.2 Usage

Services of type `\Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\HttpClientMiddlewareInterface` will automatically get the service tag `heptaconnect.http.client.middleware`. All services with the tag `heptaconnect.http.client.middleware` will be executed for each request, that is sent by the `Psr\Http\Client\ClientInterface` service. Adding a single file to your code will be sufficient for reoccurring tasks of your HTTP client. See this pattern for dumping message or like the following example for profiling:

```
use Heptacom\HeptaConnect\Portal\Base\Profiling\ProfilerContract;
use Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\HttpClientMiddlewareInterface;
use Psr\Http\Client\ClientInterface;
use Psr\Http\Message\RequestInterface;
use Psr\Http\Message\ResponseInterface;

final class ProfilerMiddleware implements HttpClientMiddlewareInterface
{
    private ProfilerContract $profiler;

    private bool $profilingEnabled;

    public function __construct(ProfilerContract $profiler, bool $configProfilingEnabled)
    {
        $this->profiler = $profiler;
        $this->profilingEnabled = $configProfilingEnabled;
    }

    public function process(RequestInterface $request, ClientInterface $handler): ResponseInterface
    {
        if (!$this->profilingEnabled) {
            return $handler->sendRequest($request);
        }

        $this->profiler->start(\sprintf('http client %s %s', $request->getMethod(), $request->getUri()));

        try {
            $response = $handler->sendRequest($request);
        } catch (\Throwable $exception) {
            $this->profiler->stop($exception);

            throw $exception;
        }

        $this->profiler->stop();

        return $response;
    }
}
```

1.8 HTTP Handler

A portal can respond to an HTTP request using HTTP handlers. This can be useful for implementing webhooks or web browser interaction (e.g. interactive OAuth 2.0 setup).

1.8.1 Intention

Many external systems support notifications for changes via webhooks. Some systems only send identifiers of affected entities while others send the entire entity or only the change set. Portals can process these notifications by implementing an HTTP handler. For maximum flexibility, handlers are provided with the entire HTTP request object and a prepared HTTP response object that should be modified. The HTTP message objects implement `\Psr\Http\Message\ServerRequestInterface` and `\Psr\Http\Message\ResponseInterface` defined in `PSR-7`.

It is recommended to keep the operations in HTTP handlers as lightweight as possible, because these components will run in a web context where arbitrary memory limits and time limits can apply. For webhook notifications containing only identities, the handler should use the `\Heptacom\HeptaConnect\Portal\Base\Publication\Contract\PublisherInterface` to offload further I/O into an emitter. Webhooks that receive entire entity payloads can use a packer and dispatch entities immediately to the `\Heptacom\HeptaConnect\Portal\Base\Flow\DirectEmission\DirectEmissionFlowContract`.

1.8.2 Usage

object-oriented notation **short notation**

```
use Heptacom\HeptaConnect\Playground\Dataset\Bottle;
use Heptacom\HeptaConnect\Portal\Base\Mapping\MappingComponentCollection;
use Heptacom\HeptaConnect\Portal\Base\Mapping\MappingComponentStruct;
use Heptacom\HeptaConnect\Portal\Base\Publication\Contract\PublisherInterface;
use Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\HttpHandleContextInterface;
use Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\HttpHandlerContract;
use Psr\Http\Message\ResponseInterface;
use Psr\Http\Message\ServerRequestInterface;

class BottleHttpHandler extends HttpHandlerContract
{
    private PublisherInterface $publisher;

    public function __construct(PublisherInterface $publisher)
    {
        $this->publisher = $publisher;
    }

    protected function supports(): string
    {
        return 'bottle';
    }

    protected function post(
        ServerRequestInterface $request,
        ResponseInterface $response,
        HttpHandleContextInterface $context
    ): ResponseInterface {
        $bottleIds = \json_decode($request->getQueryParams()['bottle-ids']);

        $bottleMappings = \array_map(static fn (string $bottleId) => new MappingComponentStruct(
            $context->getPortalNodeKey(),
            Bottle::class,
            $bottleId
        ), $bottleIds);

        $this->publisher->publishBatch(new MappingComponentCollection($bottleMappings));

        return $response->withStatus(201);
    }
}
```

```
use Heptacom\HeptaConnect\Playground\Dataset\Bottle;
use Heptacom\HeptaConnect\Portal\Base\Builder\FlowComponent;
use Heptacom\HeptaConnect\Portal\Base\Mapping\MappingComponentCollection;
use Heptacom\HeptaConnect\Portal\Base\Mapping\MappingComponentStruct;
use Heptacom\HeptaConnect\Portal\Base\StorageKey\Contract\PortalNodeKeyInterface;
use Psr\Http\Message\ResponseInterface;
use Psr\Http\Message\ServerRequestInterface;

FlowComponent::httpHandler('bottle')->post(static function (
    ServerRequestInterface $request,
    ResponseInterface $response,
    PortalNodeKeyInterface $portalNodeKey
): ResponseInterface {
    $bottleIds = \json_decode($request->getQueryParams()['bottle-ids']);

    $bottleMappings = \array_map(static fn (string $bottleId) => new MappingComponentStruct(
        $portalNodeKey,
        Bottle::class,
        $bottleId
    ), $bottleIds);

    $this->publisher->publishBatch(new MappingComponentCollection($bottleMappings));

    return $response->withStatus(201);
});
```

1.8.3 Sharing URLs

There are several ways how to access the HTTP handlers endpoint. One set of tools are available on the commandline and are explained in the administrator section for HTTP APIs. Within the utility services there is `\Heptacom\HeptaConnect\Portal\Base\Web\Http\HttpHandlerUrlProviderInterface` that can resolve an HTTP handler name into an absolute URL. There is also an HTTP client that implements `\Psr\Http\Client\ClientInterface` defined in `PSR-18` which can be used to post the resolved URL to an external API to register the HTTP handler e.g. as webhook for events.

1.8.4 Sharing code between handlers

With more and more implementations of HTTP handlers, shared code will occur. To use same code for multiple HTTP handlers you can use HTTP Handler Middlewares.

1.8.5 Dump requests and responses

The bridges trigger dumping requests and responses and provide a storage for the dumps. See the administrator section on HTTP API debugging how to use that to debug the HTTP traffic.

1.8.6 Patterns

- Serve a file from filesystem using HTTP handler

1.9 HTTP Handler Middleware

Since heptacom/heptaconnect-portal-base: 0.9.2

A portal can respond to an HTTP request using HTTP Handlers. To provide features for multiple HTTP handlers, you can use these middlewares.

1.9.1 Intention

With more and more implementations of HTTP handlers, shared code will occur e.g. authentication check, logging and profiling. To share this code you can either copy code snippets or use extracted services over and over again. With HTTP middlewares known from `PSR-15` you can also write code that intercept every inbound HTTP request for your HTTP handlers.

1.9.2 Usage

Services of type `\Psr\Http\Server\MiddlewareInterface` will automatically get the service tag `heptaconnect.http.handler.middleware`. All services with the tag `heptaconnect.http.handler.middleware` will be executed before an HTTP Handler will receive the request. Adding a single file to your code will be sufficient for reoccurring tasks of your HTTP handlers like the following example for profiling:

```
use Heptacom\HeptaConnect\Portal\Base\Profiling\ProfilerContract;
use Psr\Http\Message\ResponseInterface;
use Psr\Http\Message\ServerRequestInterface;
use Psr\Http\Server\MiddlewareInterface;
use Psr\Http\Server\RequestHandlerInterface;

final class ProfilerMiddleware implements MiddlewareInterface
{
    private ProfilerContract $profiler;

    private bool $profilingEnabled;

    public function __construct(ProfilerContract $profiler, bool $configProfilingEnabled)
    {
        $this->profiler = $profiler;
        $this->profilingEnabled = $configProfilingEnabled;
    }

    public function process(ServerRequestInterface $request, RequestHandlerInterface $handler): ResponseInterface
    {
        if (!$this->profilingEnabled) {
            return $handler->handle($request);
        }

        $this->profiler->start(\sprintf('http handler %s %s', $request->getMethod(), $request->getUri()));

        try {
            $response = $handler->handle($request);
        } catch (\Throwable $exception) {
            $this->profiler->stop($exception);

            throw $exception;
        }

        $this->profiler->stop();

        return $response;
    }
}
```

1.10 Status Reporting

Status reporting provides JSON serializable data, so it can be processed easily by many services. It is divided into multiple topics. There are four core topics to define a common set that probably every portal needs:

- **health** Provide hints about I/O connectivity
- **config** Provide hints about configuration choices
- **analysis** Provide hints about usage statistics
- **info** Provide static data that does not fit into a composer package structure

Any portal and portal extension provides status reporters for any topic they want to.

1.10.1 Intention

Status reporters are intended for the usage of the four core reportings. It is fine to add new topics, but some tooling might not support it out of the box.

This way a portal can communicate how to configure a portal node in a multi-step configuration setup or inform about the health status of the connected datasource for monitoring in everyday usage.

1.10.2 Usage

A status reporter must extend the `StatusReporterContract` and should implement the `run` method.

Health

A health status reporter can look like this:

```
namespace FooBar\StatusReporter;

use FooBar\AcmeApi\ApiClient;
use Heptacom\HeptaConnect\Portal\Base\StatusReporting\Contract\StatusReporterContract;
use Heptacom\HeptaConnect\Portal\Base\StatusReporting\Contract\StatusReportingContextInterface;

class HealthStatusReporter extends StatusReporterContract
{
    public function supportsTopic(): string
    {
        return self::TOPIC_HEALTH;
    }

    protected function run(StatusReportingContextInterface $context): array
    {
        $result = [$this->supportsTopic() => true];

        try {
            $apiClient = new ApiClient($context->getConfig()['credentials']);
            $apiClient->commonReadonlyEndpoint();
        } catch (\Throwable $exception) {
            $result[$this->supportsTopic()] = false;
            $result['message'] = $exception->getMessage();
        }

        return $result;
    }
}
```

This status reporter uses the topic key to communicate the evaluation of the portal node topics' well-being. It also uses a common endpoint on the data source to validate the configuration is usable in a production scenario and therefore validates the health state of the underlying API of the datasource.

Configuration

A configuration status reporter can look like this:

```
namespace FooBar\StatusReporter;
```

```
use FooBar\AcmeApi\ApiClient;
use FooBar\AcmeApi\Node;
use Heptacom\HeptaConnect\Portal\Base\StatusReporting\Contract\StatusReporterContract;
use Heptacom\HeptaConnect\Portal\Base\StatusReporting\Contract\StatusReportingContextInterface;

class ConfigurationStatusReporter extends StatusReporterContract
{
    public function supportsTopic(): string
    {
        return self::TOPIC_CONFIG;
    }

    protected function run(StatusReportingContextInterface $context): array
    {
        $result = [$this->supportsTopic() => true];

        try {
            $apiClient = new ApiClient($context->getConfig()['credentials']);
            $nodes = $apiClient->getSubnodes();
            $result['nodes'] = array_map(static fn (Node $node): string => $node->getId(), $nodes);
        } catch (\Throwable $exception) {
            $result[$this->supportsTopic()] = false;
            $result['message'] = $exception->getMessage();
        }

        return $result;
    }
}
```

It requests with the already existing configuration further resources (e.g. nodes) that needs to be referred to in upcoming configurations as well.

Analysis

An analysis status reporter can look like this:

```
namespace FooBar\StatusReporter;

use Heptacom\HeptaConnect\Portal\Base\StatusReporting\Contract\StatusReporterContract;
use Heptacom\HeptaConnect\Portal\Base\StatusReporting\Contract\StatusReportingContextInterface;

class AnalysisStatusReporter extends StatusReporterContract
{
    public function supportsTopic(): string
    {
        return self::TOPIC_ANALYSIS;
    }

    protected function run(StatusReportingContextInterface $context): array
    {
        return [
            $this->supportsTopic() => true,
            'lastClientUsageTimestamp' => $context->getStorage()->get('apiClientLastUsage'),
        ];
    }
}
```

This status reporter relies on a feature the API client needs to implement as well: writing the last unix timestamp into the portal node storage. This way you can track the API clients behaviour.

Information

An information status reporter can look like this:

```
namespace FooBar\StatusReporter;

use Heptacom\HeptaConnect\Portal\Base\StatusReporting\Contract\StatusReporterContract;
use Heptacom\HeptaConnect\Portal\Base\StatusReporting\Contract\StatusReportingContextInterface;

class InformationStatusReporter extends StatusReporterContract
{
    public function supportsTopic(): string
    {
        return self::TOPIC_INFO;
    }

    protected function run(StatusReportingContextInterface $context): array
    {
        return [
            $this->supportsTopic() => true,
            'demo' => [
                'username' => 'root',
                'password' => 'password',
            ],
```

```
        ];
    }
}
```

The status reporter provides static information about possible debug configuration that does not need to be calculated. This can contain various of different types of information that suits the portal needs.

1.11 Morpher

A morpher is basically an implementation of a portal that does not directly interact with an external data source but instead acts as a conversion or aggregation mechanism. Its purpose is to receive data from one or multiple portals and emit them in some other form for another portal.

1.11.1 Intention

Morphers are multi-purpose-utilities when it comes to moving data from one portal to another while modifying it on the fly. A simple example could be the following use-case:

Say you have an ecommerce portal that emits orders and customers. You also have a customer support portal that can receive support tickets. The goal is to convert orders and their respective customers into support tickets. You cannot configure a route from your ecommerce portal directly to your customer support portal, because they do not support the same data types and also you have to combine an order with a customer to have enough information for a support ticket.

Your morpher would have two receivers. One for orders and one for customers. Both of them store their received objects in the Key-Value-Storage. The order receiver will publish a support ticket object whenever it saves an order to the Key-Value-Storage.

The morpher also has an emitter supporting those support ticket objects. It will load an order from the Key-Value-Storage and (using the customer-id from said order object) it will then try to load a customer from the Key-Value-Storage. If a customer is found, all the necessary informations are present and can be converted to a new support ticket object which is then emitted. If the customer cannot be found, the order is re-published, so the emitter can try again later.

1.11.2 Usage

While the example above is the most common way to use a morpher, it is by far not the only way. A morpher is technically indistinguishable from an ordinary portal. It is its unique role as an intermediate portal that makes it a morpher. Because of that, there are numerous possibilities for what a morpher can do and it almost certainly comes down to an individual use-case.

Here are some examples for what a morpher could potentially do.

- Convert one data type to another one.
- Combine multiple objects into a single new one, even when the original objects are emitted at different times.
- Measure the throughput between two portals by logging every transfer.
- Throttling the throughput between two portals to comply with API rate limits.

1.12 Key-Value-Storage

The Key-Value-Storage is a storage component designed for portals to have a simple storage mechanism restricted to a specific portal node. The intention is to provide a storage abstraction for morphers to store intermediate object data. However, a regular portal may utilize the Key-Value-Storage as well.

1.12.1 Intention

The intended way to use the Key-Value-Storage is for a morpher to persist intermediate object data for later useage. The contract exposes a getter and a setter method and the and keys are unique for a portal node. A key can be any string of up to 255 characters.

A morpher's receiver could receive an object and store it in the Key-Value-Storage. The same morpher's emitter could later retrieve the object from the Key-Value-Storage and combine it with some other data. However, since a morpher is just a portal with a certain role to it, every regular portal may utilize the Key-Value-Storage as well.

1.12.2 Usage

Because the Key-Value-Storage is restricted to a portal node, it is contextual to a run. That is why it can be retrieved from the context of an explorer, an emitter or a receiver.

A portal can get or set any data to the Key-Value-Storage. Depending on the capabilities of the storage it is even possible to store files in it by wrapping the file contents in the corresponding struct class.

1.12.3 Error handling

In theory an implementation of the Key-Value-Storage should be able to handle any kind of data. In practise though, whether a certain set of data can be stored depends on the available storage strategies. Providing these strategies is the responsibility of the storage, so a portal cannot influence these. If a portal tries to store a value that cannot be handled by any available strategy, the Key-Value-Storage will throw an exception. Similarly, if a portal tried to read a value that cannot be handled by any strategy (maybe the corresponding denormalizer has been removed with a recent update), the Key-Value-Storage will throw an exception as well.

1.13 Explorer decoration

A portal extension allows further customizations in behaviour. This includes changing the discovery of elements.

1.13.1 Intention

A decorating explorer can list additional entries and skip unwanted entries.

1.13.2 Usage

Decorating explorers must follow the same basics as normal explorers so be sure to read the explorer explanation page first. The main difference is in their registration.

Implementing `run` in an explorer decorator like this will add further elements to the exploration process.

```
protected function run(ExploreContextInterface $context): iterable
{
    $credentials = $context->getConfig()['credentials'];
    $client = new ApiClient($credentials);

    foreach ($client->getOtherBottles() as $bottle)
    {
        $entity = new Bottle();
        $entity->setPrimaryKey((string) $bottle['id']);

        yield $entity;
    }
}
```

The explorer will iterate over the result of `$client->getOtherBottles()` and construct a data set entity for every item. The primary key is set and the entity is then yielded.

For a scenario to skip elements that should not be discovered anymore by the extended portal you have to implement `isAllowed` instead of `run`. Any explored item will be passed to the allowance check through every decorator. In the following example we only allow bottles that contain caffeinated beverages.

```
protected function isAllowed(string $externalId, ?DatasetEntityContract $entity, ExploreContextInterface $context): bool
{
    $credentials = $context->getConfig()['credentials'];
    $client = new ApiClient($credentials);

    return $client->getBottleAdditives($externalId)->contains('caffeine');
}
```

1.14 Emitter decoration

A portal extension allows further customizations in behaviour. This includes changing the emission of data on elements.

1.14.1 Intention

A decorating emitter can change values of existing scalar values and add further attachments to an entity for further complex data structures to be transferred.

1.14.2 Usage

Decorating emitters must follow the same basics as normal emitters so be sure to read the emitter explanation page first.

Implementing `run` in an emitter decorator like a normal emitter will add further elements to the emission process. This is useful when more entities are explored first otherwise we run into confusion for the further processing as for the same primary key there will be two different filled values emitted. To prevent duplicate emission you can add a check whether this is the right entity to process and otherwise return `null`.

```
protected function run(string $externalId, EmitContextInterface $context): ?DatasetEntityContract
{
    // get portal specific API client to communicate the data from the contexts configuration
    $credentials = $context->getConfig()['credentials'];
    $client = new ApiClient($credentials);

    if (!$client->isOtherBottle($externalId)) {
        return null;
    }

    $data = $client->select($externalId);

    if (\count($data) === 0) {
        return null;
    }

    return (new Bottle())
        ->setCap((new Cap())->setName($data['cap']))
        ->setCapacity(new Liter($data['volume']));
}
```

The emitter will check if this is an other bottle that has been explored by the explorer decorator first and load it. Otherwise skip it by returning `null`.

1.15 Receiver decoration

A portal extension allows further customizations in behaviour. This includes adding further writes when receiving data from elements.

1.15.1 Intention

A decorating receiver can:

- change values of existing scalar values and add further attachments to an entity for further complex data structures to be received by the decorated receiver
- replace the complete receive process of the decorated receiver
- use the received and mapped entities of the decorated receiver and do further actions on them

1.15.2 Usage

Decorating receiver must follow the same basics as normal receivers so be sure to read the receiver explanation page first.

Implementing `run` in a receiver decorator like a normal receiver will receive first all elements before the decorated receiver.

For a scenario to read further data to already received elements by the extended receiver you have to implement `receive` and `run`. `receive` has to be adjusted to not execute the `run` method with data before the decorated receiver but after the decorated receiver has run.

```
public function receive(
    MappedDatasetEntityCollection $mappedDatasetEntities,
    ReceiveContextInterface $context,
    ReceiverStackInterface $stack
): iterable {
    return $this->receiveNextForExtends($stack, $mappedDatasetEntities, $context);
}

/**
 * @param Bottle $entity
 */
protected function run(DatasetEntityContract $entity, ReceiveContextInterface $context): void
{
    if ($entity->getPrimaryKey() === null) {
        return;
    }

    $additives = $entity->getAttachment(AdditiveCollection::class);

    if (!$additives instanceof AdditiveCollection) {
        return;
    }

    $credentials = $context->getConfig()['credentials'];
    $client = new ApiClient($credentials);
    // get portal specific API client to communicate the data from the contexts configuration
    $client->upsert(
        $additives->map(static fn (Additive $a): array => [
            'additiveName' => $a->getName(),
            'bottleId' => $entity->getPrimaryKey(),
        ])
    );
}
```

The receiver will check if this entity has been received by the receiver decorator first and save additional data.

1.16 Direct Emission Explorer

A portal connects to a data source for read and write operations. To let HEPTAconnect know about objects in the data source, an explorer has to publish these objects' primary keys. This can be an issue when data sources are read-once or difficult to navigate to certain data points so emitters can't act on it properly. To solve this we allow explorers to emit as well.

1.16.1 Intention

Beside the intentions of a regular explorer this can be used for rather static data or difficult/inefficient to access data sources as this is also allowed to do an emission.

1.16.2 Usage

Explorers must implement the `ExplorerContract`. Every explorer must define which data type it supports. In the following example we see an explorer that supports the data type `Bottle`.

```
public function supports(): string
{
    return Bottle::class;
}
```

The `run` method iterates over objects in your data source and return them as dataset entities. It is crucial to set the primary key of these entities.

```
protected function run(ExploreContextInterface $context): iterable
{
    $credentials = $context->getConfig()['credentials'];
    $client = new ApiClient($credentials);

    foreach ($client->getBottles() as $bottle)
    {
        $entity = new Bottle();
        $entity->setPrimaryKey((string) $bottle['id']);
        $entity->setCapacity(new Liter($bottle['volume']));

        yield $entity;
    }
}
```

The explorer will iterate over the result of `$client->getBottles()` and construct a data set entity for every item. The primary key is set and the entity is then yielded and passed into an emission.

1.17 Dependency injection

As commonly used in the PHP community we provide a dependency injection system that allows easy reuse of utilities and services. We decided to use the Symfony dependency injection package. Read more in the ADR section about our thoughts for our decisions. The following sections require you to know basic knowledge about the Symfony package which are documented very well here.

1.17.1 Zero-configuration setup

Every service container is using a zero-configuration to allow a seamless entry into portal development. This means auto-configuration, auto-wiring, auto-binding and automatic PSR-4 resource loading is active by default. These features enable dependency injection without any setup steps for the developer.

1.17.2 How to get a service?

There are multiple utility services available for every portal node service container. Checkout the next page for a complete overview of all utility services. The examples in this section work with the PSR-3 `LoggerInterface`.

Auto-wiring

The following small status reporter implementation shows how to get an instance of a logger into the status reporter by auto-wiring:

```
namespace FooBar\StatusReporter;

use Heptacom\HeptaConnect\Portal\Base\StatusReporting\Contract\StatusReporterContract;
use Heptacom\HeptaConnect\Portal\Base\StatusReporting\Contract\StatusReportingContextInterface;
use Psr\Log\LoggerInterface;

class HealthStatusReporter extends StatusReporterContract
{
    private LoggerInterface $logger;

    public function __construct(LoggerInterface $logger)
    {
        $this->logger = $logger;
    }

    public function supportsTopic(): string
    {
        return self::TOPIC_HEALTH;
    }

    protected function run(StatusReportingContextInterface $context): array
    {
        $this->logger->warning('The status reporter has been called.');

        return [$this->supportsTopic() => true];
    }
}
```

Auto-wiring detected the `\Psr\Log\LoggerInterface` in the constructor and automatically decided to go for the logger implementation that is already available for every portal node.

Auto-configuration

The following small status reporter implementation shows how to get an instance of a logger into the status reporter by auto-configuration:

```
namespace FooBar\StatusReporter;

use Heptacom\HeptaConnect\Portal\Base\StatusReporting\Contract\StatusReporterContract;
use Heptacom\HeptaConnect\Portal\Base\StatusReporting\Contract\StatusReportingContextInterface;
use Psr\Log\LoggerAwareInterface;
use Psr\Log\LoggerAwareTrait;

class HealthStatusReporter extends StatusReporterContract implements LoggerAwareInterface
{
    use LoggerAwareTrait;
```

```
    public function supportsTopic(): string
    {
        return self::TOPIC_HEALTH;
    }

    protected function run(StatusReportingContextInterface $context): array
    {
        $this->logger->warning('The status reporter has been called.');

        return [$this->supportsTopic() => true];
    }
}
```

There is an auto-configuration rule for the `\Psr\Log\LoggerAwareInterface` interface which will later call the `setLogger` method on the instance of this class. In the snippet above there is no visible `setLogger` implementation. The missing implementation is covered by the `\Psr\Log\LoggerAwareTrait`. Eventually it is a similar way to the constructor as the logger is set right after the constructor has been called.

Auto-binding

The following is an example about accessing files. For this scenario an instance of `\Heptacom\HeptaConnect\Portal\Base\File\Filesystem\Contract\FilesystemInterface` is needed to access the files of the portal node and a configuration entry for the filename to be read from.

At first the portal definition states the `filename` as configuration:

```
namespace FooBar;

use Heptacom\HeptaConnect\Portal\Base\Portal\Contract\PortalContract;
use Symfony\Component\OptionsResolver\OptionsResolver;

class Portal extends PortalContract
{
    public function getConfigurationTemplate() : OptionsResolver
    {
        return parent::getConfigurationTemplate()
            ->setDefault('filename', 'foobar.json')
            ->setAllowedTypes('filename', 'string');
    }
}
```

The next snippet shows a service that will act as a centralized component to access the underlying data source; a JSON file:

```
namespace FooBar\Service;

use Heptacom\HeptaConnect\Portal\Base\File\Filesystem\Contract\FilesystemInterface;

class File
{
    private string $filename;

    public function __construct(FilesystemInterface $filesystem, string $configFilename)
    {
        $this->filename = $filesystem->toStoragePath($configFilename);
    }

    public function readAll(): array
    {
        return (array) json_decode(file_get_contents($this->filename) ?: '[]');
    }
}
```

This service uses auto-binding to read the values from the portal node configuration and inject it as variable into the service. The variable naming follows the pattern to add `config` as prefix and the configuration name in camelCase.

Service container

Any flow component context allows you direct access to the PSR-11 service container.

```
namespace FooBar\StatusReporter;

use Heptacom\HeptaConnect\Portal\Base\StatusReporting\Contract\StatusReporterContract;
use Heptacom\HeptaConnect\Portal\Base\StatusReporting\Contract\StatusReportingContextInterface;
use Psr\Log\LoggerInterface;

class HealthStatusReporter extends StatusReporterContract
{
    public function supportsTopic(): string
```

```
    {
        return self::TOPIC_HEALTH;
    }

    protected function run(StatusReportingContextInterface $context): array
    {
        $context->getLogger()->warning('The status reporter has been called.');

        return [$this->supportsTopic() => true];
    }
}
```

Add special attention to the implementation as it uses a `has` check before the service is acquired. This way you can have a running flow component as it adds the existence check of the service and still stays in a zero-configuration code setup. Be aware that this hides the dependency onto the logger service within the implementation of the class above.

1.17.3 Define custom services

Zero-configuration

The portal node containers make use of the PSR-4 definitions within the composer.json of the portal and portal extensions. That way any class within the referenced folders are automatically available as services:

```
<portal-dir>
├── composer.json
└── src
    ├── AcmeApi
    │   └── ApiClient.php
    ├── StatusReporter
    │   └── HealthStatusReporter.php
    └── Portal.php
```

The portal now has three services available:

- `FooBar\Portal`
- `FooBar\AcmeApi\ApiClient`
- `FooBar\StatusReporter\HealthStatusReporter`

Auto-wiring can now automatically inject an `ApiClient` instance into the `HealthStatusReporter`.

```
namespace FooBar\AcmeApi;

class ApiClient
{
    public function ping(): bool
    {
        return true;
    }
}
```

```
namespace FooBar\StatusReporter;

use FooBar\AcmeApi\ApiClient;
use Heptacom\HeptaConnect\Portal\Base\StatusReporting\Contract\StatusReporterContract;
use Heptacom\HeptaConnect\Portal\Base\StatusReporting\Contract\StatusReportingContextInterface;

class HealthStatusReporter extends StatusReporterContract
{
    private ApiClient $client;

    public function __construct(ApiClient $client)
    {
        $this->client = $client;
    }

    public function supportsTopic(): string
    {
        return self::TOPIC_HEALTH;
    }

    protected function run(StatusReportingContextInterface $context): array
    {
        return [$this->supportsTopic() => $this->client->ping()];
    }
}
```

Service argument aliases

A common pattern is to have repositories for each API resources. In the following scenario they all share the same interface `ApiResourceInterface`. When you have multiple services with the same interface, auto-wiring can't decide properly which service is the right one. In these situations it is handy to use argument aliases, so the argument names can help out. This is the very first moment you need a custom service container definition.

To load your service definition file it must be named `services.{xml,yml,yaml,php}` and it must be located inside the directory `src/Resources/config`.

The file structure should look similar to this:

```
<portal-dir>
├── composer.json
└── src
    ├── AcmeApi
    │   ├── ApiClient.php
    │   └── ApiResourceInterface.php
    │   ├── AppleRepository.php
    │   └── OrangeRepository.php
    ├── Resources
    │   └── config
    │       └── services.xml
    ├── StatusReporter
    │   └── HealthStatusReporter.php
    └── Portal.php
```

The two repositories look quite similar and are interchangeable with each other.

```
namespace FooBar\AcmeApi;

class AppleRepository implements ApiResourceInterface
{
    private ApiClient $client;

    public function __construct(ApiClient $client)
    {
        $this->client = $client;
    }

    public function findAll(): array
    {
        return $this->client->findAll('apple');
    }
}
```

```
namespace FooBar\AcmeApi;

class OrangeRepository implements ApiResourceInterface
{
    private ApiClient $client;

    public function __construct(ApiClient $client)
    {
        $this->client = $client;
    }

    public function findAll(): array
    {
        return $this->client->findAll('orange');
    }
}
```

Now the `HealthStatusReporter` requires both repositories and will render the auto-wiring invalid:

```
namespace FooBar\StatusReporter;

use FooBar\AcmeApi\ApiClient;
use FooBar\AcmeApi\ApiResourceInterface;
use Heptacom\HeptaConnect\Portal\Base\StatusReporting\Contract\StatusReporterContract;
use Heptacom\HeptaConnect\Portal\Base\StatusReporting\Contract\StatusReportingContextInterface;

class HealthStatusReporter extends StatusReporterContract
{
    private ApiClient $client;

    private ApiResourceInterface $apples;

    private ApiResourceInterface $oranges;

    public function __construct(ApiClient $client, ApiResourceInterface $apples, ApiResourceInterface $oranges)
    {
        $this->client = $client;
```

```
        $this->apples = $apples;
        $this->oranges = $oranges;
    }

    public function supportsTopic(): string
    {
        return self::TOPIC_HEALTH;
    }

    protected function run(StatusReportingContextInterface $context): array
    {
        return [
            $this->supportsTopic() => $this->client->ping(),
            'apple-count' => count($this->apples->findAll()),
            'orange-count' => count($this->oranges->findAll()),
        ];
    }
}
```

Having the following service definition it is possible to determine the difference for both services.

```
<?xml version="1.0"?>
<container
    xmlns="http://symfony.com/schema/dic/services"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://symfony.com/schema/dic/services http://symfony.com/schema/dic/services/services-1.0.xsd"
>
    <services>
        <service alias="AcmeApi\AppleRepository" id="FooBar\AcmeApi\ApiResourceInterface $apples"/>
        <service alias="AcmeApi\OrangeRepository" id="FooBar\AcmeApi\ApiResourceInterface $oranges"/>
    </services>
</container>
```

1.18 List of default utility services

You can use dependency injection to get access to various services. This is a list with brief descriptions for every default service available in the container.

1.18.1 PSR

ClientInterface

Psr\Http\Client\ClientInterface

A PSR-18 HTTP client whose implementation is based upon the choice of the bridge. Reliable service to do HTTP requests with.

RequestFactoryInterface

Psr\Http\Message\RequestFactoryInterface

A PSR-17 compliant factory that builds PSR-7 HTTP requests for the `Psr\Http\Client\ClientInterface` service.

UriFactoryInterface

Psr\Http\Message\UriFactoryInterface

A PSR-17 compliant factory that builds PSR-7 URIs for the `Psr\Http\Message\RequestFactoryInterface` service.

LoggerInterface

Psr\Log\LoggerInterface

A PSR-3 compliant logging service that logs your messages accordingly to your runtime setup.

1.18.2 HEPTAconnect portal utilities

NormalizationRegistryContract

Heptacom\HeptaConnect\Portal\Base\Serialization\Contract\NormalizationRegistryContract

Service to allow different normalization strategies. Useful to serialize objects and streams.

DeepCloneContract

Heptacom\HeptaConnect\Portal\Base\Support\Contract\DeepCloneContract

Service to clone objects.

DeepObjectIteratorContract

Heptacom\HeptaConnect\Portal\Base\Support\Contract\DeepObjectIteratorContract

Service to iterate objects like trees.

ProfilerContract

Heptacom\HeptaConnect\Portal\Base\Profiling\ProfilerContract

Service to access the profiling component to provide further detailed profiling info.

PublisherInterface

Heptacom\HeptaConnect\Portal\Base\Publication\Contract\PublisherInterface

A service to inform HEPTAconnect about the existence of entities. A publication will trigger an emission of an entity via an event driven flow.

DirectEmissionFlowContract

Heptacom\HeptaConnect\Portal\Base\Flow\DirectEmission\DirectEmissionFlowContract

A service to directly emit entities. This will skip the source emitter in a regular emission stack while the decorators will still be executed.

Psr7MessageRawHttpFormatterContract

Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\Psr7MessageRawHttpFormatterContract

Aliased as

Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\Psr7MessageFormatterContract

A service to format a PSR-7 HTTP message into a file format, that is similar to HTTP raw communication. It can be used to replay recorded requests using `nc` (netcat), `telnet` and with IDEs by Microsoft and JetBrains. See its usage in this pattern.

Psr7MessageCurlShellFormatterContract

Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\Psr7MessageCurlShellFormatterContract

A service to format a PSR-7 HTTP message into a shell script, that executes `curl` to send the request. It can be used to replay recorded requests by executing the script.

1.18.3 HEPTAconnect portal node stack specific services

PortalContract

Heptacom\HeptaConnect\Portal\Base\Portal\Contract\PortalContract

The current portal instance. It is also aliased with the real class so it works with auto-wiring.

PortalExtensionCollection

Heptacom\HeptaConnect\Portal\Base\Portal\PortalExtensionCollection

The list of active portal extensions within this container.

PortalNodeKeyInterface

Heptacom\HeptaConnect\Portal\Base\StorageKey\Contract\PortalNodeKeyInterface

The portal's portal node key instance. This can be used with multiple HEPTAconnect services and is a dependency for the following services.

PortalStorageInterface

Heptacom\HeptaConnect\Portal\Base\Portal\Contract\PortalStorageInterface

A service to store data in a key-value manner. Supports time-to-live attributes on entries to allow caching functionality.

ResourceLockFacade

Heptacom\HeptaConnect\Portal\Base\Parallelization\Support\ResourceLockFacade

A service that allows resource locking functionality to better interrupt between parallel steps.

HttpClientContract

Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\HttpClientContract

A PSR-18 based HTTP client with configuration around the original PSR-18 HTTP client. It supports following redirects, header modifications, status code based exceptions, retries on errors and response information. See reference here.

HttpHandlerUrlProviderInterface

Heptacom\HeptaConnect\Portal\Base\Web\Http\HttpHandlerUrlProviderInterface

A service that resolves HTTP handler path names into absolute URLs.

FileReferenceFactoryContract

Heptacom\HeptaConnect\Portal\Base\File\FileReferenceFactoryContract

A service that stores HTTP requests to get files, raw content of a file and public URLs to files into a file reference to process for a receiving portal. See usage here.

FileReferenceResolverContract

Heptacom\HeptaConnect\Portal\Base\File\FileReferenceResolverContract

A service that resolves a file reference created by `Heptacom\HeptaConnect\Portal\Base\File\FileReferenceFactoryContract` into an accessor to the underlying referenced file content or a public URL to access this file content. See usage here.

FilesystemInterface

Heptacom\HeptaConnect\Portal\Base\File\Filesystem\Contract\FilesystemInterface

A service, that provides methods to convert paths and URIs into each other. The URIs point to the provided portal node file system and **MUST** be used to access the file system, when these files are considered transaction data. See reference here.

1.19 Short notation for flow components

Flow components that have been described in the previous pages can also be written in a callback registration pattern. This is very useful short notation to reduce boilerplate code that is only needed to "wire" the API connecting services to the flow components. To learn more about the decisions behind this feature have a look at the related ADR.

1.19.1 How to use

For this feature a plain php file within the folder `src/Resources/flow-component/` is expected:

```
<portal-dir>
├── composer.json
└── src
    └── Resources
        └── flow-component
            └── foobar.php
```

`foobar.php` will be loaded and uses a newly introduced `FlowComponent` building utility. Every callback that is given into that builder can make use of every dependency injection feature. As the callbacks are executed by a wrapper based on the object-oriented notation, there is `$this` available pointing to the wrapping flow component instance. With this at hand, you can call other methods like `supports` or `run` from within a callback. The following section will show how to use each flow component with a file accessing scenario.

1.19.2 Explorer

Click here to see the object-oriented notation.

```php
<?php

use Heptacom\HeptaConnect\Playground\Dataset\Bottle;
use Heptacom\HeptaConnect\Portal\Base\Builder\FlowComponent;
use Heptacom\HeptaConnect\Portal\Base\File\Filesystem\Contract\FilesystemInterface;

FlowComponent::explorer(Bottle::class)->run(
    fn (FilesystemInterface $fs): iterable => scandir($fs->toStoragePath('/'))
);

FlowComponent::explorer(Bottle::class)->isAllowed(
    fn (FilesystemInterface $fs, string $id): bool => filesize($id) > 0
);
```

1.19.3 Emitter

Click here to see the object-oriented notation.

```php
<?php

use FooBar\Packer\BottlePacker;
use Heptacom\HeptaConnect\Playground\Dataset\Bottle;
use Heptacom\HeptaConnect\Playground\Dataset\Volume;
use Heptacom\HeptaConnect\Portal\Base\Builder\FlowComponent;
use Heptacom\HeptaConnect\Portal\Base\File\Filesystem\Contract\FilesystemInterface;

FlowComponent::emitter(Bottle::class)->run(
    fn (FilesystemInterface $fs, BottlePacker $packer, string $id): ?Bottle => $packer->pack(
        file_get_contents($fs->toStoragePath($id)) ?: null
    )
);

FlowComponent::emitter(Bottle::class)->batch(
    fn (FilesystemInterface $fs, BottlePacker $packer, iterable $externalIds): iterable => \iterable_map(
        $externalIds,
        fn (string $id) => $packer->pack(file_get_contents($fs->toStoragePath($id)) ?: null)
    )
);

FlowComponent::emitter(Bottle::class)->extend(
    fn (FilesystemInterface $fs, Bottle $bottle): ?Bottle => $bottle->setCapacity(
        (new Volume())
            ->setAmount(filesize($fs->toStoragePath($bottle->getPrimaryKey())))
            ->setUnit('byte')
    )
);
```

1.19.4 Receiver

Click here to see the object-oriented notation.

```
<?php

use Heptacom\HeptaConnect\Dataset\Base\TypedDatasetEntityCollection;
use FooBar\Unpacker\BottleUnpacker;
use Heptacom\HeptaConnect\Playground\Dataset\Bottle;
use Heptacom\HeptaConnect\Portal\Base\Builder\FlowComponent;
use Heptacom\HeptaConnect\Portal\Base\File\Filesystem\Contract\FilesystemInterface;

FlowComponent::receiver(Bottle::class)->run(
    function (FilesystemInterface $fs, BottleUnpacker $unpacker, Bottle $bottle): void {
        ['id' => $id, 'payload' => $payload] = $unpacker->unpack($bottle);
        file_put_contents($fs->toStoragePath($id), $payload);
    }
);

FlowComponent::receiver(Bottle::class)->batch(
    function (FilesystemInterface $fs, BottleUnpacker $unpacker, TypedDatasetEntityCollection $bottles): void {
        $bottleData = array_column(iterable_to_array($bottles->map([$unpacker, 'unpack'])), 'payload', 'id');

        foreach ($bottleData as $id => $bottle) {
            file_put_contents($fs->toStoragePath($id), $bottle);
        }
    }
);
```

1.19.5 Status reporter

Click here to see the object-oriented notation.

```
<?php

use Heptacom\HeptaConnect\Portal\Base\Builder\FlowComponent;
use Heptacom\HeptaConnect\Portal\Base\File\Filesystem\Contract\FilesystemInterface;

FlowComponent::statusReporter('health')->run(
    fn (FilesystemInterface $fs): bool => scandir($fs->toStoragePath('/')) !== false
);

FlowComponent::statusReporter('info')->run(
    fn (FilesystemInterface $fs): array => [
        'count' => count(scandir($fs->toStoragePath('/'))),
    ]
);
```

1.20 Upgrade portals

When starting development of a portal it is always useful when using the latest version of the portal base package. This guide will show you how to keep your portal up to the latest changes so you can profit from the new features.

1.20.1 Changelogs

Like every good software we provide publicly the changelogs for our open source packages as CHANGELOG.md next to the source code. They are also included in this documentation. See them in our release overview. They are written to be understood by human and machines and follow the principles of the keep a changelog proejct.

1.20.2 Applying the changelogs

Your portal makes use of a few HEPTAconnect packages at the same time, so you have to read and understand multiple changelogs. This is a big task to overview the changes and apply them. We can help you to upgrade on multiple ways:

- Each entry in the change contains a technical information like a class name and a reason for the change. This way you can relate the technical information to your code and think about the change reason and apply it to your code.
- The technical information as previously mentioned is also written to be understood by a machine. You can save a lot of time using the `check:upgrade` command in the upcoming HEPTAconnect SDK. It will skim through your code and our changelogs to supply hints to you about the upcoming upgrade.

1.21 File Reference

File references are a way to store information how to access files instead of transferring their content directly. They are useful to reduce the size of payloads in the management storage when transferring BLOBs.

1.21.1 Strategies

There are two portal node services to handle file references:
`\Heptacom\HeptaConnect\Portal\Base\File\FileReferenceFactoryContract` to create a reference and
`\Heptacom\HeptaConnect\Portal\Base\File\FileReferenceResolverContract` to resolve a reference. All resolved file references expose a public URL and the file content, so it can either be consumed by HTTP clients or the data can be read directly and passed to the next storage. Any access is only done, when used and therefore can throw exceptions. Currently, there are three strategies available.

Public URL (HTTP)

A lightweight way to transfer files as a resource is already publicly available. The reference will only be stored within the payload as there is no need to download it, just to transfer the URL. To create a file reference by HTTP URL, this method has to be used:
`\Heptacom\HeptaConnect\Portal\Base\File\FileReferenceFactoryContract::fromPublicUrl`

HTTP Request

A way to transfer files as a resource that e.g. are locked behind a login. The request will be stored serialized in the HEPTAconnect management storage, so it can be read again. When the data is fetched from the resolved file reference it will be tunneled through HEPTAconnect and the portal node service
`\Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\HttpClientContract`. It is likely that a stored request is cleaned up, so a resolved reference should be processed in a way, that it does not rely on the public URL for a longer period of time. To create a file reference by PSR-7 request, this method has to be used:
`\Heptacom\HeptaConnect\Portal\Base\File\FileReferenceFactoryContract::fromRequest`

Raw content

The last resort, that can cover any file transfer. It will store the content in the HEPTAconnect management storage, so it can be read again. It is likely that stored raw files are cleaned up, so a resolved reference should be processed in a way that it does not rely on the public URL for a longer period of time. To create a file reference by content, this method has to be used:
`\Heptacom\HeptaConnect\Portal\Base\File\FileReferenceFactoryContract::fromContents`

1.21.2 Patterns

- Transfer file reference by public URLs
- Send files from an FTP server

1.22 Filesystem

PHP can already access the filesystem, the open question is: where to place files? Each portal node has a designated location and here is how to access it.

1.22.1 Concept

Reading and writing files can be a task for a portal. The storage is not only used by portal developers but the files need also be movable for better administration in different server infrastructures. To allow integrators and administrators to safely manage the files, there has to be a way to configure the storage and keep ease of use when accessing files. There are different reasons for this directory to be movable e.g. when using a network storage across multiple HEPTAconnect app servers, so we can't just provide access to directory on disk. To accomplish this portals have the `\Heptacom\HeptaConnect\Portal\Base\File\Filesystem\Contract\FilesystemInterface` service. Read more in the ADR about the concept.

1.22.2 Protocol

The filesystem is wrapped by a stream wrapper to make it interchangeable in terms of the used storage and portal node. You likely used stream wrappers in PHP, when downloading a file via https e.g. when installing composer:

```
copy('https://getcomposer.org/installer', 'composer-setup.php');
```

This is using a stream wrapper registered on `https` to read remote files. To get your portal node specific protocol we provide `\Heptacom\HeptaConnect\Portal\Base\File\Filesystem\Contract\FilesystemInterface` as a service with two methods to convert between paths and PHP compatible URIs.

1.22.3 Location

The location can vary between different integrations. See the filesystem integrator guide to understand how to change and find the used location.

1.22.4 Patterns

- List files from filesystem
- Serve a file from filesystem using HTTP handler

1.23 Patterns

1.23.1 File reference with public URLs

This pattern shows how to:

- Let two portals transfer a file via file references
- Add configuration to a portal to toggle behaviour
- Separate API usage from entity processing

Portal A

src/Resources/flow-component/media-emit.php

```
<?php

declare(strict_types=1);

use Heptacom\HeptaConnect\Dataset\Ecommerce\Media\Media;
use Heptacom\HeptaConnect\Portal\Base\Builder\FlowComponent;
use Heptacom\HeptaConnect\Portal\Base\File\FileReferenceFactoryContract;

FlowComponent::explorer(Media::class)
    ->run(static fn (): iterable => [
        'https://picsum.photos/seed/php/300/300',
        'https://picsum.photos/seed/is/300/300',
        'https://picsum.photos/seed/nice/300/300',
    ]);

FlowComponent::emitter(Media::class)
    ->run(static function (string $externalId, FileReferenceFactoryContract $fileReferenceFactory): Media {
        $result = new Media();
        $result->setPrimaryKey($externalId);
        // externalId is a URL as used in the explorer a few lines above
        $result->setFile($fileReferenceFactory->fromPublicUrl($externalId));
        return $result;
    });
```

Portal B

src/Resources/flow-component/media-receive.php

```
<?php

declare(strict_types=1);

use Heptacom\HeptaConnect\Documentation\PortalB\Api\Client;
use Heptacom\HeptaConnect\Dataset\Base\File\FileReferenceContract;
use Heptacom\HeptaConnect\Dataset\Ecommerce\Media\Media;
use Heptacom\HeptaConnect\Portal\Base\Builder\FlowComponent;
use Heptacom\HeptaConnect\Portal\Base\File\FileReferenceResolverContract;

FlowComponent::receiver(Media::class)
    ->run(static function (Media $entity, FileReferenceResolverContract $fileReferenceResolver, Client $client, bool $configUpload): void {
        $resolvedFile = $fileReferenceResolver->resolve($entity->getFile());

        if ($configUpload) {
            $mediaLocation = $client->uploadBlob($resolvedFile);
        } else {
            $mediaLocation = $client->importBlob($resolvedFile);
        }

        $entity->setPrimaryKey($mediaLocation);
    });
```

src/Portal.php

```
<?php

declare(strict_types=1);

namespace Heptacom\HeptaConnect\Documentation\PortalB;

use Heptacom\HeptaConnect\Portal\Base\Portal\Contract\PortalContract;
use Symfony\Component\OptionsResolver\OptionsResolver;
```

```
class Portal extends PortalContract
{
    public function getConfigurationTemplate(): OptionsResolver
    {
        return parent::getConfigurationTemplate()
            ->addAllowedTypes('upload', 'bool')
            ->setDefault('upload', false);
    }
}
```

src/Api/Client.php

```
<?php

declare(strict_types=1);

namespace Heptacom\HeptaConnect\Documentation\PortalB\Api;

use Heptacom\HeptaConnect\Dataset\Base\File\FileReferenceContract;
use Heptacom\HeptaConnect\Portal\Base\Builder\FlowComponent;
use Heptacom\HeptaConnect\Portal\Base\File\FileReferenceResolverContract;
use Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\HttpClientContract;
use Psr\Http\Client\ClientInterface;
use Psr\Http\Message\RequestFactoryInterface;
use Psr\Http\Message\StreamFactoryInterface;

class Client
{
    private ClientInterface $client;

    private RequestFactoryInterface $requestFactory;

    private StreamFactoryInterface $streamFactory;

    public function __construct(
        HttpClientContract $client,
        RequestFactoryInterface $requestFactory,
        StreamFactoryInterface $streamFactory
    ) {
        $this->client = $client;
        $this->requestFactory = $requestFactory;
        $this->streamFactory = $streamFactory;
    }

    public function uploadBlob(FileReferenceContract $file): string
    {
        $request = $this->requestFactory->createRequest('POST', 'https://onlineshop.test/api/media/upload');
        $request = $request->withHeader('Content-Type', 'application/octet-stream');
        $request = $request->withBody($this->streamFactory->createStream(
            $file->getContents()
        ));
        $response = $this->client->sendRequest($request);

        return $response->getHeaderLine('Location');
    }

    public function importBlob(FileReferenceContract $file): string
    {
        $request = $this->requestFactory->createRequest('POST', 'https://onlineshop.test/api/media/import');
        $request = $request->withHeader('Content-Type', 'application/json');
        $request = $request->withBody($this->streamFactory->createStream(\json_encode([
            'url' => $file->getPublicUrl(),
        ])));
        $response = $this->client->sendRequest($request);

        return $response->getHeaderLine('Location');
    }
}
```

1.23.2 Send files from an FTP server

This pattern will focus on files from an FTP server, but your file source can really be anything. Let's assume the following problem:

A PIM system acts as your data source for products and this PIM also holds product images. These product images are stored on an FTP server and the PIM only provides you with their file paths on this FTP server.

Files on an FTP server are not accessible via HTTP, so we cannot use the source types "Public URL" or "HTTP request". So it seems, this leaves us with "File contents" as our last resort. But this would mean that files are downloaded from the FTP server to an intermediate storage and are later loaded from this intermediate storage to be sent to some destination.

In an effort to eliminate obsolete I/O operations we can utilize **HTTP handlers** to tunnel the FTP access through HTTP. Instead of downloading the file during exploration, we can instead generate a presigned URL to an HTTP handler that will perform the download later. We can then use the presigned URL as "Public URL" source. Here is all you need to make it happen.

```
use Heptacom\HeptaConnect\Dataset\Ecommerce\Media\Media;
use Heptacom\HeptaConnect\Portal\Base\Builder\FlowComponent;
use Heptacom\HeptaConnect\Portal\Base\File\FileReferenceFactoryContract;
use Heptacom\HeptaConnect\Portal\Base\Portal\Contract\PortalStorageInterface;
use Heptacom\HeptaConnect\Portal\Base\Web\Http\HttpHandlerUrlProviderInterface;
use Psr\Http\Message\ResponseInterface;
use Psr\Http\Message\ServerRequestInterface;
use Psr\Http\Message\StreamFactoryInterface;

FlowComponent::explorer(Media::class, function (
    PortalStorageInterface $portalStorage,
    HttpHandlerUrlProviderInterface $urlProvider,
    FileReferenceFactoryContract $fileReferenceFactory
): iterable {
    // Let's assume you query your data source for product media files.
    // Your data source stores media files on a FTP server and this is the file path you get.
    $filePath = 'product-data/images/12890437256/cover.jpg';

    // The plan is to generate a URL that will return the contents of your media file.
    // But the data on the FTP server must still be protected, so we add a secret token to the URL.
    // The URL will only return the image, if the query parameters contain a valid token.
    $secretToken = \bin2hex(\random_bytes(32));

    // We store the token in the portal-storage for 4 hours.
    // After this time the token is automatically invalidated.
    $portalStorage->set(
        $secretToken,
        $filePath,
        new \DateInterval('PT4H')
    );

    // Using the url-provider, we can generate a URL for an HTTP handler called "tunnel/ftp".
    // We pass the token and the file path as query parameters, so the HTTP handler can work with them.
    $presignedUrl = (string) $urlProvider->resolve('tunnel/ftp')->withQuery(\http_build_query([
        'token' => $secretToken,
        'filePath' => $filePath,
    ]));

    // Now we use the file-reference-factory to create a file-reference from our newly generated URL.
    $fileReference = $fileReferenceFactory->fromPublicUrl($presignedUrl);

    $mediaEntity = new Media();
    $mediaEntity->setPrimaryKey('12890437256_cover');
    $mediaEntity->setFile($fileReference);

    yield $mediaEntity;
});

FlowComponent::httpHandler('tunnel/ftp', function (
    ServerRequestInterface $request,
    ResponseInterface $response,
    PortalStorageInterface $portalStorage,
    StreamFactoryInterface $streamFactory,
    FtpDownloader $ftpDownloader
): ResponseInterface {
    // This HTTP handler is supposed to validate a given token and respond with the contents of the requested file path.

    $secretToken = $request->getQueryParams()['token'];
    $filePath = $request->getQueryParams()['filePath'];

    if ($portalStorage->get($secretToken) !== $filePath) {
        // The token is either not valid for the requested file path or has already expired.
        // In this case we do not send any contents but use HTTP code 401 "Unauthorized".
        return $response->withStatus(401);
    }

    // The token is valid for the requested file path.
    // We delete the token now, so the presigned URL is "read-once".
```

```
    $portalStorage->delete($secretToken);

    try {
        // Download the file using a ftp-downloader class.
        // The downloader class is not part of HEPTAconnect and must be provided by your portal.
        // Its purpose is to provide a simplified and authenticated FTP client.
        // The implementation is not shown here, because that is not the focus of this example.
        $fileContents = $ftpDownloader->downloadFile($filePath);
        $fileMimeType = $ftpDownloader->getMimeType($filePath);
    } catch (NotFoundException $exception) {
        // The file was not found on the FTP server.
        // We send no contents but use HTTP code 404 "Not Found".
        return $response->withStatus(404);
    }

    // We have successfully downloaded the file from the FTP server.
    // Now we send its contents and mime-type and use HTTP code 200 "OK".
    return $response
        ->withStatus(200)
        ->withHeader('Content-Type', $fileMimeType)
        ->withBody($streamFactory->createStream($fileContents));
});
```

1.23.3 List files from filesystem

This pattern shows how to:

- To access portal node specific filesystem using the FilesystemInterface

Portal

src/Resources/flow-component/list-files.php

```
<?php

declare(strict_types=1);

use Heptacom\HeptaConnect\Portal\Base\Builder\FlowComponent;
use Heptacom\HeptaConnect\Portal\Base\File\Filesystem\Contract\FilesystemInterface;
use Heptacom\HeptaConnect\Portal\Base\StatusReporting\Contract\StatusReporterContract;

FlowComponent::statusReporter(StatusReporterContract::TOPIC_INFO)
    ->run(static function (FilesystemInterface $fs): array {
        $files = new \RecursiveIteratorIterator(new \RecursiveDirectoryIterator($fs->toStoragePath('/')));
        $paths = [];

        /** @var \SplFileInfo $file */
        foreach ($files as $file) {
            $paths[] = $file->getPath();
        }

        return [
            StatusReporterContract::TOPIC_INFO => true,
            'files' => $paths,
        ];
    });
```

1.23.4 Serve a file from filesystem using HTTP handler

This pattern shows how to:

- To access portal node specific filesystem using the FilesystemInterface
- To response with a binary file using an HTTP handler

Portal

src/Resources/flow-component/list-files.php

```
<?php

declare(strict_types=1);

use Heptacom\HeptaConnect\Portal\Base\Builder\FlowComponent;
use Heptacom\HeptaConnect\Portal\Base\File\Filesystem\Contract\FilesystemInterface;
use Psr\Http\Message\ResponseInterface;
use Psr\Http\Message\StreamFactoryInterface;

FlowComponent::httpHandler('logo.png')
    ->get(static function (
        FilesystemInterface $fs,
        ResponseInterface $response,
        StreamFactoryInterface $streamFactory
    ): ResponseInterface {
        $path = $fs->toStoragePath('logo.png');

        if (!is_file($path)) {
            return $response->withStatus(404);
        }

        return $response
            ->withStatus(200)
            ->withHeader('Content-Type', 'image/png')
            ->withBody($streamFactory->createStreamFromFile($path));
    });
```

1.23.5 HttpClientMiddleware dumping HTTP messages on a "bad request" response

This pattern shows how to:

- access portal node specific filesystem using the FilesystemInterface
- record outbound request-response pairs using a HttpClientMiddlewareInterface
- dump HTTP messages using Psr7MessageFormatterContract

Portal

src/Http/Client/Middleware/BadRequestsDumpingMiddleware.php

```
<?php

declare(strict_types=1);

namespace Portal\Http\Client\Middleware;

use Heptacom\HeptaConnect\Portal\Base\File\Filesystem\Contract\FilesystemInterface;
use Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\HttpClientMiddlewareInterface;
use Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\Psr7MessageFormatterContract;
use Psr\Http\Client\ClientInterface;
use Psr\Http\Message\RequestInterface;
use Psr\Http\Message\ResponseInterface;

final class BadRequestsDumpingMiddleware implements HttpClientMiddlewareInterface
{
    private Psr7MessageFormatterContract $formatter;

    private FilesystemInterface $filesystem;

    public function __construct(Psr7MessageFormatterContract $formatter, FilesystemInterface $filesystem)
    {
        $this->formatter = $formatter;
        $this->filesystem = $filesystem;
    }

    public function process(RequestInterface $request, ClientInterface $handler): ResponseInterface
    {
        $response = $handler->sendRequest($request);

        if (400 <= $response->getStatusCode() && $response->getStatusCode() < 500) {
            $dumpDir = $this->filesystem->toStoragePath(sprintf('bad-requests/%s-%s-', time(), uniqid()));
            $message = $this->formatter->formatMessage($request);
            $extension = $this->formatter->getFileExtension($request);

            file_put_contents($dumpDir . 'request.' . $extension, $message);

            $message = $this->formatter->formatMessage($response);
            $extension = $this->formatter->getFileExtension($response);

            file_put_contents($dumpDir . 'response.' . $extension, $message);
        }

        return $response;
    }
}
```

1.24 Services

1.24.1 FilesystemInterface

Service to convert paths to stream wrapper prefixed URIs for portal node specific file storage access. It **MUST** be used to generate URIs, that give access to a designated filesystem, that is not shared with other portal nodes.

Service

You can get the service by id `Heptacom\HeptaConnect\Portal\Base\File\Filesystem\Contract\FilesystemInterface`.

Methods

TOSTORAGEPATH

Prefixes the path with a portal node unique PHP stream path.

FROMSTORAGEPATH

Removes the portal node unique PHP stream path scheme.

1.24.2 HttpClientContract

HTTP client that wraps around the PSR-18 `Psr\Http\Client\ClientInterface` with configurable behaviour for common use-cases.

Service

You can get the service by id `Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\HttpClientContract`. It is preconfigured to throw exceptions for status codes between 400 and 599, follow redirects up to 20 times and retry twice in case of an error or rate limit.

Methods

GETDEFAULTREQUESTHEADERS

Get the request header configurations that are applied to any request unless they are already present.

WITHDEFAULTREQUESTHEADERS

Set the request header configurations that are applied to any request unless they are already present. As it returns a new instance of itself, you **SHOULD** process its return value.

GETEXCEPTIONTRIGGERS

Get the HTTP response status codes that will throw an exception.

WITHEXCEPTIONTRIGGERS

Add HTTP response status codes that will throw an exception. As it returns a new instance of itself, you **SHOULD** process its return value.

WITHOUTEXCEPTIONTRIGGERS

Remove HTTP response status codes, so they will not throw an exception. As it returns a new instance of itself, you **SHOULD** process its return value.

GETMAXREDIRECT

Get the number of automatically followed redirects. Defaults to 0.

WITHMAXREDIRECT

Sets the number of automatically followed redirects. As it returns a new instance of itself, you **SHOULD** process its return value.

GETMAXRETRY

Get the number of automatically processed retries. Defaults to 0.

WITHMAXRETRY

Sets the number of automatically processed retries. As it returns a new instance of itself, you **SHOULD** process its return value.

GETMAXWAITTIMEOUT

Get the maximum time in seconds allowed to wait between retries per HTTP status code.

WITHMAXWAITTIMEOUT

Add the maximum time allowed timeout in seconds for an HTTP status code. As it returns a new instance of itself, you **SHOULD** process its return value.

WITHOUTMAXWAITTIMEOUT

Remove the wait timeout for an HTTP status code. As it returns a new instance of itself, you **SHOULD** process its return value.

2. Integrator

2.1 Integrate HEPTAconnect into your project

This is the place to learn how to structure your project's dependencies. Learn to decide how to decide which portals to integrate and how to optimize your hosting scenario.

2.1.1 Portals

Get to know the building blocks of your integration.

2.1.2 Bridges

Compare your project to our templates and learn how to integrate your scenario into your project.

2.1.3 Message broking

Messages are one of the main scaling factors. Learn to create high-performance message broker scenarios.

2.1.4 HTTP Handlers

Exposed HTTP endpoints from portals can be debugged. Also in production environments. Learn how to trace and replay requests.

2.1.5 Filesystem

Portals can store files on disk. Learn how to integrate network storages.

2.1.6 Portal node configuration

Learn to use environment variables and other sources for portal node configurations.

2.1.7 Logging

Logging is key for understanding applications. Get enabled to prepare logged data in scaled scenarios to take action when your application most needs it.

2.1.8 Upgrade

Master the changelogs with our tools to upgrade your integration to the next version.

2.2 Portals

Portals are the pieces of code to connect your HEPTAconnect instance to other APIs for data transfer. They are most likely installed via composer or as a plugin in your integration (e.g. Shopware 6).

2.2.1 How to get a portal?

There are multiple sources for portals. Some of them are available as open source on GitHub and packagist like our Shopware 6 portal. There are many client specific implementations for known and custom APIs we (HEPTACOM GmbH) and our HEPTAconnect partners developed in the past. To get information on our previous work and our solutions for your connector project contact us by emailing us to info@heptacom.de.

2.2.2 Develop your own

When you want to develop your portal you can move over from this integrator section over to the portal developer section. There you can find an extensive explanation of all the tools at hand that you need.

2.2.3 Usage

When you got access to your portal of choice you can now use composer to install it in your integration. To check whether it got recognized correctly you can check `heptaconnect:portal:list` to list all your installed portals. Following by that you can create portal nodes from this portal and assign a rememberable name with the command `heptaconnect:portal-node:add $FQCN nice_alias`. Learn more about administering portal nodes in the administrator section.

There are good reasons to alter the behaviour of an existing portal. For this task you use portal extensions. They allow you to completely change the behaviour of any portal and can be mixed with other portal extensions as well. Learn more about decorated flow components like explorers, emitters and receivers in the portal developer section. Learn more about the reasons why and when it is useful to create portal extensions for data tuning.

2.3 Bridges

Bridges are solid building blocks to build connections on. HEPTAconnect bridges provide the technological ground the core shall be used on. There are ready-to-use bridges that can be used right away in your integration.

2.3.1 How to get a bridge?

There are multiple sources for bridges. Some of them are available as open source on GitHub and packagist like our Shopware 6 bridge. There are also client specific implementations we (HEPTACOM GmbH) and our HEPTAconnect partners develop as well. To get information on our previous work and our solutions for your connector project contact us by emailing us to info@heptacom.de.

2.3.2 Shopware 6

The first and most used bridge that ships with a data layer build on top of the Shopware 6 DAL. It is very useful and easy to integrate as it exposes itself as Shopware bundle that can be easily used as a plugin. Ship it as bundle in your project or build a self-containing plugin with it and integrate it in your environment of choice.

2.3.3 Laravel 8

To get information on our work on our Laravel 8 bridge contact us by emailing us to info@heptacom.de.

2.3.4 Symfony 5

To get information on our work on our Symfony 5 bridge contact us by emailing us to info@heptacom.de.

2.4 Message broking

A message broking system has three types of participants: Sender, broker and consumer. HEPTAconnect builds upon message broking for task splitting over multiple processing units. Learn how to integrate message broking into your project.

2.4.1 Integrate a message broker

There is no all-fit solution as this heavily depends on the development and hosting environment in your project. Nonetheless, we can provide useful tips for your project from our experience.

2.4.2 Choose a message broker

For local development it is useful to use a relational-database-driven message broker. That makes it more comprehensive as it enables quick access to the message content. The downside is that a relational-database-engine is not well optimized for message broking, so it will neither be fast nor well performing with multiple message consumers.

For production environments you should ask your hosting provider what good services they have at hand. In the past we made good experience with Redis, RabbitMQ and Amazon SQS.

2.4.3 Bridge support

Depending on the bridge you choose the configuration differs. You will probably find your use-case below and copy the requirements. It is common to use multiple solutions in the same project to allow different environments like local development and production hosting. Changing between the solutions is best done via environment variables. Ensure to document the message broker, so you can get new persons aboard nicely.

Shopware 6 - Database

Shopware by default ships with the enqueue library and a database table called `enqueue`. This allows for no additional required work to use HEPTAconnect with a message broker.

Shopware 6 - Redis

Shopware by default ships with the enqueue library so the following is an explanation how to configure it. This example expects a Redis service running on the local system `127.0.0.1`, is accessible on the port `6379` and use the database `1`. Configure the following files that are placed in your Shopware 6 project:

```
pecl install redis
```

config/packages/enqueue.yaml **config/packages/framework.yaml** **.env**

```
enqueue:
    redis:
        transport: '%env(MESSAGE_BROKER_REDIS_URL)%'
        client: ~

framework:
    messenger:
        transports:
            default:
                dsn: '%env(MESSAGE_BROKER_DSN)%'

MESSAGE_BROKER_REDIS_URL="redis://127.0.0.1:6379/1"
MESSAGE_BROKER_DSN="enqueue://redis"
```

Shopware 6 - RabbitMQ

Shopware by default ships with the enqueue library so the following is an explanation how to configure it. This example expects a RabbitMQ service running on the local system `127.0.0.1`, is accessible on the port `5672` with the credentials `guest` / `guest`. Configure the following files that are placed in your Shopware 6 project:

```
pecl install amqp
```

config/packages/enqueue.yaml **config/packages/framework.yaml** **.env**

```
enqueue:
    rabbitmq:
        transport:
            dsn: '%env(MESSAGE_BROKER_RABBITMQ_URL)%'
        client: ~

framework:
    messenger:
        transports:
            default:
                dsn: '%env(MESSAGE_BROKER_DSN)%'

MESSAGE_BROKER_RABBITMQ_URL="amqp://guest:guest@127.0.0.1:5672/%2F?connection_timeout=1000&heartbeat=100"
MESSAGE_BROKER_DSN="enqueue://rabbitmq?queue[name]=heptaconnect"
```

2.5 Filesystem

Portals can make use of the filesystem. Scaling to an app server cluster expects the use of a network file storage.

2.5.1 Concept

Reading and writing files can be a task for a portal. The storage is movable for better administration in different server infrastructures through an abstraction layer for portal developers. To allow administrators of your project to safely manage the files, there has to be a way to configure the used storage. By default the bridges store the portal node filesystems in a directory within the project root directory. When setting up an app server cluster, you need to enable the administrator to configure a network accessible storage. When changing the storage you should document it properly so the administrator of your project can set up accordingly with the related administration guide. Read more in the ADR about the concept.

2.5.2 Sample configurations

Shopware 6 Bridge

The Shopware 6 bridge exposes itself as a Shopware bundle and makes use of the automatically provided private filesystem. In general the files are placed in `<instance-dir>/files/plugins/heptaconnect_bridge_shopware_platform/` with a subdirectory for each portal node.

```
<system-root>
└── …
    └── <instance-dir>
        └── files
            └── plugins
                └── heptaconnect_bridge_shopware_platform
                    ├── <portal-node-1>
                    ├── <portal-node-2>
                    └── <portal-node-3>
```

We suggest to control the storage location by following the hosting guide from Shopware on the shared filesystem.

2.5.3 Patterns

- Change the filesystem for a specific portal node

2.6 HTTP Handlers

HEPTAconnect exposes HTTP endpoints by portals. Handling of these HTTP handlers can be tuned for e.g. debugging.

2.6.1 Debugging dumps of HTTP messages

HEPTAconnect ships a request-response dump feature for HTTP handlers using request attributes defined in `\Heptacom\HeptaConnect\Core\Web\Http\Contract\HttpHandleServiceInterface`. This attribute is already set by features provided by bridges e.g. using a sampling rate.

Dump format

The dumped HTTP messages are formatted using the `Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\Psr7MessageFormatterContract` service. You can override this service definition to change the formatting of the dumped messages. Without any changes the dumped messages are in a raw HTTP format, so it could be used together with `nc` (netcat), `openssl` or `telnet` to replay the requests. As the named tools do just TCP and do not fully perform HTTP, beware that you have to provide the TCP connection information like host and port yourself. If you want to replay messages with `curl` you can use the `Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\Psr7MessageCurlShellFormatterContract` service instead. Read here about recording. The dumped request can be passed into the standard input using shell pipes:

IDEs **netcat** **openssl** **telnet**

The HTTP request dump can be opened and replayed in IDEs from JetBrains like WebStorm and PHPStorm and Microsoft Visual Studio (17.5+ required).

```
cat dump.http | netcat localhost 80 # Linux
cat dump.http | nc localhost 80 # macOS
cat dump.http | netcat --ssl localhost 443 # Linux to an HTTPS secured server

cat dump.http | openssl s_client -connect localhost:443

telnet localhost 80 # paste file content
```

Trigger dumps

If you want to introduce a new trigger to dump HTTP messages, you can decorate the `\Heptacom\HeptaConnect\Core\Web\Http\Contract\HttpHandleServiceInterface` service. Combining triggers of other development tools like SPX or Xdebug is a helpful approach. Read about the integration of SPX here and about the integration of Xdebug here. Adjusting the trigger to your needs is a good way to reduce the amount of dumped messages. We also provide an example to show how to dump HTTP messages only on errors.

2.7 Portal node configuration

Portal node configuration are configured using command line commands and are persistent in the storage layer. Common strategies to create staging, testing and development systems depend on configuration by environment variables. Here you can learn how to switch portal node configuration by other configuration sources. This page is separated in four sections to explain how to these few lines can configure a portal node by environment variables:

```
<?php

declare(strict_types=1);

use Heptacom\HeptaConnect\Core\Bridge\PortalNode\Configuration\Config;

Config::replace('bottle', Config::helper()->env([
    'black' => 'PORTAL_BOTTLE_BLACK',
]));
```

1. How to use in my project?
2. How to identity a portal node to configure?
3. Where to load data from?
4. How to combine different sources?

2.7.1 Bridge support

Depending on the bridge you choose the configuration differs. You will probably find your use-case below and copy the requirements. Ensure to document the newly integrated configuration sources, so you can get new persons aboard nicely.

Shopware 6

The Shopware bridge by default ships with a services that collects configuration source services by tagged services. This allows for a few changes to introduce new configuration sources. The following example shows what you need to add portal node configuration by a short-notation configuration script.

config/services.yaml **config/services.xml (alternative)**

```
Heptacom\HeptaConnect\Core\Bridge\PortalNode\Configuration\InstructionFileLoader:
    tags:
        - { name: heptaconnect_core.portal_node_configuration.instruction_file_loader }
    arguments:
        - '%kernel.project_dir%/config/portal-node-config.php'

<?xml version="1.0" ?>
<container
    xmlns="http://symfony.com/schema/dic/services"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://symfony.com/schema/dic/services http://symfony.com/schema/dic/services/services-1.0.xsd"
>
    <services>
        <service id="Heptacom\HeptaConnect\Core\Bridge\PortalNode\Configuration\InstructionFileLoader">
            <tag name="heptaconnect_core.portal_node_configuration.instruction_file_loader"/>
            <argument>%kernel.project_dir%/config/portal-node-config.php</argument>
        </service>
    </services>
</container>
```

config/portal-node-config.php

```
<?php

declare(strict_types=1);

use Heptacom\HeptaConnect\Core\Bridge\PortalNode\Configuration\Config;

// TODO enter your Config:: instructions
```

2.7.2 Portal node query

Portal nodes that shall be affected by the configuration instructions can be matched against different queries. The query to match against is the first parameter in every configuration instruction.

Portal class

You can identify a portal node by its class. This query will match every portal node of the same type and can be used to change configuration for multiple portal nodes at once. The referenced class does not need to be the portal class itself but can be anything extended class name or interface that is related to the portal class of the portal node.

```
<?php

declare(strict_types=1);

use Heptacom\HeptaConnect\Core\Bridge\PortalNode\Configuration\Config;
use Heptacom\HeptaConnect\Playground\Portal\BottlePortal;

Config::replace(BottlePortal::class, [
    'black' => '#111111',
]);
```

Portal extension class

You can identify a portal node by its active portal extensions. This query will match every portal node that has a certain activated portal extension attached to it and can be used to change configuration for multiple portal nodes at once. The referenced class does not need to be the portal extension class itself but can be anything extended class name or interface that is related to the portal extension class of the portal node.

```
<?php

declare(strict_types=1);

use Heptacom\HeptaConnect\Core\Bridge\PortalNode\Configuration\Config;
use Heptacom\HeptaConnect\Playground\PortalExtension\BottleContent;

Config::replace(BottleContent::class, [
    'contentFactor' => 1.0,
]);
```

Portal node key

You can identify a portal node by its key given in the storage layer. This key can be seen when creating a portal node or listing the portal nodes. Using the portal node key is the most specific way to configure a portal node. Therefore, this only works with an existing database.

```
<?php

declare(strict_types=1);

use Heptacom\HeptaConnect\Core\Bridge\PortalNode\Configuration\Config;

Config::replace('PortalNode:1234567890', [
    'black' => '#111111',
]);
```

Portal node alias

You can identify a portal node by its alias. The portal node alias points to a portal node key and is unique as well. This key can be defined when creating a portal node or seen when listing the portal nodes.

```
<?php

declare(strict_types=1);

use Heptacom\HeptaConnect\Core\Bridge\PortalNode\Configuration\Config;

Config::replace('bottle', [
    'black' => '#111111',
]);
```

2.7.3 Configuration sources

As seen in the initial configuration for the support of your integration, configuration happens on PHP level. PHP as configuration source is the most versatile. To simplify usage of other sources you have the following tooling available:

Environment variables

Environment variables are the most common alternative configuration source.

```
<?php

declare(strict_types=1);

use Heptacom\HeptaConnect\Core\Bridge\PortalNode\Configuration\Config;

$mapping = [
    'username' => 'PORTAL_A_USERNAME',
    'password' => 'PORTAL_A_PASSWORD',
];
$source = Config::helper()->env($mapping);
```

With the mapping array you can provide a reusable pattern, where to fetch data from. It is equivalent to:

```
<?php

$source = [
    'username' => getenv('PORTAL_A_USERNAME'),
    'password' => getenv('PORTAL_A_PASSWORD'),
];
```

Array

Restructuring arrays is a flexible way to introduce various configuration sources. In the example is a statically provided `$data` variable. This can be loaded on different ways.

```
<?php

declare(strict_types=1);

use Heptacom\HeptaConnect\Core\Bridge\PortalNode\Configuration\Config;

$data = [
    'list' => [1, 2, 3],
    'assoc' => [
        'key' => 'value',
        'secret' => 'letmein',
    ],
    'not' => 'needed',
];
$mapping = [
    'username' => 'assoc.key',
    'password' => 'assoc.secret',
    'logging' => [
        'levels' => 'list',
    ],
];
$source = Config::helper()->array($data, $mapping);
```

With the mapping array you can provide a reusable pattern, where to fetch data from. It is equivalent to:

```
<?php

$data = [
    'list' => [1, 2, 3],
    'assoc' => [
        'key' => 'value',
        'secret' => 'letmein',
    ],
    'not' => 'needed',
];
$source = [
    'username' => $data['assoc']['key'] ?? null,
    'password' => $data['assoc']['secret'] ?? null,
    'logging' => [
        'levels' => $data['list'] ?? null,
    ],
];
```

JSON

JSON files are handled very similar compared to arrays as source. The source only refers to a file instead of static data.

```
<?php

declare(strict_types=1);

use Heptacom\HeptaConnect\Core\Bridge\PortalNode\Configuration\Config;

$mapping = [
    'username' => 'assoc.key',
    'password' => 'assoc.secret',
    'logging' => [
        'levels' => 'list',
    ],
];
$source = Config::helper()->json(__DIR__ . '/config.json', $mapping);
```

INI

INI files are handled very similar compared to arrays as source. The source only refers to a file instead of static data.

```
<?php

declare(strict_types=1);

use Heptacom\HeptaConnect\Core\Bridge\PortalNode\Configuration\Config;

$mapping = [
    'username' => 'assoc.key',
    'password' => 'assoc.secret',
    'logging' => [
        'levels' => 'list',
    ],
];
$source = Config::helper()->ini(__DIR__ . '/config.ini', $mapping);
```

2.7.4 Configuration chain

Every instruction you make in the short-notation is processed in order of definition. The very first source is the cached access of reading the storage layer. After that every manipulation that matches the portal node query is applied in a decoration chain. Each data source referenced in the instruction can be:

- either a `Closure` to reference data pulled from a data store that must not be queried on definition
- or an `array` of statically provided data

Set configuration

The `set` instruction is the most versatile instruction. With great versatility comes great responsibility. This set instruction will break the configuration chain when used with static data or a closure that does not call the next step:

```
<?php

declare(strict_types=1);

use Heptacom\HeptaConnect\Core\Bridge\PortalNode\Configuration\Config;

Config::set('bottle', [
    'black' => '#111111',
]);
```

```
<?php

declare(strict_types=1);

use Heptacom\HeptaConnect\Core\Bridge\PortalNode\Configuration\Config;

Config::set('bottle', static fn (Closure $next) => [
    'black' => '#111111',
]);
```

This means that you can prevent loading from the storage layer entirely.

Merge configurations

The `merge` instruction is a short-notation for a chaining set instruction with `array_merge` and `array_merge_recursive`. If not configured differently `array_merge_recursive` is used.

```
<?php

declare(strict_types=1);

use Heptacom\HeptaConnect\Core\Bridge\PortalNode\Configuration\Config;

Config::set('bottle', static fn (Closure $next) => \array_merge_recursive(
    $next(),
    [
        'black' => '#111111',
    ]
));
Config::merge('bottle', [
    'black' => '#111111',
]);
```

Replace configurations

The `replace` instruction is a short-notation for a chaining set instruction with `array_replace` and `array_replace_recursive`. If not configured differently `array_replace_recursive` is used.

```
<?php

declare(strict_types=1);

use Heptacom\HeptaConnect\Core\Bridge\PortalNode\Configuration\Config;

Config::set('bottle', static fn (Closure $next) => \array_replace_recursive(
    $next(),
    [
        'black' => '#111111',
    ]
));
Config::replace('bottle', [
    'black' => '#111111',
]);
```

Reset configurations

The `reset` instruction is a short-notation for a chaining set instruction with a mapped calls of unset to remove data from the previous configuration.

```
<?php

declare(strict_types=1);

use Heptacom\HeptaConnect\Core\Bridge\PortalNode\Configuration\Config;

Config::reset('bottle', static function (Closure $next) {
    $previous = $next();
    unset($previous['black']);
    return $previous;
});
Config::reset('bottle', ['black']);
```

2.8 Logging

Logging is a crucial feature to understand actions of the application's insides. Having a strategy for log inspection should be part of any project and prepared quite early as this speeds up development process.

2.8.1 Concept

HEPTAconnect expects to have a PSR compliant logger to send all messages to. The interface `\Psr\Log\LoggerInterface` provided from PSR is the abstraction layer that expects the bridge to provide a logging implementation and allows the option to change the logger implementation by the integration. Bridges by default fallback to log files placed on the filesystem to always have a solution running out of the box. Integrations should specify a hosting-optimized logging facility that e.g. are scaling better. All our currently available bridges ship with the monolog library which allows for a quick setup for alternative logging providers. When changing the logging facility you should document it properly so the administrator of your project can set up accordingly with the related administration guide. Log messages frequently contain unique codes that point to the origin of a message or an exception. You can read more about them in a news entry and this ADR.

2.8.2 Sample configurations

Graylog

Graylog is a service that can be setup quickly and provides log querying, dashboards and alerts over network and therefore can be used with multiple application instances. It allows a production installation but also a setup for a local/sneak-peek environment with the following docker-compose specification.

```
version: '3'
services:
    mongo:
        image: mongo:4.2
        networks:
            - graylog
    elasticsearch:
        image: docker.elastic.co/elasticsearch/elasticsearch-oss:7.10.2
        environment:
            - http.host=0.0.0.0
            - transport.host=localhost
            - network.host=0.0.0.0
            - "ES_JAVA_OPTS=-Xms512m -Xmx512m"
        ulimits:
            memlock:
                soft: -1
                hard: -1
        deploy:
            resources:
                limits:
                    memory : 1g
        networks:
            - graylog
    graylog:
        image: graylog/graylog:4.1
        environment:
            - GRAYLOG_PASSWORD_SECRET=somepasswordpepper
            - GRAYLOG_ROOT_PASSWORD_SHA2=8c6976e5b5410415bde908bd4dee15dfb167a9c873fc4bb8a81f6f2ab448a918
            - GRAYLOG_HTTP_EXTERNAL_URI=http://127.0.0.1:9000/
        networks:
            - graylog
        restart: always
        depends_on:
            - mongo
            - elasticsearch
        ports:
            - 9000:9000
            - 12201:12201
            - 12201:12201/udp
networks:
    graylog:
        driver: bridge
```

This will start the graylog webservice, which is hosted behind the URI of the environment variable `GRAYLOG_HTTP_EXTERNAL_URI`, and an additional port for log feeding. The default credentials for this environment is `admin` / `admin`. After starting the containers you have to define an input in graylog. For this example we use the gelf protocol over UDP with the graylog default configuration.

The used gelf protocol expects this additional composer requirement `graylog2/gelf-php`. As integrator, you can now start to override the bridge's logger definition `heptacom_heptaconnect.logger`. This should depend on environment variables like `GELF_HOSTNAME` and `GELF_PORT` and can be implemented like this:

config/services.yaml **config/services.xml (alternative)** **.env**

```
heptacom_heptaconnect.logger:
    class: Monolog\Logger
    arguments:
        - 'heptacom_heptaconnect'
        -
            -   !service
                class: Monolog\Handler\GelfHandler
                arguments:
                    -   !service
                        class: Gelf\Publisher
                        arguments:
                            -   !service
                                class: Gelf\Transport\UdpTransport
                                arguments:
                                    - '%env(string:GELF_HOSTNAME)%'
                                    - '%env(int:GELF_PORT)%'

<?xml version="1.0" ?>
<container
    xmlns="http://symfony.com/schema/dic/services"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://symfony.com/schema/dic/services http://symfony.com/schema/dic/services/services-1.0.xsd"
>
    <services>
        <service id="heptacom_heptaconnect.logger" class="Monolog\Logger">
            <argument>heptacom_heptaconnect</argument>
            <argument type="collection">
                <argument type="service">
                    <service class="Monolog\Handler\GelfHandler">
                        <argument type="service">
                            <service class="Gelf\Publisher">
                                <argument type="service">
                                    <service class="Gelf\Transport\UdpTransport">
                                        <argument type="string">%env(string:GELF_HOSTNAME)%</argument>
                                        <argument type="string">%env(int:GELF_PORT)%</argument>
                                    </service>
                                </argument>
                            </service>
                        </argument>
                    </service>
                </argument>
            </argument>
        </service>
    </services>
</container>

# Docker host ip-address
GELF_HOSTNAME=127.0.0.1
# same as graylog configured input port and docker port-forwarding
GELF_PORT=12201
```

2.9 Upgrade integrations

Planning an integration does not just take now into consideration but also the future. This guide will show you how to upgrade your integration into future versions of HEPTAconnect.

2.9.1 Changelogs

Like every good software we provide publicly the changelogs for our open source packages as CHANGELOG.md next to the source code. They are also included in this documentation. See them in our release overview. They are written to be understood by human and machines and follow the principles of the keep a changelog proejct.

2.9.2 Applying the changelogs

Your integration makes use of multiple HEPTAconnect packages at the same time, so you have to read and understand multiple changelogs. This is a big task to overview the changes and apply them. We can help you to upgrade your integration on multiple ways:

- You probably have a running project that has been made by us (HEPTACOM GmbH) or our partners. In that case we probably planned the update for your project already.
- Each entry in the change contains a technical information like a class name and a reason for the change. This way you can relate the technical information to your code and think about the change reason and apply it to your code.
- The technical information as previously mentioned is also written to be understood by a machine. You can save a lot of time using the `check:upgrade` command in the upcoming HEPTAconnect SDK. It will skim through your code and our changelogs to supply hints to you about the upcoming upgrade.

2.10 Patterns

2.10.1 Change the filesystem for a specific portal node

This pattern shows how to:

- Use Flysystem v1 to connect to an FTP server. Learn more in the Flysystem documentation.
- Decorate a service to return an FTP connection as storage for a specific portal node
- Identify portal nodes within an integration

Integration

src/Core/PortalNodeFilesystemStorageFactory.php

```
<?php

declare(strict_types=1);

namespace Heptacom\HeptaConnect\Production\Core;

use Heptacom\HeptaConnect\Core\Storage\Filesystem\FilesystemFactory;
use Heptacom\HeptaConnect\Portal\Base\StorageKey\Contract\PortalNodeKeyInterface;
use Heptacom\HeptaConnect\Storage\Base\Contract\StorageKeyGeneratorContract;
use League\Flysystem\Adapter\Ftp;
use League\Flysystem\Filesystem;

class PortalNodeFilesystemStorageFactory extends FilesystemFactory
{
    private StorageKeyGeneratorContract $storageKeyGenerator;

    private FilesystemFactory $decorated;

    private string $ftpDsn;

    public function __construct(
        StorageKeyGeneratorContract $storageKeyGenerator,
        FilesystemInterface $filesystem,
        FilesystemFactory $decorated,
        string $ftpDsn
    ) {
        parent::__construct($storageKeyGenerator, $filesystem);

        $this->storageKeyGenerator = $storageKeyGenerator;
        $this->decorated = $decorated;
        $this->ftpDsn = $ftpDsn;
    }

    public function factory(PortalNodeKeyInterface $portalNodeKey): FilesystemInterface
    {
        $portalNodeAlias = $this->storageKeyGenerator->serialize($portalNodeKey->withAlias());

        if ($portalNodeAlias !== 'portal-node-a') {
            return $this->decorated->factory($portalNodeKey);
        }

        if ($this->ftpDsn === '') {
            return $this->decorated->factory($portalNodeKey);
        }

        $dsnParts = parse_url($this->ftpDsn);

        return new Filesystem(new Ftp([
            'host' => $dsnParts['host'],
            'username' => $dsnParts['user'],
            'password' => $dsnParts['pass'],
            'port' => $dsnParts['port'] ?? 21,
            'root' => $dsnParts['path'] ?? null,
            'passive' => true,
            'ssl' => true,
        ]));
    }
}
```

src/Resources/config/services.xml

```
<?xml version="1.0" ?>
<container
    xmlns="http://symfony.com/schema/dic/services"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://symfony.com/schema/dic/services http://symfony.com/schema/dic/services/services-1.0.xsd"
```

```
>
    <services>
        <service
            decorates="Heptacom\HeptaConnect\Core\Storage\Filesystem\FilesystemFactory"
            id="Heptacom\HeptaConnect\Production\Core\PortalNodeFilesystemStorageFactory"
            parent="Heptacom\HeptaConnect\Core\Storage\Filesystem\FilesystemFactory"
        >
            <argument type="service" id="Heptacom\HeptaConnect\Production\Core\PortalNodeFilesystemStorageFactory.inner"/>
            <argument type="string">%env(string:PORTAL_NODE_A_FTP_DSN)%</argument>
        </service>
    </services>
</container>
```

.env

```
PORTAL_NODE_A_FTP_DSN=ftp://user:pass@other-server/subdir
```

2.10.2 Change the dump format for HTTP handler communication to cURL shell

This pattern shows how to:

- Change a service alias to set format of HTTP message dumping to `curl` shell scripts

Integration

config/services.?

config/services.xml **config/services.yaml**

```
<?xml version="1.0" ?>
<container
    xmlns="http://symfony.com/schema/dic/services"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://symfony.com/schema/dic/services http://symfony.com/schema/dic/services/services-1.0.xsd"
>
    <services>
        <service
            id="Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\Psr7MessageFormatterContract"
            alias="Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\Psr7MessageCurlShellFormatterContract"
        />
    </services>
</container>

Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\Psr7MessageFormatterContract:
    alias: Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\Psr7MessageCurlShellFormatterContract
```

2.10.3 Use SPX extension as trigger for HTTP handler dumps

This pattern shows how to:

- Replace the `ServerRequestCycleDumpCheckerInterface` service to conditionally trigger dumps of HTTP requests
- Identify whether SPX is used for tracing to set the dump request attribute accordingly

Integration

src/Core/SpxWebHttpDumpChecker.php

```
<?php

declare(strict_types=1);

namespace Heptacom\HeptaConnect\Production\Core;

use Heptacom\HeptaConnect\Core\Web\Http\Dump\Contract\ServerRequestCycleDumpCheckerInterface
use Heptacom\HeptaConnect\Portal\Base\Web\Http\HttpHandlerStackIdentifier;
use Heptacom\HeptaConnect\Portal\Base\Web\Http\ServerRequestCycle;

final class SpxWebHttpDumpChecker implements ServerRequestCycleDumpCheckerInterface
{
    private bool $spxEnabled;

    private bool $spxAutoStart;

    public function __construct(bool $spxEnabled, bool $spxAutoStart)
    {
        $this->spxEnabled = $spxEnabled;
        $this->spxAutoStart = $spxAutoStart;
    }

    public function shallDump(HttpHandlerStackIdentifier $httpHandler, ServerRequestCycle $requestCycle): bool
    {
        return $this->isSpxActive();
    }

    private function isSpxActive(): bool
    {
        if (!\extension_loaded('spx')) {
            return false;
        }

        return $this->spxEnabled && $this->spxAutoStart;
    }
}
```

config/services.?

config/services.xml **config/services.yaml**

```
<?xml version="1.0" ?>
<container
    xmlns="http://symfony.com/schema/dic/services"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://symfony.com/schema/dic/services http://symfony.com/schema/dic/services/services-1.0.xsd"
>
    <services>
        <service
            id="Heptacom\HeptaConnect\Core\Web\Http\Dump\Contract\ServerRequestCycleDumpCheckerInterface"
            class="Heptacom\HeptaConnect\Production\Core\SpxWebHttpDumpChecker"
        >
            <argument>%env(bool:SPX_ENABLED)%</argument>
            <argument>%env(bool:SPX_AUTO_START)%</argument>
        </service>
    </services>
</container>

Heptacom\HeptaConnect\Core\Web\Http\Dump\Contract\ServerRequestCycleDumpCheckerInterface:
    class: Heptacom\HeptaConnect\Production\Core\SpxWebHttpDumpChecker
    arguments:
        - '%env(bool:SPX_ENABLED)%'
        - '%env(bool:SPX_AUTO_START)%'
```

2.10.4 Use Xdebug extension as trigger for HTTP handler dumps

This pattern shows how to:

- Replace the `ServerRequestCycleDumpCheckerInterface` service to conditionally trigger dumps of HTTP requests
- Identify whether Xdebug is used for debugging to set the dump request attribute accordingly

Integration

src/Core/XdebugWebHttpDumpChecker.php

```
<?php

declare(strict_types=1);

namespace Heptacom\HeptaConnect\Production\Core;

use Heptacom\HeptaConnect\Core\Web\Http\Dump\Contract\ServerRequestCycleDumpCheckerInterface
use Heptacom\HeptaConnect\Portal\Base\Web\Http\HttpHandlerStackIdentifier;
use Heptacom\HeptaConnect\Portal\Base\Web\Http\ServerRequestCycle;

final class XdebugWebHttpDumpChecker implements ServerRequestCycleDumpCheckerInterface
{
    public function shallDump(HttpHandlerStackIdentifier $httpHandler, ServerRequestCycle $requestCycle): bool
    {
        return $this->isXdebugEnabled();
    }

    private function isXdebugEnabled(): bool
    {
        if (!\extension_loaded('xdebug')) {
            return false;
        }

        if (!\function_exists('xdebug_info')) {
            return false;
        }

        $xdebugMode = \xdebug_info('mode');

        return !empty($xdebugMode);
    }
}
```

config/services.?

config/services.xml **config/services.yaml**

```
<?xml version="1.0" ?>
<container
    xmlns="http://symfony.com/schema/dic/services"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://symfony.com/schema/dic/services http://symfony.com/schema/dic/services/services-1.0.xsd"
>
    <services>
        <service
            id="Heptacom\HeptaConnect\Core\Web\Http\Dump\Contract\ServerRequestCycleDumpCheckerInterface"
            class="Heptacom\HeptaConnect\Production\Core\XdebugWebHttpDumpChecker"
        />
    </services>
</container>

Heptacom\HeptaConnect\Core\Web\Http\Dump\Contract\ServerRequestCycleDumpCheckerInterface:
    class: Heptacom\HeptaConnect\Production\Core\XdebugWebHttpDumpChecker
```

3. Administrator

3.1 Administer HEPTAconnect

This is the place to learn about commands and analytical methods to work with a HEPTAconnect instance.

3.1.1 Portal nodes

Learn what portals and portal nodes are and how to configure them to work your way.

3.1.2 Data routing

Learn how to connect the data routes the way the data shall flow.

3.1.3 Instance status

Understand how well your HEPTAconnect instance performs and detect issues.

3.1.4 HTTP APIs

Manage and investigate into hosted HTTP endpoints by portals.

3.1.5 Filesystem

Portals can store files on disk. Learn how to integrate network storages.

3.1.6 Logs

Detect where errors happen and track them down to their origin.

3.2 Portal nodes

Portal nodes are the data turning points in an HEPTAconnect instance. This is where the magic happens. Multiple nodes of a single portal can exist next to each other and connect to different types and instances of APIs.

3.2.1 How to create portal nodes

Portal nodes are instances of different portals. A portal is the set of code that is used to transform common HEPTAconnect structures from and into data source specific structures. When a portal node is created it can now receive configuration to access the data source and can be used for setting up data routes. At first the list of portals should be queried using the `heptaconnect:portal:list` command to see what kind of portal nodes can be created. The output can look like this:

```
Portals
_______

 ____________________________________________________________________ 
  class                                                               
 ____________________________________________________________________ 
  Heptacom\HeptaConnect\Integration\Filter\Portal                      
  Heptacom\HeptaConnect\Integration\Morph\Portal                       
  Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Portal            
  Heptacom\HeptaConnect\Portal\MayanEdms\Portal                        
  Heptacom\HeptaConnect\Portal\Shopware5\Portal                        
  Heptacom\HeptaConnect\Portal\Zammad\Portal                           
 ____________________________________________________________________ 
```

The command `heptaconnect:portal-node:add` is used to instantiate a node of a specific portal.

```
bin/console heptaconnect:portal-node:add 'Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Portal'
```

It also allows to create a portal node with a rememberable alias as well:

```
bin/console heptaconnect:portal-node:add `Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Portal` shopware_local
```

As output of the command you will receive the created primary key of the portal node which often looks like this `PortalNode:01234567890abcdef01234567890abcd`. The portal node key and an assigned alias can be used later on in other calls when you create data routes or inform yourself about the status of a portal node.

3.2.2 How to configure portal nodes

When the development team integrated additional configuration sources there are probably setup notes about it. This might invalidate some of the following samples.

Portal nodes often need API credentials or filenames to operate. To read the initial configuration the command `heptaconnect:portal-node:config:get` is used. Its output is json and can either be a single value or the complete configuration set:

```
bin/console heptaconnect:portal-node:config:get PortalNode:01234567890abcdef01234567890abcd --pretty
bin/console heptaconnect:portal-node:config:get shopware_local --pretty
```

```
{
    "dal_indexing_mode": "none"
}
```

or

```
bin/console heptaconnect:portal-node:config:get PortalNode:01234567890abcdef01234567890abcd dal_indexing_mode`
bin/console heptaconnect:portal-node:config:get shopware_local dal_indexing_mode`
```

```
none
```

This displays the information of the indexing mode of the underlying API client. A similar command can be used to change this configuration `heptaconnect:portal-node:config:set`:

```
bin/console heptaconnect:portal-node:config:set PortalNode:01234567890abcdef01234567890abcd dal_indexing_mode queue`
bin/console heptaconnect:portal-node:config:set shopware_local dal_indexing_mode queue`
```

As we are using JSON as serialization it is convenient for automated setups.

Further reading

After you set up multiple portal nodes you can use them in data routing and setup status tracking.

3.3 Routing

Routing is a setup once configuration step right after you created and configured a portal node. Creating routes is a crucial step to control the data flow. It defines which data is allowed to go in which direction.

3.3.1 Structure of a route

A route is the combination of a starting portal node, a targeted portal node and a data type. This is already an indicator that a data route is uni-directional.

3.3.2 Mapping setup

Some data needs to be mapped between two portal nodes but will not be automatically created by routed transit data. This can be due to reasons like a missing implementation as transit data or merging multiple mappings into a single mapping. Common entities affected by this are salutations, countries, currencies, payment methods and shipping methods. To administer manual mappings use the commands `heptaconnect:identity-redirect:add` and `heptaconnect:identity-redirect:list`.

3.3.3 How to use

To create a route we want to know what data types we can connect between which portal nodes. To get insights into the available portal nodes there is the command `heptaconnect:portal-node:list`. The output can look similar to this:

```
 ------------------ ------------------------------------------------------
  portal-node-key    portal-class
 ------------------ ------------------------------------------------------
  filter             Heptacom\HeptaConnect\Integration\Filter\Portal
  mayan              Heptacom\HeptaConnect\Portal\MayanEdms\Portal
  morph              Heptacom\HeptaConnect\Integration\Morph\Portal
  sw5                Heptacom\HeptaConnect\Portal\Shopware5\Portal
  sw6                Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Portal
  zammad             Heptacom\HeptaConnect\Portal\Zammad\Portal
 ------------------ ------------------------------------------------------
```

There is a similar command for the available data types. There is the command `heptaconnect:data-type:list` that lists all data types that are supported by the installed portals. An output of the command can look like this:

```
 ------------------------------------------------------
  Heptacom\HeptaConnect\Dataset\Ecommerce\Product\Product
  Heptacom\HeptaConnect\Dataset\Ecommerce\Media\Media
  Heptacom\HeptaConnect\Dataset\Ecommerce\Product\Category
  Heptacom\HeptaConnect\Dataset\Ecommerce\Customer\Customer
  Heptacom\HeptaConnect\Dataset\Ecommerce\Order\Order
  Heptacom\HeptaConnect\Dataset\Ecommerce\Currency\Currency
 ------------------------------------------------------
```

With all the information above we can create routes that can resemble a scenario like the following: * Send products, cms media, customers and orders from the old shop to the new shop * Send generated documents from the shop to the DMS * Send customers and their orders to the help desk

To setup the described scenario we create routes with the command `heptaconnect:router:add-route`. For the first mentioned instruction the command is used as the following:

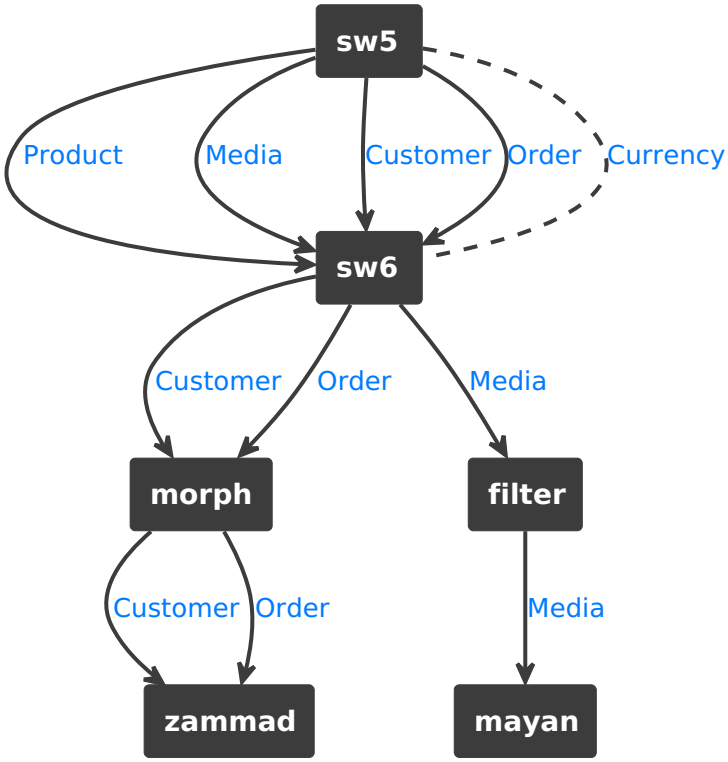
```
bin/console heptaconnect:router:add-route sw5 sw6 'Heptacom\HeptaConnect\Dataset\Ecommerce\Product\Product'
```

The complete scenario can be setup with just the following few lines:

```
bin/console heptaconnect:router:add-route sw5 sw6 'Heptacom\HeptaConnect\Dataset\Ecommerce\Product\Product'
bin/console heptaconnect:router:add-route sw5 sw6 'Heptacom\HeptaConnect\Dataset\Ecommerce\Media\Media'
bin/console heptaconnect:router:add-route sw5 sw6 'Heptacom\HeptaConnect\Dataset\Ecommerce\Customer\Customer'
bin/console heptaconnect:router:add-route sw5 sw6 'Heptacom\HeptaConnect\Dataset\Ecommerce\Order\Order'
bin/console heptaconnect:router:add-route sw6 filter 'Heptacom\HeptaConnect\Dataset\Ecommerce\Media\Media'
bin/console heptaconnect:router:add-route filter mayan 'Heptacom\HeptaConnect\Dataset\Ecommerce\Media\Media'
bin/console heptaconnect:router:add-route sw6 morph 'Heptacom\HeptaConnect\Dataset\Ecommerce\Customer\Customer'
bin/console heptaconnect:router:add-route sw6 morph 'Heptacom\HeptaConnect\Dataset\Ecommerce\Order\Order'
bin/console heptaconnect:router:add-route morph zammad 'Heptacom\HeptaConnect\Dataset\Ecommerce\Customer\Customer'
bin/console heptaconnect:router:add-route morph zammad 'Heptacom\HeptaConnect\Dataset\Ecommerce\Order\Order'
```

```
# Map EUR in SW5 to EUR in SW6
bin/console heptaconnect:identity-redirect:add 'Heptacom\HeptaConnect\Dataset\Ecommerce\Currency\Currency' sw5 1 sw6 b7d2554b0ce847cd82f3ac9bd1c0dfca
```

The routes after this will look like this.



3.4 Status reporting

Status reports are the tools to build your status page for your HEPTAconnect installation. They are developed by portal developers for capturing metrics.

3.4.1 Common metrics

HEPTAconnect suggests portal developers to supply status reports for the following topics:

- health: Health status reports are used to test configurations against the underlying portal data source to track availability and connectivity
- analysis: Analysis status reports are used for API usage rates or different self detected behaviour metrics
- info: A generic set of information that is helpful right after installation
- config: A set of information for configuration support like generated API endpoints

To have a detailed look into the thoughts of status reports have a read into the corresponding ADR.

3.4.2 How to use

The fastest way in setup is a health tracker using a crontab configuration. It will use a regular check on the portal and use crontab mailing feature to inform about an unhealthy portal. A call to the status report command `heptaconnect:portal-node:status` filtered through jq with the -e exit code option easily convert the health result into something processable in a shell.

```
bin/console heptaconnect:portal-node:status PortalNode:123 health | jq -e .health
```

3.5 HTTP APIs

HEPTAconnect itself and portals expose HTTP endpoints for various actions. To ensure correct hosting and exposure of these endpoints you can read everything you need in here.

3.5.1 Base URL

The bridges define the integration of HEPTAconnect in the surrounding application it got embedded into. To see which base URL is used by HEPTAconnect you can use the command `heptaconnect:config:base-url:get`. When it does not match the expectations you can use the command `heptaconnect:config:base-url:set` to change it.

3.5.2 Endpoint listing

Portals can resolve the absolute URLs for their registered endpoints. There is the command `heptaconnect:http-handler:list-handlers` to display the registered endpoints. The output looks like this:

```
┌─────────────┬──────────────┬──────────────────────────────────────────────────────────────────┐
│ portal-node │ path         │ url                                                              │
├─────────────┼──────────────┼──────────────────────────────────────────────────────────────────┤
│ bottle      │ hello-world  │ http://example.com/api/heptaconnect/flow/bottle/http-handler/hello-world │
└─────────────┴──────────────┴──────────────────────────────────────────────────────────────────┘
```

3.5.3 Enabled and disabling handlers

By default, every HTTP handler is enabled. In a scenario where the web activity needs to be disabled, you can use `heptaconnect:http-handler:set-configuration bottle hello-world enabled false` to set the handler's enabled configuration on path `hello-world` for portal node `bottle` to deactivate it. In a similar way you can look up the enabled configuration: `heptaconnect:http-handler:get-configuration bottle hello-world enabled`.

3.5.4 Debugging

For investigation purposes you can dump the HTTP requests and responses of the HTTP handlers. To do so you can use the configuration key `dump-sample-rate` to set the sampling rate. The expected value is an integer between 0 and 100. By default, the sampling rate is set to 0, which means no requests are dumped. To dump all requests you can set the sampling rate to 100. E.g. to set the configuration to 3/4 of the requests get recorded you can use the command `heptaconnect:http-handler:set-configuration bottle hello-world dump-sample-rate 75`. The dumped requests are stored on filesystem next to your log files.

3.6 Filesystem

Next to data moved between portal nodes, there are also files on disk, that count as transaction data. Ensure to back up and store this data to be accessible from all app servers.

3.6.1 Locations

When the development team integrated a non-standard filesystem storage there are probably setup notes about it. Otherwise, the bridge storage fallback is used, which is always a subdirectory within the project directory. See details about the bridges below.

3.6.2 Sample configurations

Shopware 6 Bridge

The Shopware 6 bridge places files in `<instance-dir>/files/plugins/heptaconnect_bridge_shopware_platform/` with a subdirectory for each portal node.

```
<system-root>
└── …
    └── <instance-dir>
        └── files
            └── plugins
                └── heptaconnect_bridge_shopware_platform
                    ├── <portal-node-1>
                    ├── <portal-node-2>
                    └── <portal-node-3>
```

If no changes are done in the integration, you can still move the data outside of this directory. You can replace a directory with a symbolic link to a directory, that suits better for storage of transaction data. When used with Docker, the directory `<instance-dir>/files/plugins/heptaconnect_bridge_shopware_platform/` is best a Docker volume. When a network storage shall be used, operating system tools like FUSE can be used to mount network storages like FTP and SMB.

3.7 Logs

Log messages are the most detailed way to get into the actions that happen in the HEPTAconnect instance at close to real time. Only debugging into it provides more details. Watch the following sources for changes and get informed about the most detail info you can get.

3.7.1 Locations

When the development team integrated a non-standard logging facility there are probably setup notes about it. Otherwise, the bridge logging fallback takes action, logs into files and the following paragraphs apply.

Files

File logs contain the most different message types and should be the first choice of investigation. The log file locations may vary as they depend on the integration your instance uses. Common locations to check:

```
<system-root>
├── …
│   └── <instance-dir>
│       ├── var
│       │   └── log
│       │       └── heptacom_heptaconnect_*.log
│       └── storage
│           └── logs
│               └── heptacom_heptaconnect_*.log
└── var
    └── log
        └── application
            └── heptacom_heptaconnect_*.log
```

Database

HEPTAconnect provides an entity-centered database table to store entity related exceptions. You can find it in your instance database by the name `heptaconnect_mapping_error_message`. It is used to store error messages that can be connected to certain items.

3.7.2 Contents

Files

Log files contain timestamps, log level, component names (e.g. EmitterStackBuilder, ExplorationActor), messages and unique codes. Depending on the message you have additional context like primary keys. When a log message is issued from a portal the message is prefix with the portal node key (aliases are supported). Unique log message are part of the error origin finding process. You can read more in a news entry and this ADR about it.

Database

The database table contains timestamps, portal node keys, mapping node keys, message, exception type, exception stacktrace for a complete exception stack. As the database table only contains exceptions there is no need of a log level.

4. Playground

4.1 Getting Started

There are two ways to quickly get started with HEPTAconnect. If you would like a setup with pre-configured portals and a working runtime, you should check out the playground. If you want to develop your own packages for HEPTAconnect, you should install the SDK first.

4.1.1 Playground

The playground provides environments (e.g. Shopware 6) for you to try out and fiddle with HEPTAconnect for different occasions. It is the easiest way to get your hands on a simple environment and test some portals you found online. Read more in the playground articles.

4.1.2 SDK

The SDK is similar to the playground as it also comes with a minimal Shopware 6 system as its runtime. But here it is even more trimmed down to focus on the development of HEPTAconnect packages. It also comes with some utility commands designed to improve the developer experience.

You can install the SDK in a directory of your choice. Upon installation it will create a symlink to the `heptaconnect-sdk` binary file in a directory of your `PATH` environment variable, so you can access it from anywhere in your terminal using `heptaconnect-sdk`. Start the installation like so.

```
composer create-project heptacom/heptaconnect-sdk:dev-master
```

If you are using Composer 2.x, the installation wizard should start immediately. If you are using Composer 1.x, it will ask you to execute the wizard manually using this command.

```
heptaconnect-sdk sdk:install
```

The installation wizard will ask you for database credentials of your MySQL server. Afterwards it will setup the database.

Success

Great job! You installed the SDK. Let's make a package now.

Building your own portal

Initialize a new package using the SDK. This will start a short questionnaire to gather information about your package.

```
heptaconnect-sdk sdk:package:init <target-dir>
```

The SDK will ask you a few questions about your new package and prepare your project accordingly. You can choose between creating a dataset, a portal and a storage. In this example we create a portal package.

```
 Choose the type of package you want to build:
  [0] heptaconnect-dataset
  [1] heptaconnect-portal
  [2] heptaconnect-storage
 > heptaconnect-portal

 Give your package a name. [<my-username>/<target-dir>]:
 > example-vendor/my-portal-package

 Specify a PSR-4 compliant namespace. [ExampleVendor\MyPortalPackage]:
 >

Loading composer repositories with package information
```

```
Updating dependencies
...
```

Your `<target-dir>` should now have the following contents:

```
<target-dir>
├── composer.json
├── composer.lock
├── src
│   └── Portal.php
└── vendor
    └── autoload.php
    └── ...
```

Success

You successfully created your first portal!

Adding your package to the SDK runtime

You now have a shiny new package and can start developing. When you want to execute your code in a HEPTAconnect runtime to see if it works, you first have to add the package to your SDK installation. This will make the SDK aware of your package and you can then use commands in the `heptaconnect` namespace with your package. Run this command to add your package.

```
heptaconnect-sdk sdk:package:add <target-dir>
```

You should see Composer updating the SDK with your package as a dependency. The SDK will now load your package with every command you execute. You can verify this (for portals) by running this command.

```
heptaconnect-sdk heptaconnect:portal:list
```

Success

Your portal is now available in the SDK!

4.2 Playground

The playground is a quick entry for getting your hands onto HEPTAconnect in action.

When you want to ...

- use HEPTAconnect for the first time, ...
- try out a portal you found online, ...
- debug into some default HEPTAconnect flows, ...
- contribute to the HEPTAconnect packages and need a basic environment, ...

this is where you should be able to try it without building an integrated system yourself. The playground comes with a fully functional example portal and data set: the bottle portal. This portal uses static data as its data source. While this will probably never be an actual use-case, it makes things very predictable and easy to comprehend.

4.2.1 What can I expect?

You can find different running project based on skeletons that are populated with some basic configuration. Choose your matching environment and try it on your local machine. To get started see our first time with playground articles. When you are already familiar with HEPTAconnect you can use it to contribute to the packages.

4.2.2 What will I miss?

There are ...

- no optimizations for message broker
- no additional caching techniques
- no cluster configurations
- public "secrets" that **must not be used in production** environments

Having an easy entry with predefined configurations has its drawbacks for other cases. You should not expect these project skeletons to be similar to a production ready environment. They are configured to be run on a single machine with only a few additional dependencies to lower the initial hurdle.

4.3 First time

4.3.1 First time with HEPTAconnect

Clone your environment

To set up your first HEPTAconnect environment fetch a fresh copy of the HEPTAconnect playground:

```
git clone https://github.com/HEPTACOM/heptaconnect-playground
cd heptaconnect-playground
```

Now you have the all the sample code you need for different projects to try out. Make sure you have the tool `make` ready on your system to run later support scripts.

Choose an environment

We have environments for different frameworks ready to try out. See the related "first time with …" pages for the different environments below:

- Shopware 6

4.3.2 First time HEPTAconnect with Shopware 6

Features

When you choose to use a Shopware 6 as environment you have:

- Shopware 6 integration, commandline and administration
- Preconfigured Shopware 6 portal node (alias `shopware`)
- Preconfigured Bottle portal node (alias `bottle`)
- Preconfigured administration credentials (user `admin` password `shopware`)

Check system

To have your first Shopware 6 setup in your playground copy you need to fulfil the following requirements:

- PHP 7.4+
- Composer 2+
- At least 256MB memory limit for PHP processes
- Shopware 6 suitable database (e.g. MySQL 5.7, see their requirements)

When you have an environment that supports hosting of a PHP application you can point it to the directory as root `shopware-platform/public`. The default settings for the database credentials are:

- **user** – `root`
- **password** – `root`
- **host** – `localhost`
- **port** – `3306`
- **database** – `heptaconnect_shopware_platform`

When this differs from your available database you can change an environment variable to override the default connection.

```
DATABASE_URL=mysql://root:root@localhost:3306/heptaconnect_shopware_platform
```

Either change the entry in the `shopware-platform/.env` or `export` that statement for a single shell session.

Set up the playground

Now when the prerequisites are met you can run:

```
make shopware-platform
```

and watch your shop set up automatically. Later on you have a Shopware 6 CLI available at `shopware-platform/bin/shopware`. When you configured hosting you can use the shopware administration under your URL `/admin`.

Keep in mind that this Shopware does not run any queue workers by default. In case you want to transfer data in that environment you have to run:

```
shopware-platform/bin/shopware messenger:consume
```

4.4 Commandline

Your commandline entrypoint depends on the environment you chose before. See below the executables to try out:

- Shopware 6 `shopware-platform/bin/shopware`

4.4.1 First commands

The best commands to discover your HEPTAconnect integrated application are:

- `heptaconnect:portal:list` to see your available portals to connect
- `heptaconnect:portal-node:list` to see your available portal nodes to draw routes between them
- `heptaconnect:portal-node:config:get --pretty bottle` to see a portal node configuration
- `heptaconnect:portal-node:alias:overview` to see that the portal node names are just aliases to reduce complexity for configuration and usages
- `heptaconnect:router:list-routes` to see what routes have already been configured

The next step for you should be taking a look at adding further portals to your environment.

4.5 Add more portals

To discover what other portals you can try out you can query different package and code distributing platforms for the tag `heptaconnect-portal`.

Add portals from GitHub

Let's take a look at the results at GitHub. You can probably take any repository and clone it into the `/repos/` folder that is at the root directory of the playground.

```
git clone https://github.com/example/repo repos/example-repo
composer require -d ./shopware-platform/ example/repo
```

4.5.1 See the new portals

The best commands to discover the changes within your HEPTAconnect integrated application are:

- `heptaconnect:portal:list` to see your new portal
- `heptaconnect:portal-node:add 'Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Portal' nice_portal` to create a new portal node with the alias `nice_portal`
- `heptaconnect:portal-node:config:set nice_portal api_key` to set a portal node configuration e.g. `api_key`

4.6 Contribute to HEPTAconnect packages

4.6.1 Why use the playground for contribution?

The playground uses vcs clones to have a central place for packages: the `/repos` directory. This has multiple advantages: 1. You can place any composer package in that folder, and it is automatically detected 2. You can place also non composer packages in that folder 3. You can have it from any vcs (e.g. git hosted on GitHub)

Looking at the third point we can now easily see that we have a versioned copy of the package. With a copy at almost root level of the project you can easily apply changes without diving into the depth of a composer vendor folder and can check it working in any playground environment. Now with the copy being a versioned copy you can commit your changes and prepare a pull request for us to look at.

5. Contributor

5.1 How to be a HEPTAconnect contributor

This is all about setting up a development environment to work on HEPTAconnect projects.

5.1.1 Technical requirements

- PHP 7.4 or above
- Composer 1.8 or above

5.1.2 Used tools, technologies and techniques

Adding static type hints with psalm and phpstan helps us to provide safe and more comprehensive code. Using these tools adds generics and class string functionalities that helps understanding code without an execution context.

Developing on the maintainer side is mainly done in JetBrains PHPStorm but is not limited to it. You are free to use any IDE or text editor although the developer experience is improved on the usage of PHPStorm.

Tests and their coverages are ensured using pest (phpunit) unit tests and code mutators to improve testing quality.

HEPTAconnect aims to be working at bleeding edge technology and supporting the latest stable versions. To achieve this the composer requirements are only setup with a lower bound to allow the easy usage of latest releases.

By the growth of the HEPTAconnect community and connected APIs we look out for easy ways to support developers to allow their projects run steadily and non-breaking while the core runtimes get improved and extended by time. One of these ways are fallback implementations. A fallback class always implements an interface completely in a way that makes the code at least be valid in a php code runtime. When you use these fallback classes we can support your extension without breaking it.

Message brokers and asynchronous messaging allows HEPTAconnect to be just a little impact on the performance of the main application that provides the bridge. In addition asynchronous messaging allows for a scalable increase of reactivity and flexibility.

5.1.3 Licensing

Thank you for considering contribution! Be sure to sign the CLA after creating the pull request. 

5.1.4 Steps to contribute

1. Fork the repository
2. `git clone yourname/heptaconnect-framework`
3. Make your changes to a branch
4. `make coverage`
5. Create your Pull-Request

5.2 Writing changelogs

5.2.1 Motivation

Working in the past with different frameworks and libraries we experienced it was to a certain degree easy to integrate into projects. In most cases it was much more difficult to keep the dependencies updated properly compared to introducing them. This is something we want to be different. In our fast-paced world of changes we need to also look out for these qualities.

5.2.2 Structure

The changelog file in the root of the package has a preface for the rules inside the changelog file. We follow the format of the keep a changelog project with semantic versioning which formats the file in a standardized layout.

For every version we will have changes listed. It is allowed to release a package without a change to properly release a group of interdependent packages so an empty section is fine. There is always a version entry called `Unreleased` to collect new entries up to the release of the upcoming version.

The logs of each version are grouped into their classification of additions, changes, deprecations, removals, fixes and security fixes. To ensure a better understanding of the change the logs are focussed on features of the package written in present tense. Each log has to contain a technical reference to look up usages of the feature.

5.2.3 Examples

Depending on the change we can form scenarios into changelogs. When you are unsure what to write you can find some sample texts below:

Extract code of private API into code of public API

This introduces a feature, so we should write something like:

```
### Added

- Introduce fiddling of stuff into new class `\Heptacom\HeptaConnect\StuffFiddler`
```

As seen in version 0.8.0 of heptaconnect-core:

Added

- Extract path building from `\Heptacom\HeptaConnect\Core\Storage\Normalizer\StreamNormalizer` and `\Heptacom\HeptaConnect\Core\Storage\Normalizer\StreamDenormalizer` into new service `\Heptacom\HeptaConnect\Core\Storage\Contract\StreamPathContract`

Add parameter to method of public API

This introduces new behaviours, so we also add a new feature:

```
### Added

- Add optional parameter `$foobar` in `\Heptacom\HeptaConnect\StuffFiddler::fiddle`
```

As seen in version 0.7.0 of heptaconnect-core:

Added

- Add parameter for `\Psr\Log\LoggerInterface` dependency in `\Heptacom\HeptaConnect\Core\Portal\PortalStorage::__construct` and `\Heptacom\HeptaConnect\Core\Portal\PortalStorageFactory::__construct`

Specialize component

When you have a generic component, that is not well optimized for certain cases, can be replaced with a new more optimized component:

```
### Added

- Add class `\Heptacom\HeptaConnect\GizmoStuffFiddler` that can fiddle better with stuff of gizmos in terms of memory handling

### Removed

- Remove `\Heptacom\HeptaConnect\StuffFiddler::fiddle`. Use `\Heptacom\HeptaConnect\GizmoStuffFiddler::fiddleGizmos` instead
```

As seen in version 0.8.0 of heptaconnect-storage-base:

Added

- With storage restructure explained in this ADR we add `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Get\RouteGetActionInterface` for reading metadata of routes by the given `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Get\RouteGetCriteria` to return a `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Get\RouteGetResult`

Removed

- With storage restructure explained in this ADR we remove implementation `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\RouteRepositoryContract::read` in favour of `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Get\RouteGetActionInterface::get` that allows for optimizations in the storage implementation

Upgrade composer dependency

Pre-checking composer upgrades is important to reveal changes of further application fine-tuning, so we mention them as well:

```
### Changed

- Upgrade composer dependency `psr/log: ^1.0` to support future versions `psr/log: ^2.0`
```

Fix a bug

The difficult part here is the differentiation between a security bugfix and an unexpected behaviour:

```
### Fixed

- When passing foobaz into `\Heptacom\HeptaConnect\StuffFiddler::fiddle` did not pay respect to a GizmoStuff situation
```

As seen in version 0.7.0 of heptaconnect-portal-base:

Fixed

- `\Heptacom\HeptaConnect\Portal\Base\Support\Contract\DeepObjectIteratorContract::iterate` drops usage of `\spl_object_hash` to not break on garbage collection

Add unique log code for lookups

When anything is logged or an exception is thrown a package-unique code should be generated. Using a UNIX timestamp is handy as it is an integer and plays nicely with `\Throwable::getCode`. These have to be documented in the changelogs as well to raise awareness and be the first contact point for persons in need of an explanation.

```
### Added

- Add log exception code `123456789` to `\Heptacom\HeptaConnect\StuffFiddler::fiddle` when fiddling with stuff that is not allowed to access gizmos
```

As seen in version 0.8.0 of heptaconnect-core:

Added

- Add log exception code `1636503503` to `\Heptacom\HeptaConnect\Core\Job\Handler\ReceptionHandler::triggerReception` when job has no related route

Rename classes or move classes between namespaces

When a code refactoring needs moving a class a plain `rename` or `move` hint should to be added to the changelog. Additional explanation is optional but suggested as most refactoring have a good reasoning regarding functionality.

```
### Changed

- Rename `\Heptacom\HeptaConnect\StuffFiddler` to `\Heptacom\HeptaConnect\StuffFiddlerHandler`
```

As seen in version 0.5.0 of heptaconnect-dataset-base:

Changed

- Rename `\Heptacom\HeptaConnect\Dataset\Base\Translatable\GenericTranslatable` to `\Heptacom\HeptaConnect\Dataset\Base\Translatable\AbstractTranslatable`

5.3 Building flow components

5.3.1 Preparation

This guide assumes you have been using HEPTAconnect as a portal developer or integrator before. A possible reason you are reading this is, that you are in a similar situation like the following and think of solving the issue within the HEPTAconnect framework.

Situation

An existing flow component like the receiver gets multiple receiver decorator implementations to stop following receivers on the stack to take action. You see a pattern that all these changes in the reception stack are meant to prevent writing to an API and do a lookup instead.

The next steps are:

- to describe the pattern and locate it in its current situation
- to isolate its features
- to extract its exclusive features compared to existing flow components
- to question its introduction into the next version

Pattern detection

The above detectable pattern is a reception decoration to lookup existing entries in a reception targeted portal node. It takes place within a data flow and introduces new behaviour that is not communicated by the receiver service contract. The new behaviour prevents writing and makes a data lookup instead. It can replace a reception and therefore be a new flow component.

Feature isolation and exclusiveness

The potential new flow component can only replace a receiver. This can also be turned into supporting readonly APIs without using a flow component that communicates writing in its description. A portal developer can use code separation to separate features and different usage. An administrator has a more flexible usage of the portal without additional configuration provided by the portal developer.

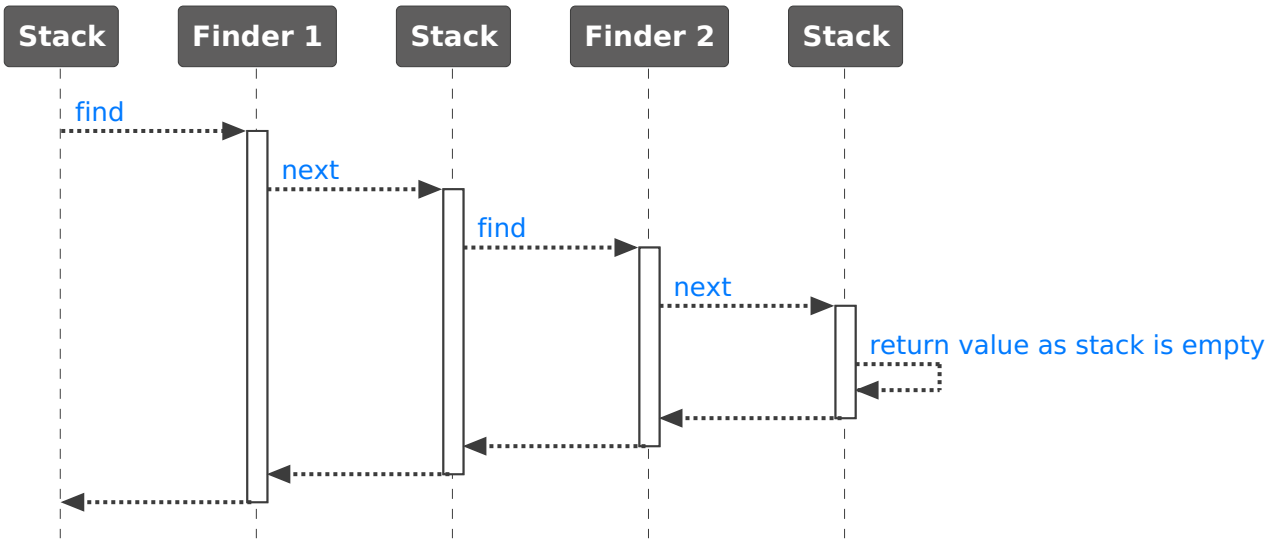
Introducing the new flow component

Any existing setup should not behave differently, therefore using the new flow component needs to be optional. A name has to be chosen carefully to match its features and usage. In this case the name `Finder` has been chosen as flows are entity focused, and we try to find an entity in the targeted portal node. In addition to the component name we will name the namespace `Find` and the methods will use the verb `find` as well. As this flow component is intended to take part in a data flow we can use route capabilities to configure its behaviour. New documentation on its usage for portal developer and administrator needs to be written. The new flow component should follow patterns in implementation that other flow components have in common.

5.3.2 Common implementation

Stack

Flow component have certain properties, that allow to group them by. For example, receivers are grouped by their supported dataset entity and their portals and supporting portal extensions they are provided with. Based upon that, a portal and a portal extension can provide an implementation for a flow component that belong together and can influence each other. Their code origin influences their order in the stack. The order can be used to build an ordered stack out of it. This stack is used to pass a certain payload into it, pass through every layer in their respective order and allow each layer to modify a possible result that is returned at the end of the stack iteration. In the case of an emitter stack the payload is a set of identities and its return value are resolved entities. The flow components in a stack can intercept stack iteration to allow full influence of behaviour. The `next` method of the stack is the first entrypoint of a stack to start the layer iteration.



Stack building

The stack order and its overall contents can differ in each usage situation. To ensure reproducible order the building process is abstracted into its own service. The stack builder is aware of a source instance, that can be provided from a portal or a portal extension. This instance is the first in the order and is used in general as last entry in the stack. Every other instance on the stack is called a decorator and provided by portal extensions.

Call decoration

When the `next` method on the stack is called it has to call the `find` method of the first instance on the stack. The `find` method of the finder instance itself gets the current stack as argument and can now take over the control of the following `$stack->next()` call. This way a flow component can change the inbound payload, the result and break the execution.

Context

In addition to the omnipresent stack there is also the context. Each flow component type has its own context. The different contexts barely have anything in common and are specialized to provide functionality that only makes sense in usage of this specific flow component and cannot be provided by a service in the portal node container. For example, the context in an emission has the method `isDirectEmission` to allow knowledge about its usage like a direct emission. The `FindContext` could probably get information whether it is preceding a reception.

Flow component contract class

A contract class for a flow component always has at least three methods:

- supports
- find
- run

The `supports` method represents every getter method that returns data to group a flow component instance by. It is neither aware of the context nor the stack as it will be used to prepare both of these. Every implementation of a flow component needs to provide the information about its supported topic or dataset entity.

The `find` method has already been introduced in the stack explanation. It is named to the verb of the component and therefore varies between the different flow components. It needs to be pre-implemented to chain as described so a portal developer does not need to implement it. The portal developer still needs to be able to override its implementation as this method is controlling the processing flow through the stack.

The `run` method has to have a signature that allows for a possible implementation with the least needed instructions to take effect. This is the first entrypoint a portal developer will look for and has to enable the developer to see effects quickly. The first entrypoint does not need to be the most efficient way but the most efficient way needs to possible and should be less complex than implementing `next` yourself. A suitable example is the `\Heptacom\HeptaConnect\Portal\Base\Reception\Contract\ReceiverContract` it has a `run` and `batch` method, that allows to implement both scenarios but works out of the box independently whether `run` or `batch` is implemented.

Short notation

The short notation provides a different way to implement the flow component contract class. For every overridable method of the `FinderContract` except `find` the portal developer shall be able to provide a closure as implementation. The signature of these closures can have a custom rule set and must not be limited to the respective signature of the method in the contract to allow dependency injection by the portal node container.

All closures are collected in a token. The token class `FinderToken` has no features beside storing closures.

To provide a fluent interface for portal developers to configure the token a builder class is needed. All methods of the flow component specific builder need to be named the same as in the flow component contract class so its usage is the very similar to implementing the contract class. Each method will store the parameters in the wrapped token instance and return itself to ensure a fluent usage.

To execute the closures in the token we need a generic implementation of the contract. It will take the token in the constructor and execute each callback in the respective duplicated methods. At this place you have to analyze the parameters of the token's closure and lookup any services from the service container. This also allows custom rules to take effect. As an example we can look into the `EmitterContract`: when you forward the `run` method to the closure you can also resolve a string parameter called `externalId` to be the previous parameter `$externalId` from the `run` method.

To access the new builder component the `\Heptacom\HeptaConnect\Portal\Base\Builder\FlowComponent` facade needs to provide a factory method named like the new flow component, so it can be used in short notation files. It will also need to factorize the flow components.

Code origin finder

Reasonable log messages are crucial. Therefore, whenever HEPTAconnect is aware of a flow component being part of the log message's context, the file the flow component is written in is logged as well. This feature is not possible without a `FinderCodeOriginFinder`. It can differentiate between a token based implementation of the `FinderContract` and an object-oriented implementation. The token based implementation needs to evaluate the source of the closures in the token instead of the class implementations' source file. The new code origin finder class can now be used along with the others in the `\Heptacom\HeptaConnect\Core\Component\Logger\FlowComponentCodeOriginFinderLogger` to improve log messages.

Portal node container

Building the portal node container has a big impact on the usage of the newly created flow component. It loads the short notation files and detects all implementations of the new contract class to propagate their existences. This has to be implemented by scanning all implementations and pass the service references to the `\Heptacom\HeptaConnect\Core\Portal\FlowComponentRegistry`.

Flow component registry

The flow component registry is the central place of a portal node container to supply all flow components for a portal node. Therefore, a getter method for instances of the new flow component must be added to the registry. This new getter will rely on the new parameter in the constructor and the newly found services in the portal node container.

Factory

As the context and the stack building are specialized for situations, factories are essential tooling. These factories are not accessible by portals. The `FinderStackBuilderFactory` will load a portal node container and request the finder flow component instances from the flow component registry. Now the stack builder has everything to work with later on.

Actor

Most of the previous parts are taking place in the portal base package. Everything is ready for portals and extensions to use the new flow component. The next big step is to teach the core package what to do with the new flow component. The service that will actually work with the new flow components is an actor, the `FindActor`. Its implementation precisely knows how to process a stack properly. The `performFind` method of this service looks similar to a contract class `find` as it does not create the stack and context it will work with later. An actor often validates incoming data (e.g. does not forward to the stack at all when empty), triggers different actions like follow-up flows.

Service

The main entrypoint for every execution of the new flow component is its own service. `FindService` will take as few arguments as needed to build a stack, create a context and execute the `FindActor`.

Jobs

HEPTAconnect can outsource flow component processing as jobs in different processes (commonly on different machines). To support this we need to introduce a job for the new flow component. Job classes need to be based upon `\Heptacom\HeptaConnect\Core\Job\Contract\JobContract`. Instances of a job class contain all infos that is needed to process a job. In the best scenarios a job is only aware of an identity. In our scenario we want to behave similar to the reception and therefore also need the entity to lookup later. The entity will be part of the payload of the `Find` job. Instances of these job type instances can now be dispatched using the `\Heptacom\HeptaConnect\Core\Job\Contract\JobDispatcherContract`. Its implementation ensures forwarding the job and its payload to be used in other process like a message queue or a child PHP process.

Job handling

The job has been dispatched to be handled separately from the current PHP process. When the job is ready to execute, it needs to be handled. The `FindJobHandler` will track the job processing state, read the jobs' payload and pass the job payload to the `FindService`. On finishing the `Find` job, a follow-up `Reception` job should be generated, when the route capability allows it.

Route capability usage

Now we are about to finish the initial task. The storages need to know about the new route capability. We designed it to be optional and name it `find`. Right before `Reception` jobs will be dispatched, we decide to ask the storage for the route capability and dispatch a `Find` job instead. There is no functionality lost as the `Find` job will be able to generate a follow-up `Reception` job.

Admin UI

Often it is useful to have certain utilities for the administrator. In this scenario you only have to make sure the new route capability is available in the storage implementations, but the admin UI is already able to display the route capability. It is probably that your new flow component will be part of the admin UI.

5.3.3 Summary

Portal base

We need to provide contracts for the portal developer to use. You are most likely having a file structure of new files like this:

```
<portal-base-source>
├── Builder
│   ├── Builder
│   │   └── FinderBuilder.php
│   ├── Component
│   │   └── Finder.php
│   └── Token
│       └── FinderToken.php
└── Find
    ├── Contract
    │   ├── FindContextInterface.php
    │   ├── FinderCodeOriginFinderInterface.php
    │   ├── FinderContract.php
    │   └── FinderStackInterface.php
```

```
    ├── FinderCollection.php
    └── FinderStack.php
```

Core

The actual usage of the new flow component needs to be handled within the core. You are most likely having a file structure of new files like this:

```
<core-source>
├── Find
│   ├── Contract
│   │   ├── FindActorInterface.php
│   │   ├── FindContextFactoryInterface.php
│   │   ├── FindServiceInterface.php
│   │   ├── FinderStackBuilderFactoryInterface.php
│   │   └── FinderStackBuilderInterface.php
│   ├── FindActor.php
│   ├── FindContext.php
│   ├── FindContextFactory.php
│   ├── FindService.php
│   ├── FinderCodeOriginFinder.php
│   ├── FinderStackBuilder.php
│   └── FinderStackBuilderFactory.php
└── Job
    ├── Contract
    │   └── FindHandlerInterface.php
    ├── Handler
    │   └── FindHandler.php
    └── Type
        └── Find.php
```

5.4 Building storage actions

5.4.1 Preparation

This guide assumes you have been using HEPTAconnect as an integrator before. A possible reason you are reading this is, that you want to introduce a new flow component and thus need to change the management storage layout to provide new interactions.

Situation

A new flow component needs a new storage action to find out on which routes it is expected to action. Route capabilities are the way to configure flow components on routes, so we can look out for existing patterns. In case you are in a situation that is not already done in a similar way, have a look into the ADR capturing our thoughts on storage actions.

The next steps are:

- to define the interface by
- defining parameters
- defining result
- implement new tests
- implement in storage packages

Storage action

Storage actions interfaces are grouped in sub namespaces of `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\` depending on the storage component it affects.

```
<storage-base-source>
└── Contract
    └── Action
        ├── FileReference
        ├── Identity
        ├── IdentityError
        ├── Job
        ├── PortalExtension
        ├── PortalNode
        ├── PortalNodeAlias
        ├── PortalNodeConfiguration
        ├── PortalNodeStorage
        ├── Route
        ├── RouteCapability
        └── WebHttpHandlerConfiguration
```

Our situation requires to look into the route section as we want to query route data. There we have the following actions at release of version 0.9.0.0.

```
<storage-base-source>
└── Contract
    └── Action
        └── Route
            ├── ReceptionRouteListActionInterface.php
            ├── RouteCreateActionInterface.php
            ├── RouteDeleteActionInterface.php
            ├── RouteFindActionInterface.php
            ├── RouteGetActionInterface.php
            └── RouteOverviewActionInterface.php
```

`ReceptionRouteListActionInterface` is the closest one to our situation, so we can copy it, but we will look into the next steps when designing a new action.

STORAGE ACTION PARAMETER

This storage action looks for routes by a certain criteria: the entity type in question and the source portal node. To allow an extendable way to add new parameters without breaking the action interface, the parameters are grouped into a DTO class implementing the `\Heptacom\HeptaConnect\Dataset\Base\Contract\AttachmentAwareInterface`. This class is placed in a sub namespace

that all DTOs share. The namespace is similarly built compared to the namespace for the action interface. For this situation it will be `\Heptacom\HeptaConnect\Storage\Base\Action\Route\Listing\` and can look like this:

```
<?php

declare(strict_types=1);

namespace Heptacom\HeptaConnect\Storage\Base\Action\Route\Listing;

use Heptacom\HeptaConnect\Dataset\Base\AttachmentCollection;
use Heptacom\HeptaConnect\Dataset\Base\Contract\AttachmentAwareInterface;
use Heptacom\HeptaConnect\Dataset\Base\Contract\DatasetEntityContract;
use Heptacom\HeptaConnect\Dataset\Base\Support\AttachmentAwareTrait;
use Heptacom\HeptaConnect\Portal\Base\StorageKey\Contract\PortalNodeKeyInterface;

final class FindRouteListCriteria implements AttachmentAwareInterface
{
    use AttachmentAwareTrait;

    protected PortalNodeKeyInterface $sourcePortalNodeKey;

    /**
     * @var class-string<DatasetEntityContract>
     */
    protected string $entityType;

    /**
     * @param class-string<DatasetEntityContract> $entityType
     */
    public function __construct(PortalNodeKeyInterface $sourcePortalNodeKey, string $entityType)
    {
        $this->attachments = new AttachmentCollection();
        $this->sourcePortalNodeKey = $sourcePortalNodeKey;
        $this->entityType = $entityType;
    }

    public function getSourcePortalNodeKey(): PortalNodeKeyInterface
    {
        return $this->sourcePortalNodeKey;
    }

    public function setSourcePortalNodeKey(PortalNodeKeyInterface $sourcePortalNodeKey): void
    {
        $this->sourcePortalNodeKey = $sourcePortalNodeKey;
    }

    /**
     * @return class-string<DatasetEntityContract>
     */
    public function getEntityType(): string
    {
        return $this->entityType;
    }

    /**
     * @param class-string<DatasetEntityContract> $entityType
     */
    public function setEntityType(string $entityType): void
    {
        $this->entityType = $entityType;
    }
}
```

STORAGE RESULT

As the result is a list we can either go for a collection class or an iterable of returning each row. Using iterables through generators is good as it can reduce memory usage as not all rows have to be present in memory at once. Generators in implementations are only a bad choice, when the actions are meant to perform write operations in the storage, as these methods are only executed when an iteration happens and therefore might not happen. In our situation we only load data from the storage and can go for an iterator expectation. Consequently, we only need a single class: a DTO class to hold a single result.

```
<?php

declare(strict_types=1);

namespace Heptacom\HeptaConnect\Storage\Base\Action\Route\Listing;

use Heptacom\HeptaConnect\Dataset\Base\AttachmentCollection;
use Heptacom\HeptaConnect\Dataset\Base\Contract\AttachmentAwareInterface;
use Heptacom\HeptaConnect\Dataset\Base\Support\AttachmentAwareTrait;
use Heptacom\HeptaConnect\Storage\Base\Contract\RouteKeyInterface;

final class FindRouteListResult implements AttachmentAwareInterface
{
    use AttachmentAwareTrait;

    protected RouteKeyInterface $routeKey;
```

```
    public function __construct(RouteKeyInterface $routeKey)
    {
        $this->attachments = new AttachmentCollection();
        $this->routeKey = $routeKey;
    }

    public function getRouteKey(): RouteKeyInterface
    {
        return $this->routeKey;
    }
}
```

STORAGE ACTION INTERFACE

As we finished everything what goes in and goes out, we can now define the interface:

```
<?php

declare(strict_types=1);

namespace Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route;

use Heptacom\HeptaConnect\Storage\Base\Action\Route\Listing\FindRouteListCriteria;
use Heptacom\HeptaConnect\Storage\Base\Action\Route\Listing\FindRouteListResult;
use Heptacom\HeptaConnect\Storage\Base\Exception\UnsupportedStorageKeyException;

interface FindRouteListActionInterface
{
    /**
     * List all routes for a find scenario.
     *
     * @throws UnsupportedStorageKeyException
     *
     * @return iterable<FindRouteListResult>
     */
    public function list(FindRouteListCriteria $criteria): iterable;
}
```

It is important to write a comment onto the interface to define an expectation for writing the tests, using the action and implementing the action. Actions are provided by a factory implementing the `\Heptacom\HeptaConnect\Storage\Base\Bridge\Contract\StorageFacadeInterface`. When introducing this new storage action interface, a new method has to be added to the storage facade interface to get an instance of the storage action implementation. With a modified interface, every implementation and related test needs to be adjusted as well.

Storage test suite

Tests for the storage are defined in the `heptacom/heptaconnect-test-suite-storage` of the framework. This set of tests can be used by all storage implementations to test against. A test in the test suite is an abstract class that expects to be run by phpunit. To provide the implementation to test, an abstract method needs to provide an instance of `\Heptacom\HeptaConnect\Storage\Base\Bridge\Contract\StorageFacadeInterface`. In general, you will find a lifecycle test in these tests. A lifecycle test is like an e2e (end to end) test, but for data. So we create data, query data, modify data, query data, delete data and query data again. It can look like this:

```
<?php

declare(strict_types=1);

namespace Heptacom\HeptaConnect\TestSuite\Storage\Action;

use Heptacom\HeptaConnect\Portal\Base\StorageKey\PortalNodeKeyCollection;
use Heptacom\HeptaConnect\Storage\Base\Action\PortalNode\Create\PortalNodeCreatePayload;
use Heptacom\HeptaConnect\Storage\Base\Action\PortalNode\Create\PortalNodeCreatePayloads;
use Heptacom\HeptaConnect\Storage\Base\Action\PortalNode\Create\PortalNodeCreateResult;
use Heptacom\HeptaConnect\Storage\Base\Action\PortalNode\Delete\PortalNodeDeleteCriteria;
use Heptacom\HeptaConnect\Storage\Base\Action\Route\Create\RouteCreatePayload;
use Heptacom\HeptaConnect\Storage\Base\Action\Route\Create\RouteCreatePayloads;
use Heptacom\HeptaConnect\Storage\Base\Action\Route\Delete\RouteDeleteCriteria;
use Heptacom\HeptaConnect\Storage\Base\Action\Route\Listing\FindRouteListCriteria;
use Heptacom\HeptaConnect\Storage\Base\Bridge\Contract\StorageFacadeInterface;
use Heptacom\HeptaConnect\Storage\Base\Enum\RouteCapability;
use Heptacom\HeptaConnect\Storage\Base\Exception\NotFoundException;
use Heptacom\HeptaConnect\Storage\Base\RouteKeyCollection;
use Heptacom\HeptaConnect\TestSuite\Storage\Fixture\Dataset\EntityA;
use Heptacom\HeptaConnect\TestSuite\Storage\Fixture\Portal\PortalA\PortalA;
use Heptacom\HeptaConnect\TestSuite\Storage\Fixture\Portal\PortalB\PortalB;
use Heptacom\HeptaConnect\TestSuite\Storage\TestCase;

/**
 * Test pre-implementation to test find route related storage actions. Some other storage actions e.g. PortalNodeCreate
 * are needed to set up test scenarios.
```

```
 */
abstract class FindRouteTestContract extends TestCase
{
    /**
     * Validates a complete find route "lifecycle" can be managed with the storage. It covers creation, usage,
     * configuration and deletion of routes.
     */
    public function testLifecycle(): void
    {
        $facade = $this->createStorageFacade();
        $portalNodeCreateAction = $facade->getPortalNodeCreateAction();
        $portalNodeDeleteAction = $facade->getPortalNodeDeleteAction();
        $routeCreateAction = $facade->getRouteCreateAction();
        $routeFindRouteListAction = $facade->getFindRouteListAction();
        $routeDeleteAction = $facade->getRouteDeleteAction();

        $portalNodeCreateResult = $portalNodeCreateAction->create(new PortalNodeCreatePayloads([
            new PortalNodeCreatePayload(PortalA::class),
            new PortalNodeCreatePayload(PortalB::class),
        ]));
        $firstResult = $portalNodeCreateResult->first();
        $lastResult = $portalNodeCreateResult->last();

        static::assertInstanceOf(PortalNodeCreateResult::class, $firstResult);
        static::assertInstanceOf(PortalNodeCreateResult::class, $lastResult);
        static::assertNotSame($firstResult, $lastResult);

        $portalA = $firstResult->getPortalNodeKey();
        $portalB = $lastResult->getPortalNodeKey();

        $createPayloads = new RouteCreatePayloads([
            new RouteCreatePayload($portalB, $portalA, EntityA::class, [RouteCapability::FIND]),
            new RouteCreatePayload($portalA, $portalB, EntityA::class, [RouteCapability::FIND]),
        ]);

        $createResults = $routeCreateAction->create($createPayloads);
        static::assertCount($createPayloads->count(), $createResults);

        $findListResult = \iterable_to_array($routeFindRouteListAction->find(new FindRouteListCriteria($portalA, EntityA::class)));

        static::assertCount(1, $findListResult);

        $routeDeleteAction->delete(new RouteDeleteCriteria($routeKeys));

        $findListResult = \iterable_to_array($routeFindRouteListAction->find(new FindRouteListCriteria($portalA, EntityA::class)));

        static::assertCount(0, $findListResult);

        try {
            $this->routeDeleteAction->delete(new RouteDeleteCriteria(new RouteKeyCollection([$findListResult[0]->getRouteKey()])));
            static::fail('This should have been throwing a not found exception');
        } catch (NotFoundException $exception) {
        }

        $portalNodeDeleteAction->delete(new PortalNodeDeleteCriteria(new PortalNodeKeyCollection([$portalA, $portalB])));
    }

    /**
     * Provides the storage implementation to test against.
     */
    abstract protected function createStorageFacade(): StorageFacadeInterface;
}
```

Storage implementation

Without getting into too many details we have for tooling for performant SQL queries in our `doctrine/dbal` based storage, here some points we look out for when implementing these actions.

1. To track down issues with SQL queries, we add publicly known unique query identifiers as a comment to recognise them quickly in logs and profilers
2. Every write operation is wrapped in a transactional operation to ensure batch writes are either done completely or rolled back
3. Every select statement is ensured to be paginated to keep package sizes in a certain level
4. Every paginated query is ensured to have a proper order by statement
5. Every select needs to only use indices for queries
6. Implement every abstract test provided by the `heptacom/heptaconnect-test-suite-storage` package

5.5 Contributor License Agreement

This Contributor License Agreement ("CLA") documents the rights granted by Contributors to HEPTACOM GmbH. This Agreement will govern all Contributions made by the Contributor.

This Agreement is between Heptacom GmbH, Am Tabakquartier 62, 28197 Bremen [Germany] ("HEPTACOM"), and the person [or entity] making a Contribution to this Software ("Contributor" and collectively with HEPTACOM, the "Parties").

5.5.1 1. Definitions

1.1 "**Contribution**" means any intellectual creation (software and / or documentation), including any revisions or additions to existing works submitted by the Contributor to a project and in which the Contributor has the rights of use and exploitation under copyright law.

1.2 "**Contributor**" means the copyright owner or legal entity authorized by the copyright owner that is entered into this Agreement.

1.3 "**Submitting**" means any form of physical, electronic or written correspondence transmitted to the project using "GitHub".

1.4 "**Project**" means any open source project from "HEPTACOM" on "GitHub".

1.5 "**GitHub**" is a free web-based service, which is used by HEPTACOM as a social coding platform for software development projects.

5.5.2 2. Contractual Object

The parties agree on the non-remuneration of the Contributions submitted by the Contributor. The Contributor may submit one or several Contributions to one or several projects. The logic of the CLA is that the Contributor submits Contributions, including the corresponding usage rights, to HEPTACOM.

5.5.3 3. License Grants

3.1 **Grant of Copyright License.** The Contributor hereby grants HEPTACOM the worldwide, free, irrevocable, unlimited by time and location (for the duration of the copyright), right to transfer any number of simple rights of use to third parties and the right to grant sublicenses to third parties, in particular:

a. the right to publish the Contribution,

b. the right to alter the Contribution, the elaboration of derivative works on the basis of the Contribution as well as derived works that contain them, and the merger of the Contribution with different software code,

c. the right to reproduce the Contribution in an original or modified form,

d. dissemination, public accessibility and public communication of the Contribution in original or modified form.

Moral rights remain unaffected as they are recognized under current law and a waiver in this regard is not permissible.

3.2 **Grant of Patent License.** According to any Contribution (reference to point 1.1) the Contributor hereby grants to HEPTACOM a perpetual, worldwide, non-exclusive, free of charge, royalty-free, irrevocable, unlimited license with the right to transfer any number of non-exclusive licenses to third parties and the right to grant sublicenses to third parties. Furthermore, the Contributor hereby grants HEPTACOM the right to produce, use, sell, import or otherwise transfer the Contribution and the Contribution in combination with the materials (as well as components of this combination). This license applies to those patent claims licensable by the Contributor that are necessarily infringed by the Contribution alone or in combination with the materials to which the Contribution were submitted.

5.5.4 4. Rights and obligations of the Parties

4.1 **Ownership.** The Contributor guarantees that each Contribution is the Contributor´s original creation. The Contributor guarantees that any Contribution is free of patent or other industrial property rights or copyrights of third parties and that Contributor is legally entitled to grant the licenses above.

4.2 **Disclosure.** If the Contribution contains any rights of third parties, the Contributor is obliged to provide complete details of any third-party license or other restrictions (including, but not limited to, related patents and trademarks) associated with any part of the Contribution.

4.3 **Support.** The Contributor is not expected to provide support for Contributions.

4.4 **Change notice.** The Contributor is obliged to inform HEPTACOM if there will be changes according to the aforesaid provisions.

4.5 **Licensing obligations of HEPTACOM.** HEPTACOM undertakes to license a Contribution that is compatible with the existing licenses in the project, including all rights to acquire future license versions.

5.5.5 5. General Provisions

5.1 **Liability.** The liability of the Contributor is limited to intent and fraudulent intent. The compensation for negligently caused consequential damages by the Contributor is excluded. The Contributor shall be liable for his representatives and vicarious agents according to § 278 of the German Civil Code (BGB).

5.2 **Term and Termination.** This Agreement shall enter into force upon signature and shall be entered into for an indefinite period of time. Both parties may terminate this Agreement by giving six (6) weeks notice at the end of a calendar quarter. Upon termination of this Agreement, HEPTACOM shall no longer receive any Contribution from the Contributor.

5.3 **Survival.** Upon termination or expiration of this Agreement, all terms of the Agreement, including the license grants, shall remain in full force and effect, with the exception that the Contributor will no longer make submissions to HEPTACOM.

5.4 **Governing law and legal venue.** This Agreement and any disputes, claims, court proceedings or other procedures arising out of or related to it, shall be governed by the law of the Federal Republic of Germany under exclusion of the provisions concerning conflict of laws and the UN Convention of Contracts for the International Sale of Goods. Exclusive legal venue is HEPTACOM’s registered place of business.

5.5 **Amendments.** HEPTACOM may amend this Agreement at any time by providing notice to the Distributor or by posting the revised Agreement online where Contributions are made. By making a subsequent Contribution, the Contributor thereby agrees to the revised Agreement for all Contributions made by the Contributor. If the Contributor do not agree to the revised Agreement, the Contributor may stop making Contributions.

6. Reference

6.1 Reference

Here you find technical references about design decisions, graphical representations and plain lists of information.

6.1.1 Plain information

Glossary

So we are all on the same page.

License

See our open source license.

6.1.2 Graphical references

Package structure

See all the components of the framework coming together.

Data flows

See different data flows.

Basic data flow in detail

See the three components of the basic flow in detail.

6.1.3 Architecture Decision Records

- 2022-10-06 - Filesystem abstraction with stream wrapper
- 2022-06-12 - Type safe class strings
- 2022-03-02 - Final classes
- 2022-01-24 - SemVer with generation version
- 2022-01-05 - Code documentation
- 2021-10-30 - Route capabilities
- 2021-10-29 - Flow components are not CRUD
- 2021-09-25 - Optimized storage actions
- 2021-09-06 - Exception and log message codes
- 2021-08-24 - PHP 8.0 named arguments
- 2021-06-17 - Flow component short notation
- 2021-04-13 - Portal dependency injection implementation
- 2021-02-03 - Direct emission exploration

- 2020-12-10 - Portal service container
- 2020-10-30 - Job messages and payloads
- 2020-10-15 - Portal status reporters
- 2020-08-28 - Parallelization locks
- 2020-08-10 - Architecture decision records
- 2020-04-30 - Contracts and interfaces
- 2020-01-27 - Telemetry recording

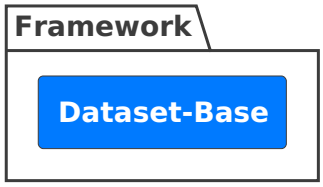
6.2 General resources

6.2.1 Package structure

This software is divided into several repositories that have composer dependecies on each other. Some of the packages are always required for a functional HEPTAconnect ecosystem while others are more or less optional or specific to the use case. This article attempts to clarify the structure of the different packages and outline their role in the ecosystem.

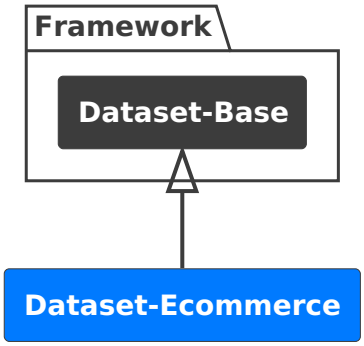
DATASET-BASE

HEPTAconnect is all about data. Reading data, moving it from one point to an other and writing it again. To make different APIs understand each other, they need a common ground to understand each other. A dataset is a group of class definitions for a type of data. Usually these data types are grouped into sets by their topic. The dataset base consists of interfaces and helper classes to make up a base for the individual datasets.



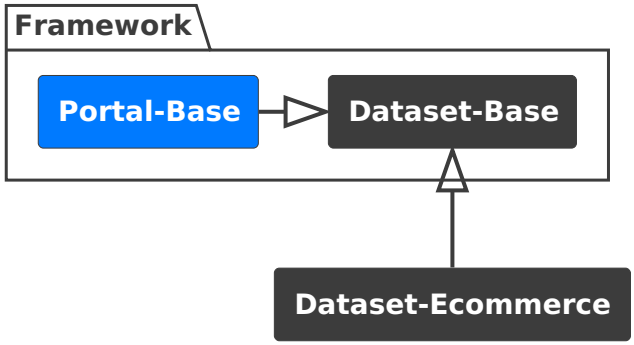
DATASET

A single dataset can hold a number of classes for different data types. Datasets can also require other datasets to make up larger sets.



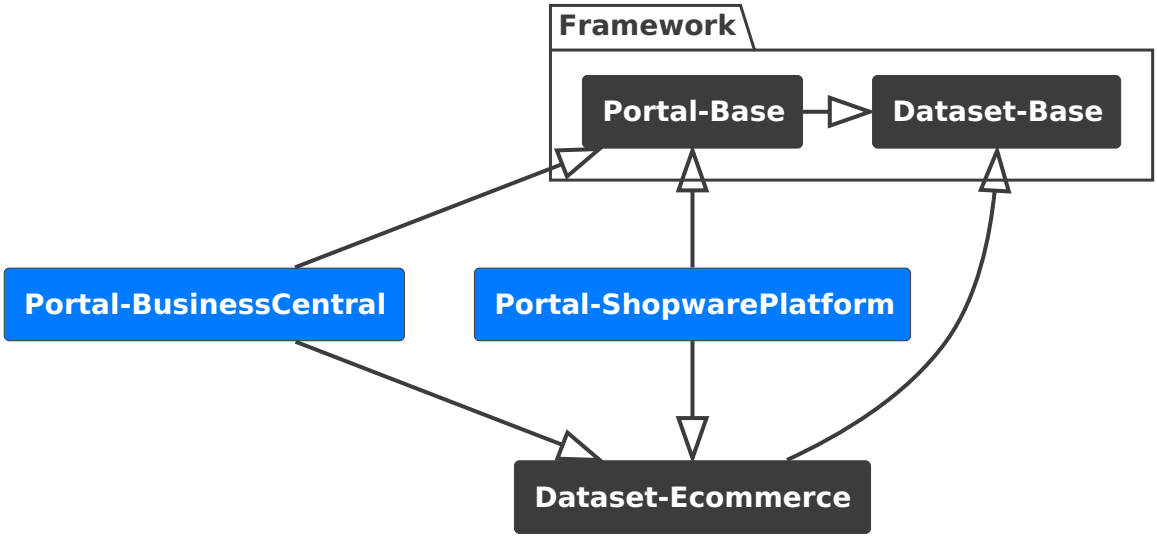
PORTAL-BASE

Since HEPTAconnect itself is not much more than a framework, it does not come with any external connectivity. To connect an external API, you will need to provide a portal for this API. A portal has to require the portal base and whatever datasets it may support. The portal base comes with structs and interfaces that a portal will need in order to work.



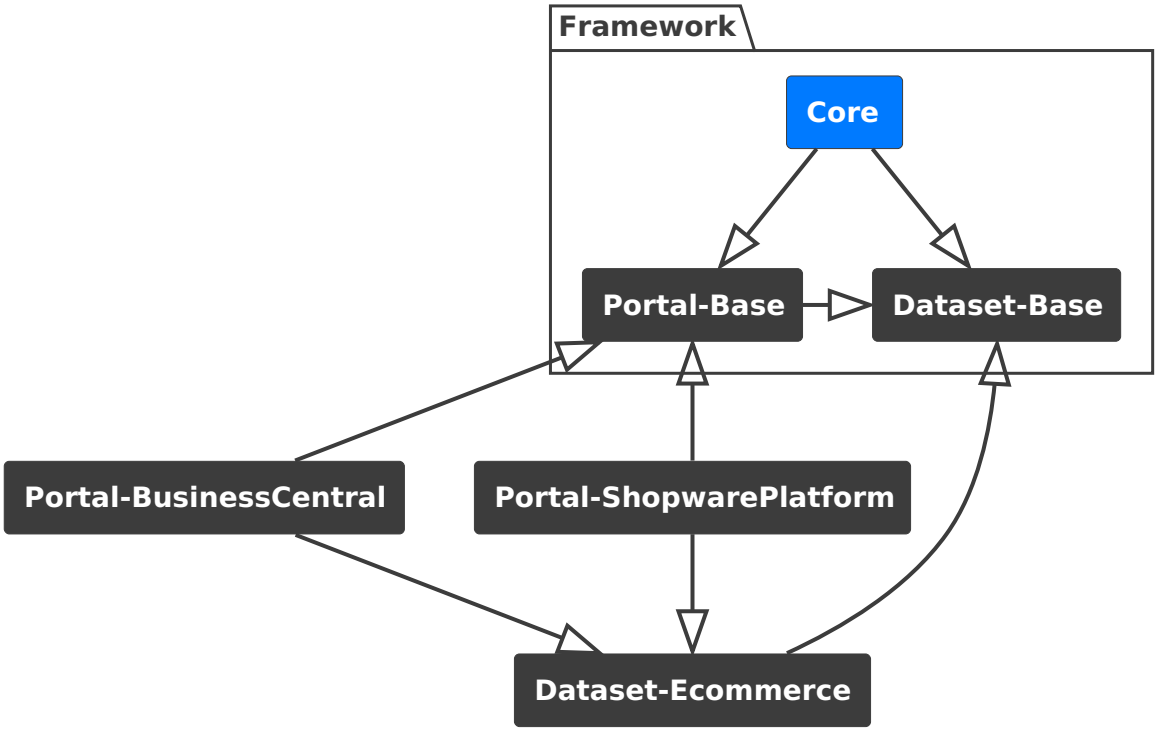
PORTAL

Because HEPTAconnect should bring data of different systems together, it has to be someone's responsibility to actually connect to different systems. This is where portals come into play. A portal is a package with emitters and receivers that can read data from and write data to an endpoint. In most cases this endpoint is an API of some sort, but it does not have to be one. In theory this can also be an access to a static local file or a local database. The important part is that a portal connects an external system with the HEPTAconnect ecosystem. The portal has to require all its supported datasets. Portals with shared supported datasets are natively compatible with each other, as their data can be easily transferred from one portal to the other.



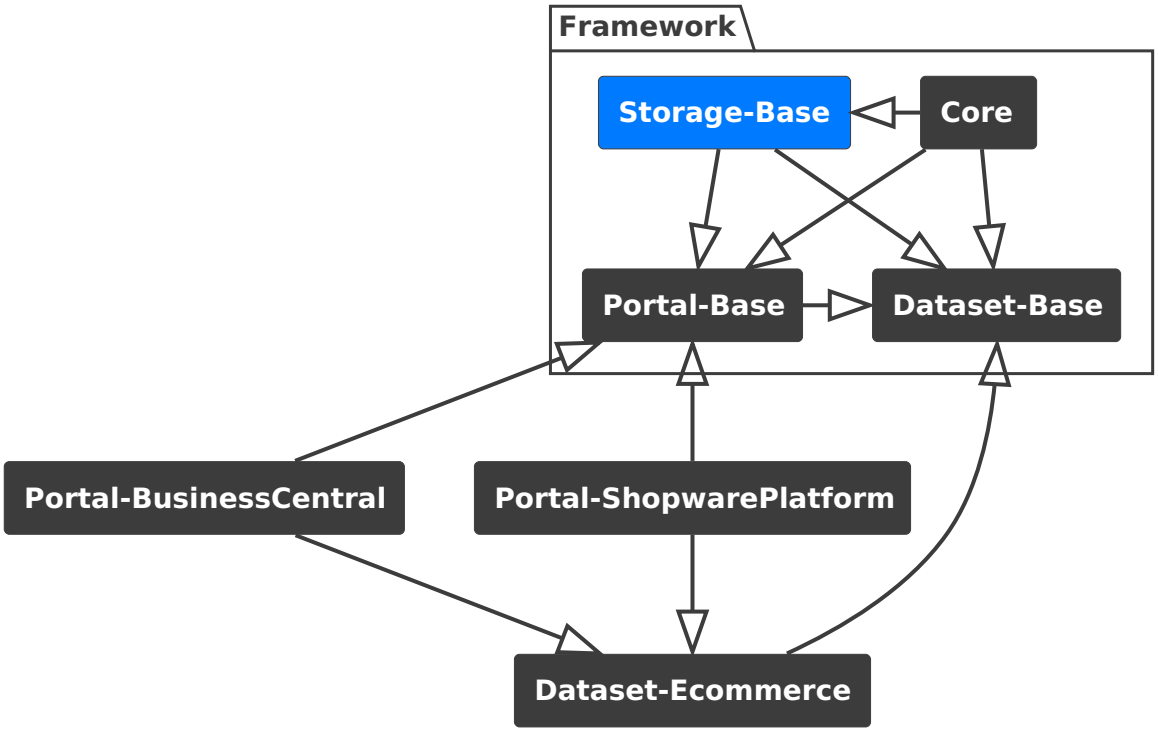
CORE

At its core HEPTAconnect manages data streams between different endpoints via asynchronously handled messages. One side goes through its entities for a dataset and emits whatever it can find. The other side receives these entities and saves them to another endpoint. It is the core's job to coordinate this traffic and keep things organized. So the core provides a router, a mapping service, an emit service, a receive service, http handling and other tooling.



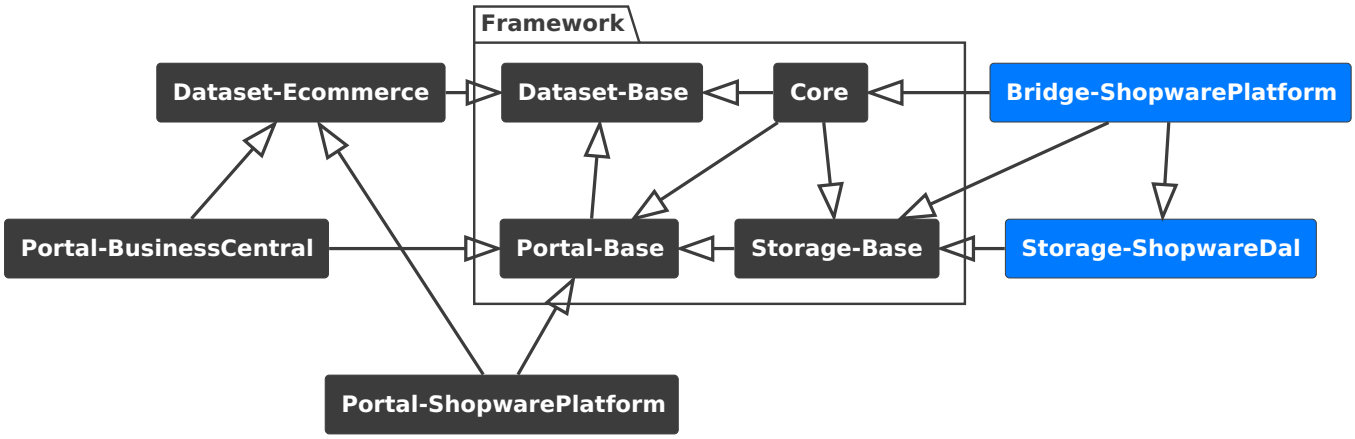
STORAGE-BASE

Certain components of HEPTAconnect require a form of persistent storage. An example is the mapping of entities. To remember which records of different portal nodes are actually the same entity, a mapping is created and stored in a storage. The storage base will only provide interfaces for the storage, so the core can interact with the storage but does not need to know the actual implementation.



BRIDGE

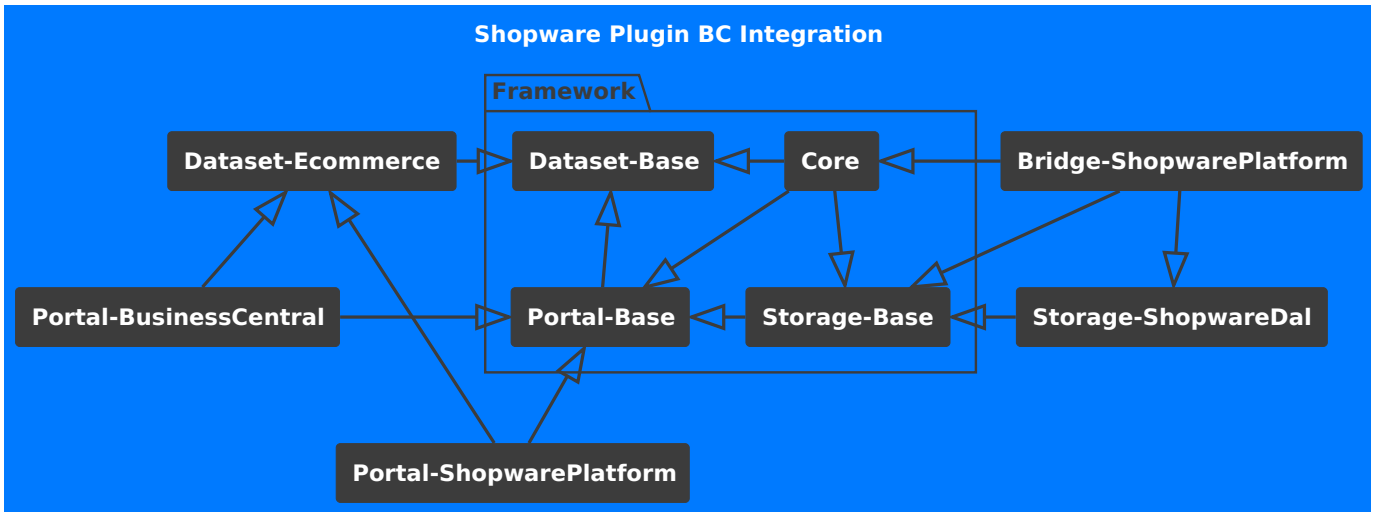
HEPTAconnect is designed to be able to adapt to its surrounding software. Therefore the core itself will not run without a surrounding runtime. To make this work, there are bridges to connect the core with a runtime. The runtime will then (through the bridge) provide a storage, a messenger and several other components that the core will then make use of. Because of this approach, the core is very portable and can run in a number of runtimes (if they can provide all requirements).



INTEGRATION

The integration is the one package that holds it all together. It is a composition of all required packages necessary for the use case at hand. So this package really changes from project to project and is the most individual part of the software. Typically an integration is composed by specifying all portals that should be connected with each other and a runtime for the software to run. In this example we choose the Shopware bridge as the runtime and the portals for Shopware and Business Central as our portals.

Because our runtime is Shopware, the integration has to be a Shopware plugin with the bridge and the portals registered as additional bundles.



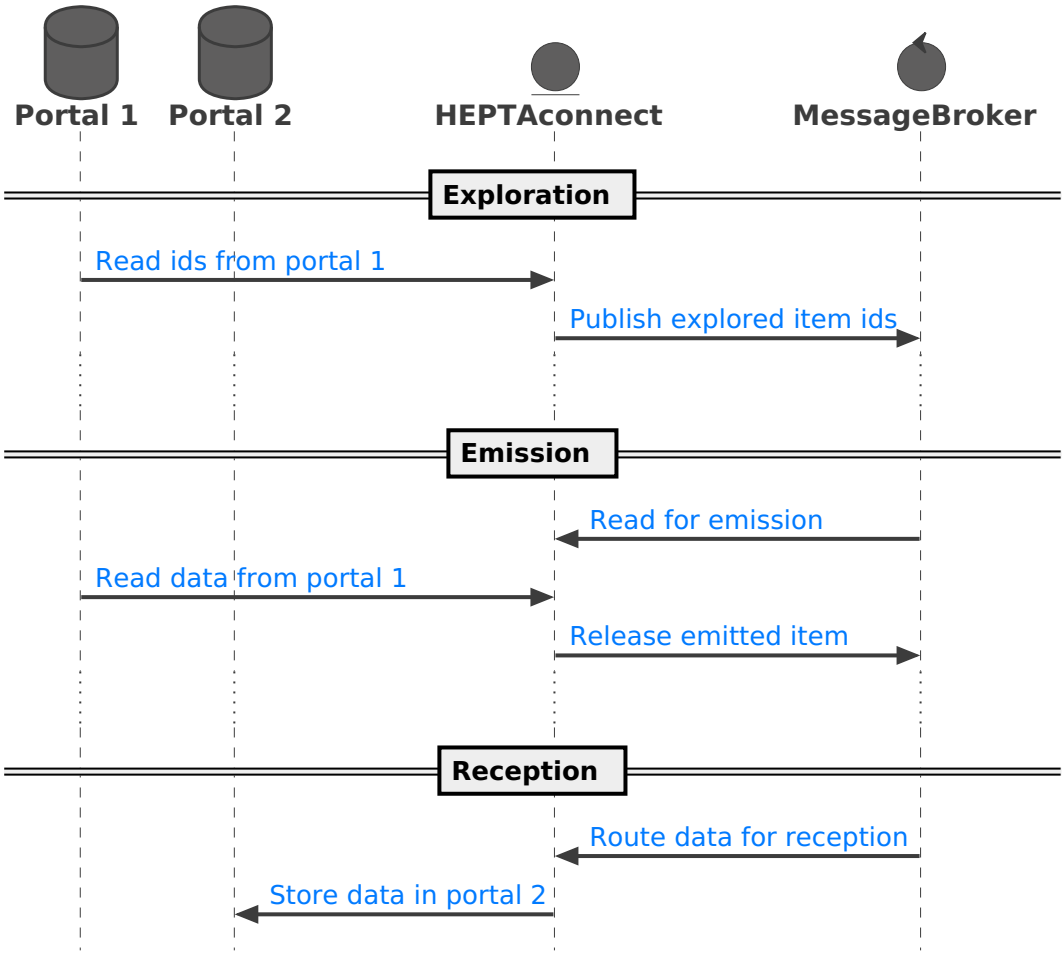
6.2.2 Data flow

Data flow

In HEPTAconnect we separate different steps where data processing is happening to have different entry points for developers and enable horizontal scaling for each step. The main steps are exploration, emission and reception. The exploration can be triggered from different places and will follow in emissions and receptions when the data routes exists. In the following paragraphs you will see what kind of data flows can occur:

BASIC FLOW

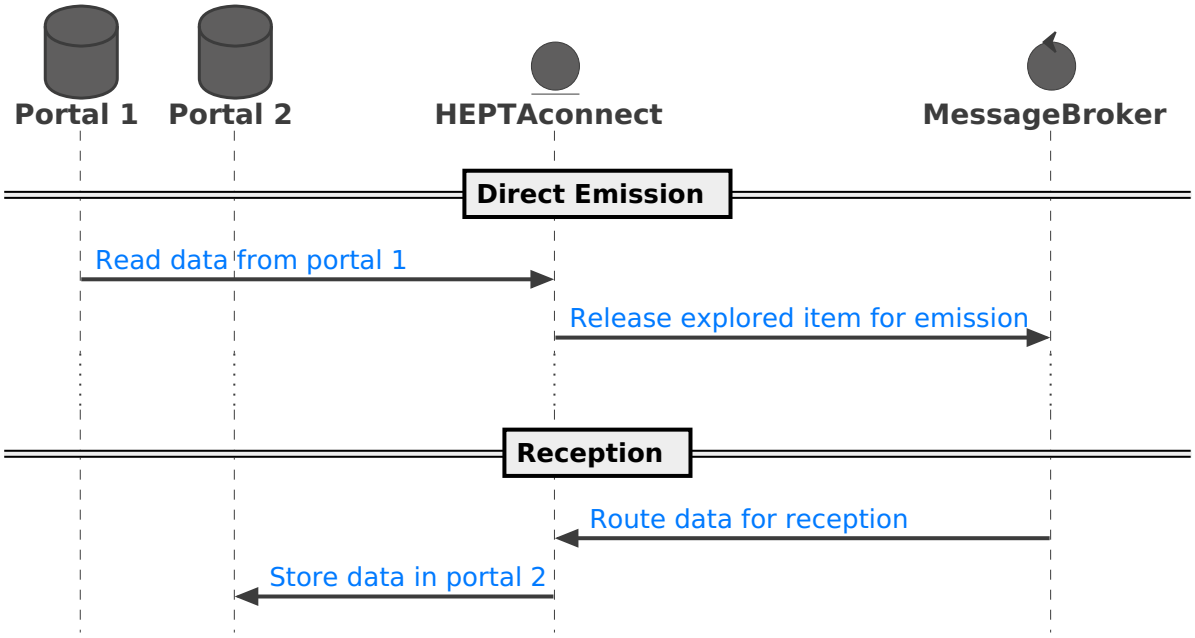
The basic flow of the previous mentioned steps exploration, emission and reception in their most common form. For this you need to implement a handler for each step: An explorer, an emitter and a receiver.



Click here to read more about the details that happen within the basic flow.

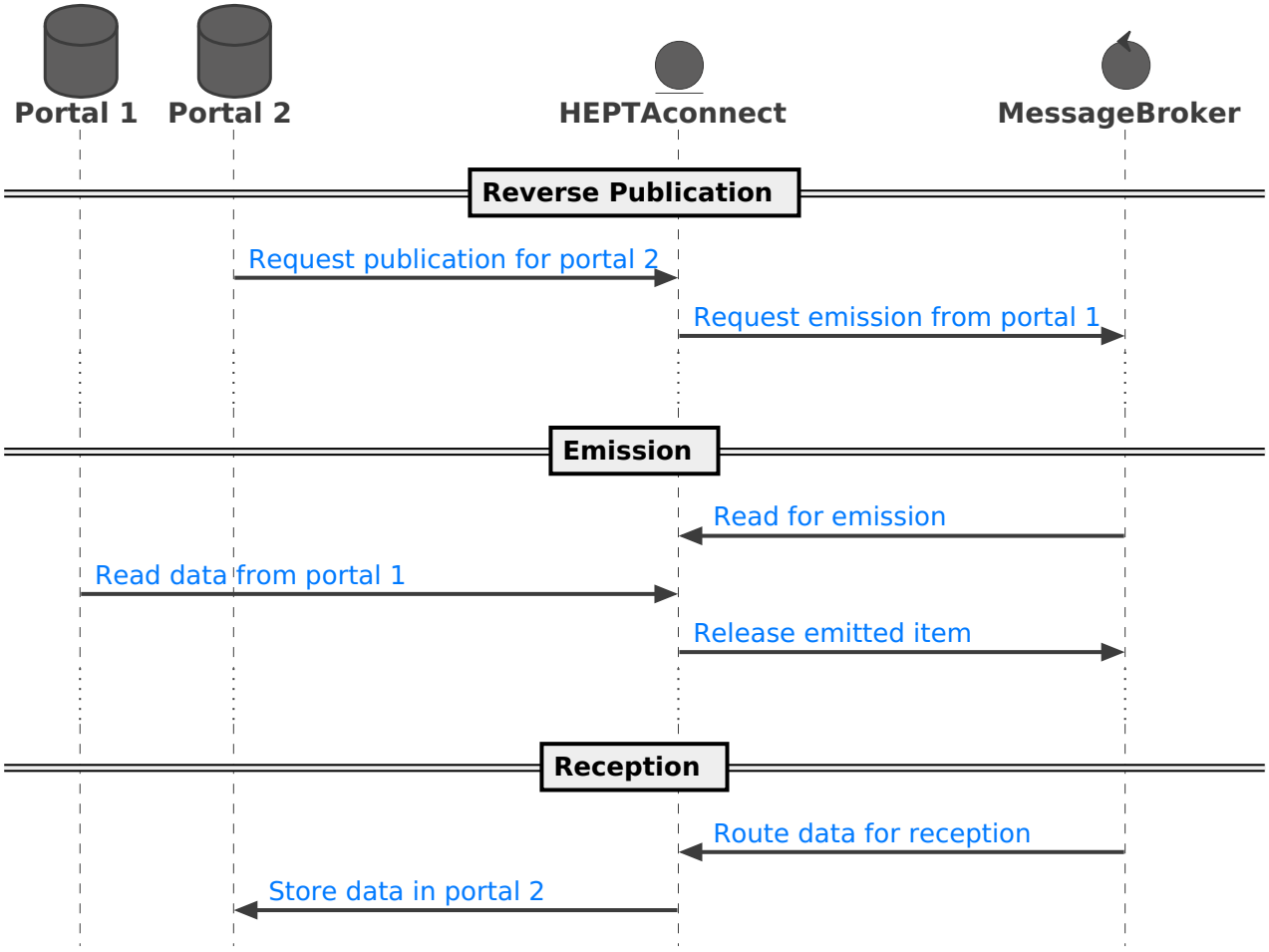
DIRECT EMISSION FLOW

This is a condensed form of the basic flow as the first two steps are merged into one. A very useful pattern for sources that do not differ between gathering selecting primary keys and their corresponding data on it. For this flow you only need to implement explorers as direct emission explorers and receivers. To ensure other flows like the next one you still have to provide an emitter which can be omitted otherwise.



REVERSE PUBLICATION FLOW

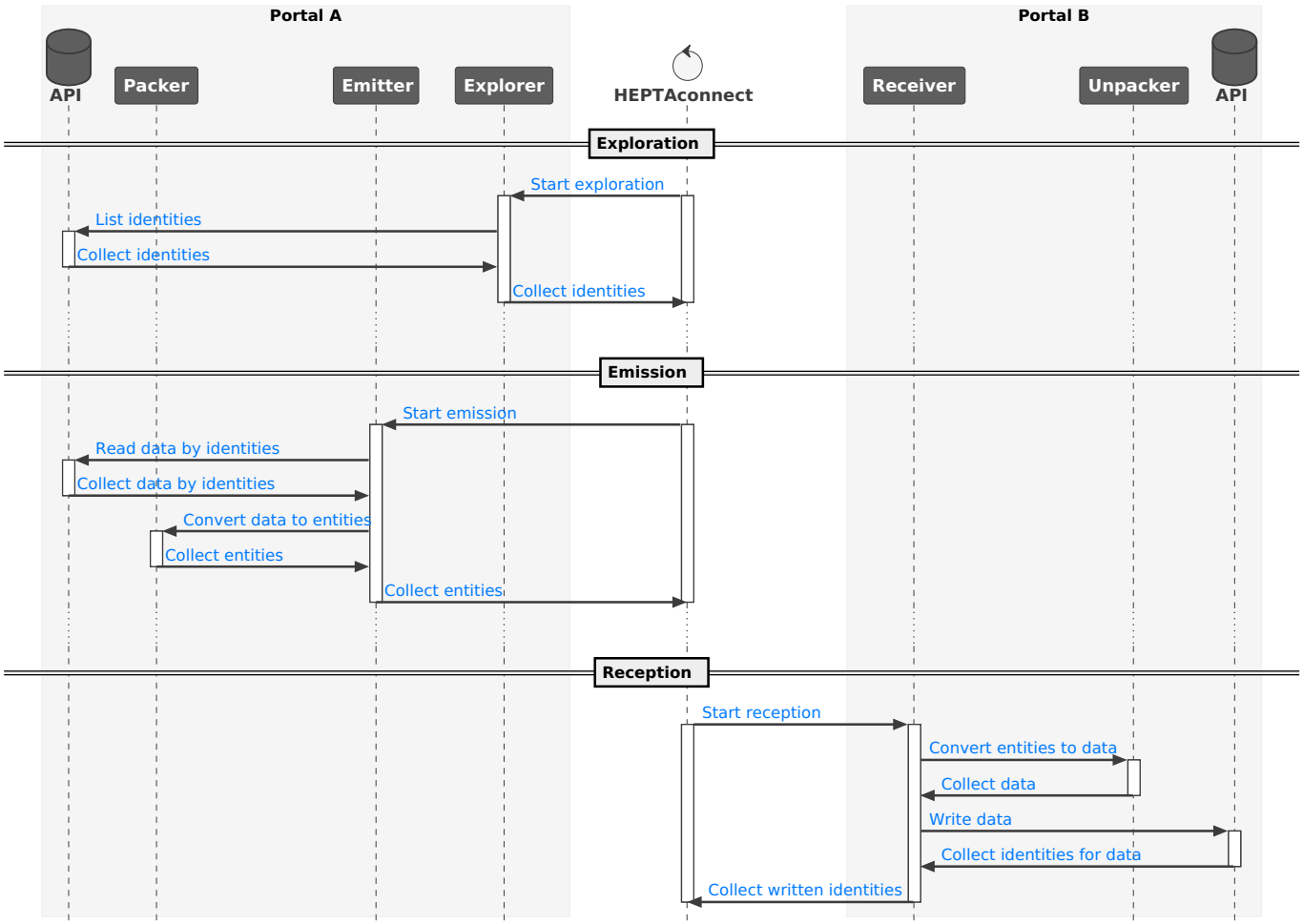
The reverse publication is requesting data from a previously running transfer a second time to keep data up-to-date. This is useful for any event driven data transfer that has to happen on demand.



Basic flow in detail

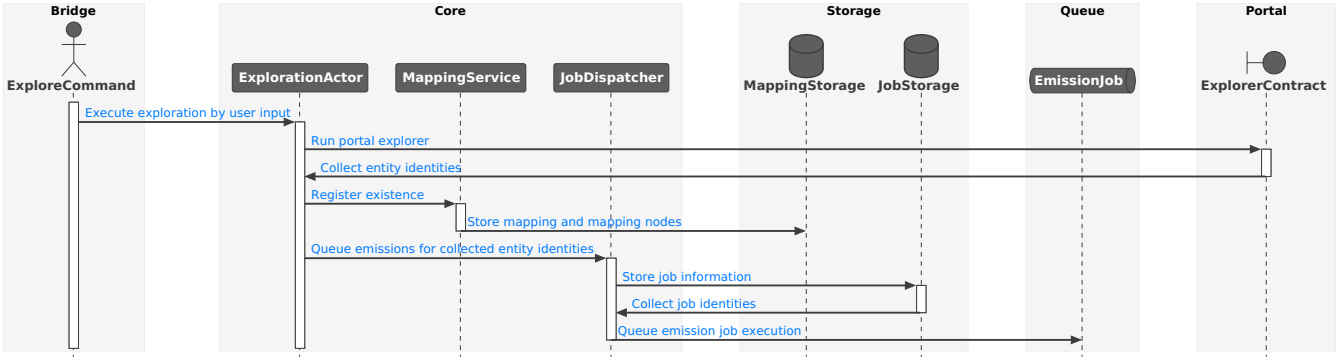
In the following we get more into the details from the basic flow from two perspectives. First perspective is seen from the portal and the second perspective is based upon the HEPTAconnect Core.

PORTAL'S POINT OF VIEW

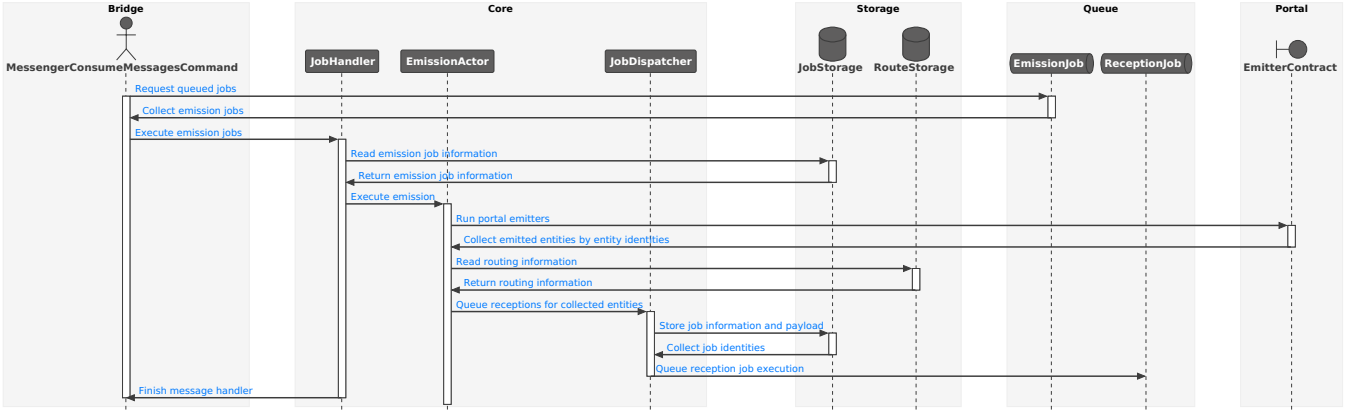


CORE'S POINT OF VIEW

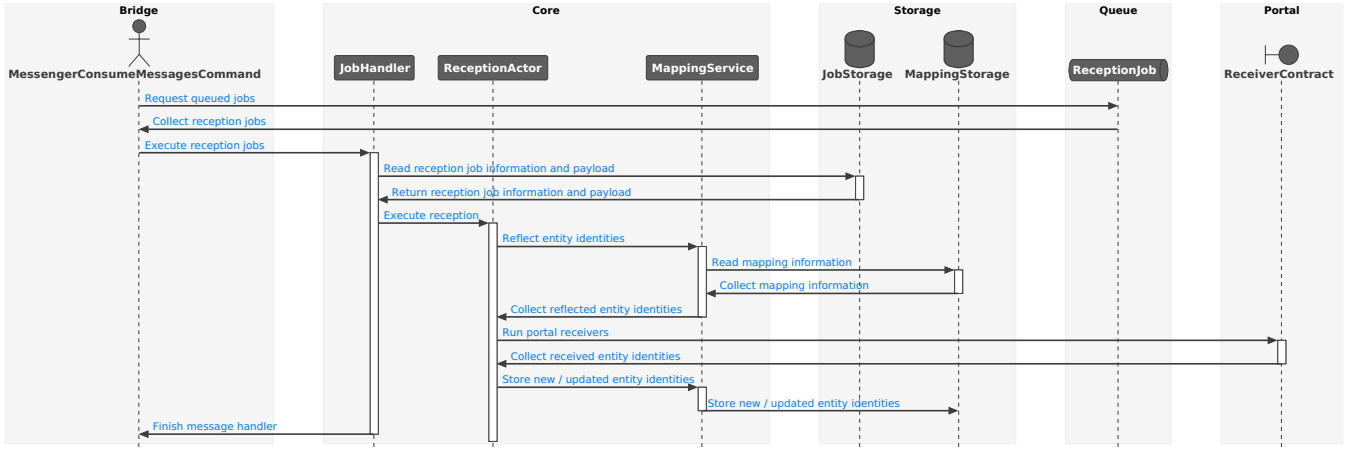
Exploration



Emission



Reception



6.2.3 Mapping

Since HEPTAconnect enables various systems to exchange data and keep it synchronized, it will need to keep track of the data it transfers from one system to another. This is done via mappings. This article will go into detail about how mappings are structured and how they work in HEPTAconnect.

MAPPINGNODE

Every entity in a PortalNode (that should be visible to HEPTAconnect) must receive a mapping. So a mapping represents a single entity inside of a PortalNode. Every mapping also has a MappingNode. These are used to connect different mappings from different PortalNodes with each other. Several mappings can share a MappingNode to indicate that they mark the same entity throughout different PortalNodes. A MappingNode itself knows its data type and its origin (the PortalNode of the first mapping associated with this MappingNode). The associated mappings know the external identifier and have an association to their PortalNode.

Example: A product in an ERP system is published to HEPTAconnect. This means, it now has a mapping which in turn has a MappingNode. HEPTAconnect then transfers this product to an ecommerce system. Upon creation the ecommerce system responds with an identifier for the newly created product. HEPTAconnect will now save a new mapping for the product in the ecommerce system with the same MappingNode. So there are now two mappings for the same product in two different PortalNodes that share a MappingNode.

In some situations mappings are not attached to a mapping node but instead preconfigured using an identity redirect. This is useful when multiple mappings in one portal node are represented as a single mapping in a different portal node. As this scenario is not supported by the singularity approach of a mapping node, redirects are applied separately and manually created. PortalNodes cannot influence redirects as they are cross portal nodes by design.

PUBLISHER

To make HEPTAconnect aware of an entity inside a PortalNode, that entity has to be published. The publisher will create a mapping and a MappingNode for an entity and it will schedule the entity to be emitted. Because the actual emitting will happen asynchronously, the publisher can be called during a web request with a minimal performance impact. For the initial integration into an existing HEPTAconnect ecosystem, it is recommended for a PortalNode to publish every entity that should be synchronized by HEPTAconnect. This process is called exploration.

EXPLORATION

A recommended step in adding a new PortalNode into a HEPTAconnect ecosystem is the exploration. A portal can ship multiple explorers that will each publish every entity of a certain type from its PortalNode. While this process could in theory be done manually, it is recommended to create classes implementing the ExplorerContract. The benefit is a better integration into automated processes of HEPTAconnect, so your exploration process can be triggered by the system rather than relying on a manual trigger.

IDENTITIES

To perform operations on mappings, various storage actions can be used depending on the exact use case:

- `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Identity\IdentityMapActionInterface` used to create missing mappings for entities
- `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Identity\IdentityPersistActionInterface` used to update mappings on existing mapping nodes
- `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\IdentityRedirect\IdentityRedirectCreateActionInterface` used to create identity redirects to connect multiple mappings on one portal node to a single mapping on a different portal node

These service can save mappings to the storage and find a counterpart of a mapping for another PortalNode. This is done after an identifier has been set on an entity by a receiver.

The process of finding a counterpart of a mapping for another PortalNode is called reflecting. The `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Identity\IdentityReflectActionInterface` will check, if the provided entities have known mappings or identity redirects in the management storage. Afterwards the service checks, if a reference mapping for the requested PortalNode already exists and assigns it.

ERROR HANDLING

There are situations that are prone to error regarding mappings, like e.g. when a mapping is passed to a receiver but the receiver throws an exception. These errors are stored in the database alongside the mapping. The idea is to associate errors to the entities they are related to.

6.3 ADRs

6.3.1 2020-01-27 - Telemetry recording

Context

API web requests, file system access, database requests, message queue dispatches, I/O interrupts, interprocess communication and similar interactions are part of any portal. The usage of these resources should be known to a certain degree for monitoring of usage compared against a usage limitation, order of calls across portal nodes and their time of action.

Decision

The information have to be connected to a portal node and its structural resource subdivision. There is no context given as it is just about when and what kind of categories are affected. For example a processing involving the transport of a file does not contain the complete file but can be tagged with e.g. the mime type or encoding, ... There need to be common decorators for common implementations to simplify automatic recording like PSR HTTP client. Adding common decorators in a global registry simplifies the usage of portal node developer that can depend on common decorator implementations. The telemetry entry only represents a single direction in a synchronous interaction and should represent two directions as two entries in an asynchronous interaction. Incoming web requests and cronjob runs are automatically recorded. Telemetry has to be optional by core configuration as the storage is impacted heavily and it can affect the performance. Storing telemetry data is not allowed to break the portal node flow by e.g. throwing an exception. Recording describes an incrementable reference object that stores the timestamps of each increment and is taggable.

The exact way for analysis of the recorded data is up to discussion and postponed to a different point in time.

Consequences

PROS

- Analysis of API usages helps to identify upcoming limitation breaks.
- By requests across portal nodes it helps to identify errors in flow and order.
- Taggable incrementable reference objects allow for data point analysis of groups and of fine grain later on.

CONS

- Although supporting infrastructure is given it adds a heavy complexity layer to portal node developers that needs to be simplified.
- Depending on the implementation the storage is heavily impacted by amount of data and frequent usage. This can end up in a project-wide bottleneck.

HOW TO USE

A suggested pattern is that portal class prepares API clients. They should be easily wrapped by a decorator whereas the instantiation should be lightweight. Any context is able to supply a telemetry accessor that can be passed to other services and decorators for easy usage.

Related thoughts

The space reduction of the storage and performance impact can be accomplished by a recording instance that receives the prepared payloads via a message broker. Telemetry recording can be stored in-memory and flushed later for performance reasons. Telemetry should not be abused for success/failure metrics as this is part of internal analysis on mapping errors.

6.3.2 2020-04-30 - Contracts and interfaces

Context

There has to be a guide how to structure code to allow extendability. We have to ensure that functionality can be exchanged without interfering with adjacent packages within the package hierarchy.

Decision

The approved approach is using the language feature type hierarchy. Using interfaces, traits and contracts (abstract classes) is a good way to structure and allow replacements by ensuring certain behaviors.

We use interfaces when multiple implementation will exist and are unknown at any time for the package that it is introduced in. For example, we do not know what kind of storage is used within the portal-base, but it will need some kind of storage.

To supply some basic logic for commonly used interfaces we can provide traits for others to implement them easier.

We use contracts similar to interfaces but use their advantages to contain any logic beforehand. This enables us to add additional code later with a reduced level of changes that can be non-breaking without removing the replacing option. Contracts are best without dependencies that have to be given in the constructor as this forces other implementations to follow this pattern regardless whether they need it.

Consequences

PROS

- Others can build their own logic and replace existing one more easily

CONS

- This adds more complexity in designing functionality as decisions have to be made whether interfaces or contracts are best suited

6.3.3 2020-08-10 - Architecture decision records

Context

We document architecture and technical decisions for HEPTAconnect. Inspired by https://github.com/shopware/platform/ we follow the same principles and use ADRs to keep track of decision making to make it easier to understand why things are how they are:

- A Simple but Powerful Tool to Record Your Architectural Decisions
- Documenting Architecture Decisions
- When should I write an Architecture Decision Record

Decision

Having the ADRs as part of the versioning adds more pros:

- Decisions remain in sync with the code itself
- The Git history is also the decision history
- Decisions are public available and accessible for every developer
- Also external developers can add new ADRs via GitHub pull requests

Consequences

Every architectural change or addition should contain a Markdown file like this to have a brief understanding what are the pros, cons and how it should be used. Until the release of version 1.0.0 it is ok to add ADRs for decisions made in the past. This is the reason why you probably find ADRs with a timestamp before we decided to add ADRs.

ADRs have to approved by a maintainer when they are proposed by a contributor. As a past ADR can't be changed it has to be marked as deprecated by copying it into the deprecated folder, change the original file to link to the copy and refer to the new superseeding ADR.

How does an ADR look like?

You can use this first ADR as an orientation. The filename of the ADR should contain the date and a meaningful title. The content of the ADR should always use the following template:

```
# [Date] - [Title]
## Context
## Decision
## Consequences
### Pros
### Cons
### How to use
## Related thoughts
```

6.3.4 2020-08-28 - Parallelization locks

Context

In horizontally scaled processes problems of parallelization can happen like race conditions (a resource is accessed simultaneously by multiple processes and can have a different value for each process as a previous one wrote to the resource before).

Decision

To support horizontal scaling there is also the need to allow resource locking. As the resource accessing is part of transmitting data over time or space, the storages and portals should be able to use this feature. Therefore we will put in the contracts into the portal-base. As locks need some sort of storage to maintain a lock state an additional repository has to be added to the storages.

The featured methods shall be:

- isLocked
- lock
- release

A utility class or methods to easily write spinlocks by time or iteration shall be added.

Locks shall rather run out of time or a different measurement instead to unintentionally lock a resource.

Consequences

PROS

- Horizontal scaling is easier to support for portals and storages
- It is easier to implement checks to prevent resource requests that should not be overlapping

CONS

- Portals and storages who uses this feature will perform worse than without using this feature

6.3.5 2020-10-15 - Portal status reporters

Context

A portal should have some kind of status page. It has to tell an administrating person and a health check automation whether the portal is in a good state.

A good state can differ from the point of perspective. A configuration might be syntactically correct but is not able to setup an I/O connection to the datasource.

Decision

- A portal and the portal extension have to be able to provide new status topics and have impact on the contents they report.
- Every reporter has to expose JSON serializable content for easy automation access.
- Every reporter should expose a boolean value keyed with the topics' key to determine whether the report displays a good state.
- Every portal should expose a status reporter for topic `health` when the portal interacts with a datasource connected via I/O operations to determine correct configuration and connectivity.
- Portal extensions have to prefix their own keys they expose with a reasonable identifier.
- Every topic should be accessible on their own.
- A status report should act fast and use as little I/O operations as possible to allow frequent health checks.
- A status report should be promoted for the following use cases:
- static information that are not part of the providing composer package
- health check of the datasource connection
- portal internal behaviour analysis (last time usage, remaining API calls by time limitations)
- configuration support

Consequences

PROS

- A portal has a way to expose data independently of data transportation.
- Automated processes and humans can process this data.
- Health checks can be implemented in a standardized way for every portal.
- Multi-step configuration is easier to provide.
- Internal API usage can be exposed for behaviour analysis.

CONS

- It can be misused for data reading that does not belong to the intended use cases.

HOW TO USE

A command like `heptaconnect:portal-node:status PortalNode:123 health | jq -e .health` as a simple health check condition can be setup as crontab entry. An other example is `heptaconnect:portal-node:status PortalNode:123 config` to display possible values for further configuration.

Regarding the usage of behaviour analysis it is suggested to compare the intended functionality to be achieved is compared against the telemetry feature as this allows a very specific way to deal with behaviour analysis.

6.3.6 2020-10-30 - Job messages and payloads

Context

In case of a structural change in a dataset you might need to migrate serialized data in a way to make in work with the latest code. The data that is affected of the structural change can still be within a message queue provider and is often out of access until message handling. You could unintentionally send duplicated messages to drain performance and increase I/O operations overall.

Decision

- Extract job actions from the messages into the storage.
- Separate job payloads from their actions.
- Prevent sending of duplicate messages.
- Normalize the structure of a job regarding the current message structure.

Consequences

This change adds a little overhead on the message dispatching but adds multiple benefits regarding upcoming structure changes and future I/O operations that can be prevented.

PROS

- Prevention of duplicate messages reduces handler calls and follow-up I/O operations.
- Message payloads can be accessed without knowledge about the used message provider.
- When emptying a message queue the messages can be reconstructed in a plausible order.
- The message provider has less data to store than before.
- It is easy to add new job types in the core without changing the storage.

CONS

- More I/O operations have to be done when the message handlers have to hydrate the message with the payload storage before processing can be continued.
- It is difficult to add new job types that are not similar to the existing jobs when there is a structural mismatch.

Related thoughts

Datasets could use some sort of migration pattern which can be applied by the storage implementations onto their stored payload to react to a structural change. When a new job has to be introduced that is not related to this job pattern which is tied to mapping components a new message type can be introduced instead.

6.3.7 2020-12-10 - Portal service container

Context

Portal extensions shall be able to interfere with any operation the supported portal is doing to make any business logic within the supported portal adjustable. This is already possible having the explorer, emitter and receiver stacks when it is about changing the flow and the incoming data from HEPTAconnect and the outgoing data towards HEPTAconnect. There is no way yet to change the exact behaviour how an API is used within the supported portal. It has been suggested to expose the operational APIs as public methods in the supported portal class. Portal extension are able to interact with the same APIs like their supported portal but not yet able to change the implementation of these public methods.

Decision

- Use PSR-11 containers
- Do not use inheritance / decoration between portal extensions and the portal itself
- Do not use a container builder as there is no common interface yet and is not needed yet
- Do not use a hook pattern

Consequences

PROS

- Portal developers can decide what is allowed to be changed
- Portal and portal extension developers can use the commonly known container way of publishing services within an application

CONS

- Portal developers have to publish every implementation to allow being changed
- Containers have to be managed per stack to prevent
- To have static typing you have to add additional `instanceof` checks or trust type hints

Related thoughts

One could use inheritance and decoration pattern to allow portal extension claim to be the supported portal but this moves all the development overhead to the developer

There is a follow up to this regarding the exact implementation.

6.3.8 2021-02-03 - Direct emission exploration

Context

Data sources like plain tables (.csv, .tsv, .txt, .json, .xml) and slow/rate-limited APIs both share the fact that you want to keep the interaction count low. Plain tables need to be parsed and are missing an index for fast navigation wich reduces speed in reading and often uses computation time and memory. Rate-limited APIs should not be consumed twice for the same information. When they allow retrieving list data similar to single entries that should be preferred to allow for smart rate-limit usage.

Decision

- Allow emission to take place in the same moment as exploration

Consequences

- There are now two places that can emit data and therefore the conversion logic from a data source payload to a HEPTAconnect dataset entity should be extracted

PROS

- Portal developers can support multiple data flows
- Named data source can be processed more efficient
- Portal developers do not have to cache data structures on exploration anymore for an efficient emission

CONS

- Portal developers have to decide which of the data flow models they want to support
- Additional complexity in the return type of explorers' explore method as this can change the data flow to a direct emission flow

6.3.9 2021-04-13 - Portal dependency injection implementation

Context

This is a follow-up to the ADR about general service containers.

Dependency injection is a common pattern to create reusable components that can build upon each other. Portals will have to communicate with their API of choice in different flow components. Presumably a portal developer wants to build an API client that can be used within all flow components. This is where dependency injection comes in handy. Depending on the implementation this can also be used to decorate services which will add more freedom for modifications. We were able to provide a service container for each portal node matching PSR-11 in the past, which allowed easy access to services but no injection into flow components. There is the PSR-11 standard to define a service container but not a service container builder and therefore there are no drop in implementations.

Decision

- Use Symfony dependency injection
- Replace custom service container with Symfony
- Enable auto-wiring, auto-configuration, auto-binding and automatic PSR-4 resource loading
- Automatically load flow components and drop their definition from portals

Consequences

PROS

- Portal developers are most likely familiar with Symfony dependency injection
- Using service definitions based on xml or yaml do not require a tight composer dependency and can be used fluently with different Symfony versions
- Symfony's dependency injection has a very good documentation
- Symfony's dependency injection supports tagged services
- Symfony's auto-wiring, auto-configuration, auto-binding and resource auto-loading allows for zero-configuration dependency injection out of the box for portal developers

CONS

- Portal developers need to know or learn Symfony dependency injection

Related thoughts

We looked up comparisons of different dependency injection implementations and evaluated those against the criteria: * performance in building * performance in usage * the least friction against current and possible future dependencies

Symfony's implementation isn't the best in that comparison paper, but it allows for compilation and is well known in our context of work. In the past we also saw very frictionless migrations between the Symfony versions using service definitions that are based upon xml and yaml.

6.3.10 2021-06-17 - Flow component short notation

Context

When a portal has its code separated into different domains, many flow components will look like this:

```
<?php

define(string_types=1);

namespace FooBar\Emitter;

use FooBar\Packer\BottlePacker;
use FooBar\Service\ApiClient;
use Heptacom\HeptaConnect\Dataset\Base\Contract\DatasetEntityContract;
use Heptacom\HeptaConnect\Portal\Base\Emission\Contract\EmitContextInterface;
use Heptacom\HeptaConnect\Portal\Base\Emission\Contract\EmitterContract;

class BottleEmitter extends EmitterContract
{
    private ApiClient $client;

    private BottlePacker $packer;

    public function __construct(ApiClient $client, BottlePacker $packer)
    {
        $this->client = $client;
        $this->packer = $packer;
    }

    public function run(string $externalId, EmitContextInterface $context) : ?DatasetEntityContract
    {
        return $this->packer->pack($this->client->getBottleData($externalId));
    }
}
```

This sample emitter of about 30 lines of code only consists of, when trimmed down to the essentials, two lines of instructions.

Acquisition of dependencies:

```
public function __construct(ApiClient $client, BottlePacker $packer/*, string $externalId*/)
```

Wiring everything into an emitter run method:

```
return $this->packer->pack($this->client->getBottleData($externalId));
```

Using this perspective we have 28 lines of code that are basically boilerplate. Boilerplate code is code we want to eliminate.

Decision

- Allow declaration of flow components in a callback registration way.

Consequences

- There are now two places that can define flow components

PROS

- Portal developers can wire their API clients to HEPTAconnect infrastructure in a very efficient way
- Not extending certain classes
- Storing services dependencies in fields
- Dropping unused default parameters (like `$context`)
- Portal developers do not have to use them

CONS

- Although discouraged, it is not possible to decorate the generated flow component services as the id naming is not predictable
- Code that follows PSR-4 is next to plain code in the same directory hierarchy

- The generated flow components can't make use of class inheritance features

Additional thoughts

This developer experience is heavily inspired from Laravel route definitions. Mixing PSR-4 compliant code with plain php is also done like this in Symfony bundles.

6.3.11 2021-08-24 - PHP 8.0 named arguments

Context

At one point the old RFC for named arguments got an update, has been approved and implemented. With the implementation of this feature a language aspect changes how public API is perceived. When using PHP >=8.0 function arguments are not just positional but also associative and keyed by their name depending on the callers function calling behaviour.

Decision

We do not support this feature and claim the argument names as private API. This includes to wrap calls for `func_get_arguments` in an `array_values` or similar approaches to remove names from parameters. This feature can be replicated to a certain degree with parameter classes the only contain the data for a method call in a single object, and therefore is not a language feature we depend on. We already use this occasionally with a trend towards this. This allows setting parameters by name via using their respective setter. It also has the same developer experience across PHP versions.

Consequences

PROS

- Contributors to HEPTAconnect packages have one breaking change complexity layer less to work on
- Contributors to HEPTAconnect packages can apply the same backward compatibility promises across all supported PHP versions (which includes versions prior to this feature)
- Contributors to HEPTAconnect packages are allowed to rename parameters

CONS

- Users of HEPTAconnect packages can make use of this feature, but they should implement a test that targets this feature to ensure functionality using HEPTAconnect defined private API.
- HEPTAconnect can be evaluated as non-fully PHP 8 compatible

6.3.12 2021-09-06 - Exception and log message codes

Context

Making log messages that are helpful is difficult for various reason. You can not be sure which persona will read them (either administrator, integrator or portal developer). Therefore, phrasing a good message takes additional thoughts. Log messages in general just have a single line of text and some human-readable metadata that are crammed into the same line of text. Sometimes crucial data is left out of a log message. Most log messages are written when an exception is caught. Exceptions can have integer codes assigned. Exception codes are set on construction, which is in almost all cases the moment they are thrown.

Decision

Exception stack traces are too big for log messages, so we need to refer to them in a different way. When we pass a unique code to the exception constructor, we can identify the source code that triggers the exception immediately. This makes exception codes a good alternative to stack traces. There are also log message written without the situation to log an exception. These message have no code from an exception but can benefit from a code to identify their origin as well. Exception codes and log message codes need documentation with a class reference and a reason for occurrence to supply information for an administrator and a developer.

Consequences

PROS

- Every log message can have a code that identifies origin
- Codes can move with refactoring and can keep their meaning which is better than different stack traces for the same issue
- Codes can be looked up in the documentation with helpful information for any common first-responders

CONS

- A small changes like adding a log message is handled with additional complexity in a release
- More documentation needs to be written

6.3.13 2021-09-25 - Optimized storage actions

Context

In the past we were approaching an entity repository pattern for reading and writing HEPTAconnect internal data. This has been useful for building applications inside various frameworks in the past as their storages often come with a database abstraction layer that follows the same pattern. In any implementation that follows the entity repository pattern we have the issue that reading the data for different use cases is different and therefore performance varies significantly. When we started to extract reading access for mappings in different parts we had much more control in the underlying storage layer.

Decision

We split up the different reading scenarios into separate classes. Writing operations potentially need to read other data first so these need to be extracted as well. All write operations should be transactional to ensure a known state in case of an exception. With their transactional behaviour their return value can safely be a fixed collection of values. Other operations that run indefinitely long as they look for and load data should return an iterable of a single entry. As every operation will have its own class and the intention is not to add future methods into the services, we will use interfaces. Names of the operations are not prescribed and should represent the expected business logic. Ingoing payload- and criteria- as well as returned result objects should have verbose property names to prevent ambiguity errors in the future. With this in mind we found these naming patterns to be useful:

- list - is a non-human-used listing of a search based upon a criteria
- overview - is a human-used listing of a pageable search with various information
- create - creates a batch of entries which should return the primary keys
- get - read a list of entries based upon primary keys
- find - look for a certain entry by its unique components

Consequences

PROS

- Every storage access can be optimized separately
- Storage accessing tests can be mocked more easily

CONS

- We need a services for every storage operation
- New access variations need a new release of the storage-base and all storage implementations

6.3.14 2021-10-29 - Flow components are not CRUD

Context

At the time of writing we have explorers, emitters and receivers as three main flow components. They resemble CR and U from the well-known CRUD. Most APIs are CRUD or BREAD based and therefore match the three named flow components. For now, emitting and receiving entities can be also used differently as this "just" sends data from one portal node and is received by another portal node. Emitters and receivers could send commands instead of entities. As previously mentioned we do not have a deletion flow component. A receiver could receive an entity with a custom deletion command with any previous version of HEPTAconnect. This is discouraged but possible. We have already seen implementations, that receive data but don't write anything to the API the portal resembles. This is a misuse that is similar to described scenario above. Looking at the other existing flow components we also have webhooks and status reporters. These are not related to CRUD at all, so we are not limited to CRUD.

Decision

Receivers are not meant to do everything, when it is about receiving a command. Receivers are meant to be used for entities only. Grouping explorers, emitters, receivers and "deleters" into a single CRUD flow component enforces structures that probably don't benefit APIs, that do not fall into this pattern. Grouping flow components is not helpful when we do not know the possible groups in beforehand and therefore can't be done right. Every other transfer needs a new flow component. As routes connect emitters and receivers they need to learn how to decide which flow components to use on a route. This is described in a different ADR.

Consequences

PROS

- New data flows can be implemented by custom integrations without misusing existing components, which could lead to unexpected behaviour
- Separating different flows into unique components allows for clear code structures

CONS

- New data flows need new flow components to be developed, integrated in routes and implemented
- Routes need to be configured per each flow scenario

6.3.15 2021-10-30 - Route capabilities

Context

Routes define directions for data to flow. The interpretation or use-case for a flow can be different for various reasons. In general, we support read-multiple-times write-multiple-times scenarios, and they are very generous in options to work with but often needs to be limited in integrations. Limitations like transferring data only once or transferring data for finding the equivalent on the target are missing but requested. We need a way to configure route behaviour in core without adding more work to the integrators.

Decision

All limitations (e.g. transferring once) will be implemented as skipping existing steps. These changes in behaviour can be represented by simple boolean flags. Every step that is not a limitation will result in further flow components that will get a boolean flag.

Consequences

PROS

- Common known and implemented behaviours can be handled more globally and applied to any route
- This allows for wide range of operations portal developers can provide which can be combined later on by configuration

6.3.16 2022-01-05 - Code documentation

Context

Code often needs documentation. Not every code is self-explanatory without documentation. In most cases this is due to code separation by interfaces and contracts from their implementation that is used in e.g. strategy patterns. Code documentation online is often easy to query due to use of search engines. Code documentation online is not easy to separate by code version for neither the ones writing documentation nor the ones looking for documentation. Any documentation can easily be forgotten to be updated.

Decision

In our ticket refinement process we state right away which parts of the online documentation are likely to be affected and need to be checked. We add documentation at source code level. We add expectations to interfaces and contracts to reach API providers and API consumers.

Consequences

PROS

- Online documentation is likely to be up-to-date with every release to match its content
- Developers can make use of IDE features like tooltips or symbol navigation to read documentation right away

6.3.17 2022-01-24 - SemVer with generation version

Context

We follow semantic versioning to label our releases with expectations for its users, when upgrading. At point of writing updating to a new major version includes upgrade options, that portals are likely to still work without any changes and storage implementations can continue working on previous storage structures. There can be a time in the future, where these upgrading options are not applicable anymore or at a certain complexity level, that is above the current upgrade expectations. Can we put this expectation in semver?

Decision

Yes, we can. Each semver version part describes a certain expectation regarding the magnitude of changes between two versions. A difference in a version part, that is on the left, has more breaking changes than a difference in a version part that is further on the right. The first number is already allowed to include breaking changes. Increments in the first place are therefore expecting breaks. When we add a number on the left side similar expectations are readable from semver although we do not comply completely with semver anymore. We call this first number "generation" as a follow-up/next generation is allowed to be non-compatible with its ancestors as it evolves.

Consequences

Releases need to have a rating to explain our expectations like increments in second place is still a major breaking change. Every repository that follows this variance in semver must have an explanation in its README.md file. Related packages should follow this versioning schema to reduce ambiguities.

PROS

- Users have additional release information, which simplifies risk management on planning an upgrade
- HEPTAconnect is allowed to evolve into HEPTAconnect 2 or 3 while keeping the brand and project name without creating technical ambiguities

CONS

- Users can misread version string
- Releases need to have additional information

6.3.18 2022-03-02 - Final classes

Context

During development, we noticed that autocompletion suggested to extend from implementations that were not meant to be extended and result in confusion for API beginners. This is a signal for bad developer experience. In discussions with Macro "Ocramius", who is a well known defensive programmer, we took advise from him to use `final`. With that you will notice the `final` keyword more often, when e.g. using PSR-7 implementation from Tobias Nyholm `nyholm/psr7`, final has been recently shifted to a PHP doc comment.

Decision

We add `final` keyword to everything, when it is not breaking extensibility. As most of our solutions are based on strategy patterns and have contracts and interfaces, we can rely on these to define our API without exposing the implementations. We prefer the keyword over the PHP doc comment as the keyword is already present on language level without further extensions needed. DTOs may also be final when they implement the `Heptacom\HeptaConnect\Dataset\Base\Contract\AttachmentAwareInterface`.

Consequences

We need to evaluate every class to be final. To support this we provide an internal phpstan test case and pay attention to its hints.

PROS

- We can be sure our implementations are not reused, when we don't expect it
- We have more contracts and interfaces to keep flexibility and extensibility

CONS

- We can not mock every implementation anymore and might need to rewrite implementations
- We can not have implementations depending on each other by inheritance like we do at a few spots

6.3.19 2022-06-12 - Type safe class strings

Context

PHP and the PHP community are striving further towards static analysis. More and more tools arise next to the movement of the php-src developers adding language features to improve type safe programming. Tools like phpstan and psalm add PHPDoc comments based features like type templating/generics. There are also typed class strings. One can add comments to basic strings and indicate that these strings are references to fully-qualified class names. We make use of this class-string feature as we have references for types at several places. For example: we reference the types of entities to store data transfer relations between portal nodes. The only issue is, that this class-string feature is solely powered by PHPDoc comments. There are no validations during runtime. Without these validations the code is only valid in theory but can fail at real life usage. Real life usage includes scenarios like references to classes that do not exist anymore due to refactoring.

Decision

We need validation at runtime. Therefore, we need a replacement for class-string type hints. Although string is a very simple type, we will add a complex meaning to it. To have similar features at runtime like we have during static analysis, we need multiple classes. The following features must be implemented:

- Strings encapsulated in objects must behave like before and keep their original initial value
- Valid references mean, that the referenced class can be loaded
- String objects, that reference a class, do not necessarily have to reference an existing class
- String objects, that reference a class and claim to be valid, must perform validation on creation so its usage always ensures a valid reference
- String objects, that reference a class and claim to be of a certain type, must perform validation on creation so its usage always ensures a valid reference

Consequences

With different string validations at hand, various classes will follow different expectations towards validation of class strings. Loading data from the storage layer must assume invalid historical data and therefore must not invoke validation when loading the data. In contrast, business logic, that expects certain classes to be available for further processing, may use more strictly validating string classes.

PROS

- Class string references can now be validated at runtime

CONS

- More decisions have to be made, when to use which degree of validation

6.3.20 2022-10-06 - Filesystem abstraction with stream wrapper

Context

From the release of Shopware 5.1 (2015) via the release of Shopware 6.0 developer preview up to the latest version of Shopware 6.4 (2023), which is the latest released version as of writing, Flysystem is used as abstraction layer to the filesystem for extensions. We started developing HEPTAconnect within the Shopware ecommerce framework, so we followed the same filesystem abstraction guidelines. With the dependency within Shopware and within HEPTAconnect core to Flysystem v1 we can easily swap the filesystem storage from local disk to memory, AWS S3, Azure Blob Storage, SFTP. This simplifies administrative tasks to set up auto-scaling app servers accessing a network storage while also running the same code on a small development setup on a local machine. Flysystem v1 has been completely overhauled with the releases v2 and v3. As we depend on Shopware integrations we can't use a different Flysystem version. With the limitation to Flysystem v1 we can only support frameworks, that also support Flysystem v1. This excludes e.g. Laravel 9.

Decision

We deprecate and remove later the dependency on `league/flysystem`. As replacement, we use the even older interface, that did not receive breaking changes in our lifetime of using it: stream wrappers. It is possible to wrap Flysystem with a stream wrapper, so you can keep integrations, that make use of it, without forcing them to switch. It is possible to wrap stream wrapper with a Flysystem adapter, so you can keep portals, that make use of it, without forcing them to switch. We need to rewrite portals and integrations regarding their file access. There is a deprecation release before the removal of Flysystem so one can migrate step by step.

Consequences

- Portals, that use Flysystem need to be rewritten to access streams, files and the filesystem in a different way

PROS

- Integrates easier into other frameworks
- Use more native PHP methods, that will work with more libraries, that are not compatible with Flysystem

CONS

- Stream wrappers are more difficult to debug as the stack trace due to a PHP script calling a PHP method, that internally calls a different user-provided PHP class with a different set of arguments

HOW TO MIGRATE

File listing

before

```
/** @var \League\Flysystem\FilesystemInterface $filesystem */
$files = $filesystem->listContents('', true);
$paths = [];

foreach ($files as $file) {
    $paths[] = $file['path'];
}
```

after

```
/** @var \Heptacom\HeptaConnect\Portal\Base\File\Filesystem\Contract\FilesystemInterface $filesystem */
$fileIterator = new RecursiveIteratorIterator(new RecursiveDirectoryIterator($filesystem->toStoragePath('/')));
$paths = [];

/** @var SplFileInfo $file */
foreach ($fileIterator as $file) {
    $paths[] = $file->getPath();
}
```

Reading file content

before

```
/** @var \League\Flysystem\FilesystemInterface $filesystem */
$content = $filesystem->read('foobar.txt');
```

after

```
/** @var \Heptacom\HeptaConnect\Portal\Base\File\Filesystem\Contract\FilesystemInterface $filesystem */
$content = file_get_contents($filesystem->toStoragePath('foobar.txt'));
```

Writing file content

before

```
/** @var \League\Flysystem\FilesystemInterface $filesystem */
$filesystem->put('foobar.txt', 'Hello world');
```

after

```
/** @var \Heptacom\HeptaConnect\Portal\Base\File\Filesystem\Contract\FilesystemInterface $filesystem */
file_put_contents($filesystem->toStoragePath('foobar.txt'), 'Hello world');
```

Moving files

before

```
/** @var \League\Flysystem\FilesystemInterface $filesystem */
$filesystem->rename('foobar.txt', 'gizmo.txt');
```

after

```
/** @var \Heptacom\HeptaConnect\Portal\Base\File\Filesystem\Contract\FilesystemInterface $filesystem */
rename($filesystem->toStoragePath('foobar.txt'), $filesystem->toStoragePath('gizmo.txt'));
```

Deleting files

before

```
/** @var \League\Flysystem\FilesystemInterface $filesystem */
$filesystem->delete('foobar.txt');
```

after

```
/** @var \Heptacom\HeptaConnect\Portal\Base\File\Filesystem\Contract\FilesystemInterface $filesystem */
unlink($filesystem->toStoragePath('foobar.txt'));
```

6.4 Glossary

There are several types of classes or entities referenced throughout this documentation. To have a uniform understanding of their meanings they are listed here with a short definition and explanation.

6.4.1 Flow components

Explorer

An `Explorer` reads ids from the source and publishes them for emission. This is suitable for initial object discovery in the data source and successive data transfer via emissions.

Emitter

An `Emitter` reads data from the endpoint or data storage of a `PortalNode`, prepares the data in a structured form and then emits these structs. When it is asked to read data, a collection of `Mappings` is passed to its `emit` method. It is the `Emitters` job to connect to its data source and read the data (identified by the passed `Mappings`). The data should then be structured as a collection of `MappedDatasetEntityStructs` and returned or yielded.

Direct Emitter

An `Explorer` that does the work like an `Emitter` as it reads and yields complete objects instead of ids from the source and release them for emission. This is suitable for faster object transfer or transfer that is triggered only by the source portal.

Receiver

A `Receiver` receives a collection of `DatasetEntities` and writes the data to the endpoint or data storage of a `PortalNode`. When it is asked to write data, it traverses over the given collection, writes the data and retrieves an external identifier from the endpoint of the `PortalNode`. This identifier is then set in the given mapping and the collection of mappings is returned or yielded.

6.4.2 Portal

A `Portal` is the implementation of an endpoint to connect it via HEPTAconnect. When you want to provide connectivity for an external API or some other form of data storage, you implement a portal. So a portal is just a name for the composition of code (e.g. a composer package) that is necessary for HEPTAconnect to communicate with an endpoint.

PortalNode

A `Portal` is not the connection to an endpoint but the implementation of an endpoint. A `Portal` can then be configured with customizable fields. These fields may hold information like API-URLs, user credentials, file locations and so on. A configured `Portal` that is ready to communicate to an endpoint or data storage is called a `PortalNode`. A single `Portal` can potentially be used for many `PortalNodes`.

PortalRegistry

The `PortalRegistry` is provided by HEPTAconnect and can be used as a factory for `PortalNodes`. When a component has an identifier of a `PortalNode` and needs the corresponding instance to interact with it, this service should be used to retrieve the instance.

Bridge

The `Bridge` implements the core functionality in a certain environment by providing services for behaviours of the core that are dependent on the runtime of the surrounding application. As HEPTAconnect is environment agnostic it is not specified by default which database server, ORM, message broker, request cycle manager, request routing or file storage is in use.

Publisher

The `Publisher` is a central service that can be accessed by a `Bridge` to create `Mappings` for new entities. Publishing means, you target one specific object inside one specific `PortalNode` and have HEPTAconnect create a `Mapping` for it. The `Publisher` will prepare and schedule the freshly created `Mapping` for the `Emitter`. This happens asynchronously, so a `Publisher` will not take up a lot of computing time and it can be called during a web request with minimal performance impact.

Morpher

A `Morpher` is a special form of `PortalNode`. `Morphers` can receive various data types and store the entities temporarily. When certain conditions are met, the `Morpher` triggers its own `Emitter` to emit processed data. This could be used to collect different aspects of an entity and resolve dependencies. A `Morpher` could e.g. collect orders, addresses and customers and keep the data to itself until every sub-entity of the order has been received (a. k. a. all dependencies are resolved). After that the `Morpher` will emit a compound `DatasetEntity` with all the necessary data.

Packer

A `Packer` is a class that supports `Flow components` like `Direct Emitter` and `Emitter` packing API specific data into `DatasetEntities`. This naming has been really helpful in the past to find the right entrypoint when extending other portals. There is no interface or contract to follow.

Unpacker

An `Unpacker` is a class that supports `Flow components` like `Receiver` unpacking `DatasetEntities` into portal API specific payloads. This naming has been really helpful in the past to find the right entrypoint when extending other portals. There is no interface or contract to follow.

6.4.3 Dataset

A `Dataset` is a collection of common data structs that various `Portals` can rely on. There are different `Datasets` for different use cases and even some compound `Datasets` (e.g. `ecommerce`) that consist of multiple smaller `Datasets` (e.g. `physical-location`). `Datasets` are required by `Portals` to have a shared understanding of data and to establish communication between them.

DatasetEntity

A single entity in a `Dataset` is called a `DatasetEntity`. They are used to have a common data structure to pass objects from one `Portal` to another. Effectively a `Portal` does not need to know other `Portals` but simply work with `DatasetEntities` that other `Portals` also work with. This way any two `Portals` that share support for common `Datasets` can be connected.

6.4.4 Mapping

A `Mapping` is used to identify an entity in a `PortalNode`. It has an external identifier that points to the foreign entity, a `PortalNode` identifier that points to the `PortalNode` and a `MappingNode`. A mapping can also exist without an external identifier when the goal is to describe the connection between a `DatasetEntity` and a `PortalNode` before the foreign entity exists in the `PortalNode`. In practise this is used with `Receivers` when the foreign entity is yet to be created. HEPTAconnect will prepare a `Mapping` with the `PortalNode` identifier and a `MappingNode` but it will leave the external identifier empty. During a reception taking place in a `Receiver` the entities receive primary keys after they've been sent to the portal's API. After the `Receiver` finished its reception, any assigned primary key is passed to the management storage, which will store these external identifiers as mapping.

MappingNode

A `MappingNode` is used to associate various `Mappings` for different `PortalNodes` with each other. While one `Mapping` only points to a single foreign entity in a `PortalNode`, this is not enough to connect entities of different `PortalNodes` with each other. Every `Mapping` must have exactly one `MappingNode`, while one `MappingNode` can have multiple `Mappings`.

6.4.5 Identity Redirect

An identity redirect has two mapping representations where one mapping points to another mapping with the aspect, that these mappings do not have to exist in the storage. It is used to bypass the singularity aspect of a mapping node and allows to connect multiple mappings on one portal node to a single mapping on a different portal node.

6.4.6 Router

The `Router` is a central point in the data flow between different `PortalNodes`. When an `Emitter` emits a collection of `DatasetEntities`, the `Router` will search for matching `Routes` with the corresponding `PortalNode` as source. It will then pass the collection of `DatasetEntities` to every `PortalNode` that is specified as target in these `Routes`.

Route

A `Route` defines a direction for data to flow from one `PortalNode` to another. After setting up various `PortalNodes` it is necessary to create some `Routes`. A `Route` has a source, a target and a data type.

6.4.7 Storage

HEPTAconnect requires a form of storage in order to be functional. The storage is used to keep track of mappings, configurations and other data that is relevant to the system. All access to a storage provider is abstracted in the storage base and the core only relies on these interfaces.

Keys

The storage provider alone has data sovereignty over the keys that are used to persist entities in the data storage. `Keys` can be obtained by a factory that is provided by the storage provider. A `Key` is a small data structure that is a valid identifier in its origin storage (e.g. an auto-incremented integer or a UUIDv4). The existence of a `Key` itself guarantees its validity.

6.5 License

Dual licensed under the GNU Affero General Public License v3.0 (the "License") and proprietary license; you may not use this project except in compliance with the License. You may obtain a copy of the AGPL License at https://spdx.org/licenses/AGPL-3.0-or-later.html. Contact us on our website for further information about proprietary usage.

```
                GNU AFFERO GENERAL PUBLIC LICENSE
                   Version 3, 19 November 2007
```

Copyright (C) 2007 Free Software Foundation, Inc. https://fsf.org/ Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

```
                        Preamble
```

The GNU Affero General Public License is a free, copyleft license for software and other kinds of works, specifically designed to ensure cooperation with the community in the case of network server software.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, our General Public Licenses are intended to guarantee your freedom to share and change all versions of a program--to make sure it remains free software for all its users.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

Developers that use our General Public Licenses protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License which gives you legal permission to copy, distribute and/or modify the software.

A secondary benefit of defending all users' freedom is that improvements made in alternate versions of the program, if they receive widespread use, become available for other developers to incorporate. Many developers of free software are heartened and encouraged by the resulting cooperation. However, in the case of software used on network servers, this result may fail to come about. The GNU General Public License permits making a modified version and letting the public access it on a server without ever releasing its source code to the public.

The GNU Affero General Public License is designed specifically to ensure that, in such cases, the modified source code becomes available to the community. It requires the operator of a network server to provide the source code of the modified version running there to the users of that server. Therefore, public use of a modified version, on a publicly accessible server, gives the public access to the source code of the modified version.

An older license, called the Affero General Public License and published by Affero, was designed to accomplish similar goals. This is a different license, not a version of the Affero GPL, but Affero has released a new version of the Affero GPL which permits relicensing under this license.

The precise terms and conditions for copying, distribution and modification follow.

```
                    TERMS AND CONDITIONS
```

1. Definitions.

   "This License" refers to version 3 of the GNU Affero General Public License.

   "Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

   "The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

   To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

1. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

1. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

1. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

1. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

```
a) The work must carry prominent notices stating that you modified
it, and giving a relevant date.

b) The work must carry prominent notices stating that it is
released under this License and any conditions added under section
7.  This requirement modifies the requirement in section 4 to
"keep intact all notices".

c) You must license the entire work, as a whole, under this
License to anyone who comes into possession of a copy.  This
License will therefore apply, along with any applicable section 7
additional terms, to the whole of the work, and all its parts,
regardless of how they are packaged.  This License gives no
permission to license the work in any other way, but it does not
invalidate such permission if you have separately received it.

d) If the work has interactive user interfaces, each must display
Appropriate Legal Notices; however, if the Program has interactive
interfaces that do not display Appropriate Legal Notices, your
work need not make them do so.
```

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

1. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

```
a) Convey the object code in, or embodied in, a physical product
(including a physical distribution medium), accompanied by the
Corresponding Source fixed on a durable physical medium
customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product
(including a physical distribution medium), accompanied by a
written offer, valid for at least three years and valid for as
long as you offer spare parts or customer support for that product
model, to give anyone who possesses the object code either (1) a
copy of the Corresponding Source for all the software in the
product that is covered by this License, on a durable physical
medium customarily used for software interchange, for a price no
more than your reasonable cost of physically performing this
conveying of source, or (2) access to copy the
Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the
written offer to provide the Corresponding Source.  This
alternative is allowed only occasionally and noncommercially, and
only if you received the object code with such an offer, in accord
```

```
with subsection 6b.

d) Convey the object code by offering access from a designated
place (gratis or for a charge), and offer equivalent access to the
Corresponding Source in the same way through the same place at no
further charge.  You need not require recipients to copy the
Corresponding Source along with the object code.  If the place to
copy the object code is a network server, the Corresponding Source
may be on a different server (operated by you or a third party)
that supports equivalent copying facilities, provided you maintain
clear directions next to the object code saying where to find the
Corresponding Source.  Regardless of what server hosts the
Corresponding Source, you remain obligated to ensure that it is
available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided
you inform other peers where the object code and Corresponding
Source of the work are being offered to the general public at no
charge under subsection 6d.
```

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

1. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

```
a) Disclaiming warranty or limiting liability differently from the
terms of sections 15 and 16 of this License; or

b) Requiring preservation of specified reasonable legal notices or
author attributions in that material or in the Appropriate Legal
Notices displayed by works containing it; or

c) Prohibiting misrepresentation of the origin of that material, or
requiring that modified versions of such material be marked in
reasonable ways as different from the original version; or

d) Limiting the use for publicity purposes of names of licensors or
authors of the material; or

e) Declining to grant rights under trademark law for use of some
trade names, trademarks, or service marks; or

f) Requiring indemnification of licensors and authors of that
material by anyone who conveys the material (or modified versions of
it) with contractual assumptions of liability to the recipient, for
any liability that these contractual assumptions directly impose on
those licensors and authors.
```

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

1. Termination.

   You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

   However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

   Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

   Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

1. Acceptance Not Required for Having Copies.

   You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

1. Automatic Licensing of Downstream Recipients.

   Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

1. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

1. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the

Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

1. Remote Network Interaction; Use with the GNU General Public License.

Notwithstanding any other provision of this License, if you modify the Program, your modified version must prominently offer all users interacting with it remotely through a computer network (if your version supports such interaction) an opportunity to receive the Corresponding Source of your version by providing access to the Corresponding Source from a network server at no charge, through some standard or customary means of facilitating copying of software. This Corresponding Source shall include the Corresponding Source for any work covered by version 3 of the GNU General Public License that is incorporated pursuant to the following paragraph.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the work with which it is combined will remain governed by version 3 of the GNU General Public License.

1. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU Affero General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU Affero General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU Affero General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU Affero General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

1. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

1. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

1. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

```
                 END OF TERMS AND CONDITIONS

        How to Apply These Terms to Your New Programs
```

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does.>
Copyright (C) <year>  <name of author>

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Affero General Public License as published
by the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License
along with this program.  If not, see <https://www.gnu.org/licenses/>.
```

Also add information on how to contact you by electronic and paper mail.

If your software can interact with users remotely through a computer network, you should also make sure that it provides a way for users to get its source. For example, if your program is a web application, its interface could display a "Source" link that leads users to an archive of the code. There are many ways you could offer source, and different solutions will be better for different programs; see section 13 for the specific requirements.

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU AGPL, see https://www.gnu.org/licenses/.

7. Release

7.1 Releases

Here is an overview of all public changelogs we supply. They contain helpful hints for upgrading your code. When you want to upgrade your code have a look at the integrator upgrade guide. For everything, that will be coming in the future, you can see on the "live roadmap" for the upcoming features.

7.1.1 Bridge Shopware Platform

This is the HEPTAconnect package to provide a runtime in a shopware platform project.

Read the Bridge Shopware Platform changelog

7.1.2 Core

This is the HEPTAconnect core package. Here are all processes and entrypoints combined.

Read the Core changelog

7.1.3 Dataset Base

This is a HEPTAconnect package to provide basic dataset structures like structs and collections. Any other dataset library has to use the classes to work with HEPTAconnect utilities.

Read the Dataset Base changelog

7.1.4 Dataset Ecommerce

This is the ecommerce dataset. It provides all common entities for the transfer of data between different ecommerce portals.

Read the Dataset Ecommerce changelog

7.1.5 Portal Base

This is a HEPTAconnect package that provides base structures for portals. Any other portal library has to use the classes to work with HEPTAconnect utilities.

Read the Portal Base changelog

7.1.6 Portal Local Shopware Platform

This is a HEPTAconnect package that allows to communicate multiple entity types with a Shopware 6 instance that also integrates HEPTAconnect.

Read the Portal Local Shopware Platform changelog

7.1.7 Package HTTP

This is a HEPTAconnect package to support flow components working with HTTP clients or act as HTTP server.

Read the Package HTTP changelog

7.1.8 Package Web Frontend

This is a HEPTAconnect package to build web frontends for portals and integrations.

Read the Package Web Frontend changelog

7.1.9 Package Shopware 6

This is a HEPTAconnect package all about communicating to Shopware 6 APIs. You can use it in combination with the Shopware 6 Portal.

Read the Package Shopware 6 changelog

7.1.10 Storage Base

This is a HEPTAconnect package that provides base structures for storage providers.

Read the Storage Base changelog

7.1.11 Storage Shopware DAL

This is a HEPTAconnect package that offers an implementation for the storage within shopware 6.

Read the Storage Shopware DAL changelog

7.1.12 [0.9.7.0] - 2024-02-10

Changed

- Command `heptaconnect:job:run` now accepts multiple values for argument `job-key` and runs the jobs simultaneously

7.1.13 [0.9.6.0] - 2024-01-03

Added

- Add interface `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Parallelization\LockStoreFactoryInterface` with corresponding implementation `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Parallelization\LockStoreFactory` to provide lock stores for parallelization

Fixed

- Fix service container when no database url is configured by catching connection errors and falling back to in-memory lock store in `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Parallelization\LockStoreFactory`

7.1.14 [0.9.5.0] - 2023-07-10

Added

- Add dependency `heptacom_heptaconnect.logger` to service `Heptacom\HeptaConnect\Core\Job\Handler\ExplorationHandler`
- Add dependency `heptacom_heptaconnect.logger` to service `Heptacom\HeptaConnect\Core\Job\Handler\EmissionHandler`

Changed

- Raise composer dependency constraint for `heptacom/heptaconnect-core`, `heptacom/heptaconnect-dataset-base`, `heptacom/heptaconnect-portal-base` and `heptacom/heptaconnect-storage-base` from `^0.9.4` to `^0.9.6`

7.1.15 [0.9.4.0] - 2023-05-27

Added

- Add service definition `Psr\Http\Message\StreamFactoryInterface.heptaconnect` factorized by `\Http\Discovery\Psr17FactoryDiscovery::findStreamFactory`
- Add service definition `Psr\Http\Message\UploadedFileFactoryInterface.heptaconnect` factorized by `\Http\Discovery\Psr17FactoryDiscovery::findUploadedFileFactory`
- Add service definition `Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\Psr7MessageMultiPartFormDataBuilderInterface` for class `Heptacom\HeptaConnect\Core\Web\Http\Psr7MessageMultiPartFormDataBuilder`
- Add alternative service id `Heptacom\HeptaConnect\Core\Component\Composer\Contract\PackageConfigurationLoaderInterface` for definition `Heptacom\HeptaConnect\Core\Component\Composer\PackageConfigurationLoader`
- Add alternative service id `Heptacom\HeptaConnect\Core\Emission\Contract\EmitServiceInterface` for definition `Heptacom\HeptaConnect\Core\Emission\EmitService`
- Add alternative service id `Heptacom\HeptaConnect\Core\Exploration\Contract\ExplorationActorInterface` for definition `Heptacom\HeptaConnect\Core\Exploration\ExplorationActor`
- Add alternative service id `Heptacom\HeptaConnect\Core\Exploration\Contract\ExploreServiceInterface` for definition `Heptacom\HeptaConnect\Core\Exploration\ExploreService`
- Add alternative service id `Heptacom\HeptaConnect\Core\Exploration\Contract\ExplorerStackBuilderFactoryInterface` for definition `Heptacom\HeptaConnect\Core\Exploration\ExplorerStackBuilderFactory`
- Add alternative service id `Heptacom\HeptaConnect\Core\Portal\Contract\PortalFactoryContract` for definition `Heptacom\HeptaConnect\Core\Portal\PortalFactory`

- Add alternative service id `Heptacom\HeptaConnect\Core\Portal\Contract\PortalRegistryInterface` for definition `Heptacom\HeptaConnect\Core\Portal\PortalRegistry`
- Add alternative service id `Heptacom\HeptaConnect\Portal\Base\Parallelization\Contract\ResourceLockingContract` for definition `Heptacom\HeptaConnect\Core\Parallelization\ResourceLocking`
- Add alternative service id `Heptacom\HeptaConnect\Core\Reception\Contract\ReceiverStackBuilderFactoryInterface` for definition `Heptacom\HeptaConnect\Core\Reception\ReceiverStackBuilderFactory`
- Add alternative service id `Heptacom\HeptaConnect\Core\Reception\Contract\ReceiveServiceInterface` for definition `Heptacom\HeptaConnect\Core\Reception\ReceiveService`
- Add alternative service id `Heptacom\HeptaConnect\Core\Reception\Contract\ReceptionActorInterface` for definition `Heptacom\HeptaConnect\Core\Reception\ReceptionActor`
- Add alternative service id `Heptacom\HeptaConnect\Portal\Base\Support\Contract\EntityStatusContract` for definition `Heptacom\HeptaConnect\Core\Support\EntityStatus`

Changed

- Add dependency in `Heptacom\HeptaConnect\Core\Job\Contract\ExplorationHandlerInterface` on `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Job\JobFailActionInterface`
- Add dependency in `Heptacom\HeptaConnect\Core\Job\Contract\EmissionHandlerInterface` on `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Job\JobFailActionInterface`
- Add dependency in `Heptacom\HeptaConnect\Core\Job\Contract\ReceptionHandlerInterface` on `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Job\JobFailActionInterface`
- Add method call `setHttpHandleService` to service definition `Heptacom\HeptaConnect\Core\Portal\Contract\PortalStackServiceContainerBuilderInterface`
- Add dependency in `Heptacom\HeptaConnect\Core\Portal\Contract\PortalStackServiceContainerBuilderInterface` on `Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\Psr7MessageMultiPartFormDataBuilderInterface`
- Add dependency in `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Web\Http\HttpHandlerController` on `Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\Psr7MessageMultiPartFormDataBuilderInterface`
- Add dependency in `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Web\Http\HttpHandlerController` on `Psr\Http\Message\StreamFactoryInterface.heptaconnect`
- Add dependency in `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Web\Http\HttpHandlerController` on `Psr\Http\Message\UploadedFileFactoryInterface.heptaconnect`

7.1.16 [0.9.3.0] - 2023-03-04

Added

- Add option `time-limit` to command `heptaconnect:job:cleanup-finished` to limit the time the command is running measured in seconds
- Add service definition `Heptacom\HeptaConnect\Core\Web\Http\Formatter\Support\Contract\HeaderUtilityInterface` for class `\Heptacom\HeptaConnect\Core\Web\Http\Formatter\Support\HeaderUtility`
- Add service definition `Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\Psr7MessageCurlShellFormatterContract` for class `\Heptacom\HeptaConnect\Core\Web\Http\Formatter\Psr7MessageCurlShellFormatter`
- Add service definition `Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\Psr7MessageRawHttpFormatterContract` for class `\Heptacom\HeptaConnect\Core\Web\Http\Formatter\Psr7MessageRawHttpFormatter`
- Add service definition `Heptacom\HeptaConnect\Core\Web\Http\Dump\Contract\ServerRequestCycleDumpCheckerInterface` for class `\Heptacom\HeptaConnect\Core\Web\Http\Dump\SampleRateServerRequestCycleDumpChecker`
- Add service alias `Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\Psr7MessageFormatterContract` to set `Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\Psr7MessageRawHttpFormatterContract` as default implementation
- Implement `\Heptacom\HeptaConnect\Core\Bridge\File\HttpHandlerDumpPathProviderInterface` in `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\File\HttpHandlerDumpPathProvider`

- Add service definition `Heptacom\HeptaConnect\Core\Web\Http\Dump\Contract\ServerRequestCycleDumperInterface` for class `\Heptacom\HeptaConnect\Core\Web\Http\Dump\ServerRequestCycleDumper`
- Add dependency `Heptacom\HeptaConnect\Core\Web\Http\Dump\Contract\ServerRequestCycleDumpCheckerInterface` and `Heptacom\HeptaConnect\Core\Web\Http\Dump\Contract\ServerRequestCycleDumperInterface` to service `Heptacom\HeptaConnect\Core\Web\Http\Contract\HttpHandleServiceInterface`
- Add dependency `Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\Psr7MessageCurlShellFormatterContract` and `Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\Psr7MessageRawHttpFormatterContract` to service `Heptacom\HeptaConnect\Core\Portal\Contract\PortalStackServiceContainerBuilderInterface`
- Add service definition `Heptacom\HeptaConnect\Storage\Base\Contract\Action\IdentityRedirect\IdentityRedirectDeleteActionInterface` provided by `Heptacom\HeptaConnect\Storage\Base\Bridge\Contract\StorageFacadeInterface`
- Add service definition `Heptacom\HeptaConnect\Storage\Base\Contract\Action\IdentityRedirect\IdentityRedirectCreateActionInterface` provided by `Heptacom\HeptaConnect\Storage\Base\Bridge\Contract\StorageFacadeInterface`
- Add service definition `Heptacom\HeptaConnect\Storage\Base\Contract\Action\IdentityRedirect\IdentityRedirectOverviewActionInterface` provided by `Heptacom\HeptaConnect\Storage\Base\Bridge\Contract\StorageFacadeInterface`
- Add command `heptaconnect:identity-redirect:add` in service definition `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\IdentityRedirect\AddIdentityRedirect` to add an identity redirect
- Add command `heptaconnect:identity-redirect:remove` in service definition `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\IdentityRedirect\RemoveIdentityRedirect` to remove an identity redirect
- Add command `heptaconnect:identity-redirect:list` in service definition `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\IdentityRedirect\ListIdentityRedirects` to list identity redirects
- Add identity redirect into evaluation of `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\MappingNode\ListMappingNodeSiblings`

Changed

- Use count of deleted jobs as progress indicator in command `heptaconnect:job:cleanup-finished`
- Delete jobs, that have not been finished at the start of the command `heptaconnect:job:cleanup-finished`, but finished during the command run
- Remove Symfony, connection, proxy and transfer related header from requests handled in `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Web\Http\HttpHandlerController`
- Raise composer dependency constraint for `heptacom/heptaconnect-core`, `heptacom/heptaconnect-dataset-base`, `heptacom/heptaconnect-portal-base` and `heptacom/heptaconnect-storage-base` from `^0.9.3` to `^0.9.4`
- Raise composer dependency constraint for `heptacom/heptaconnect-storage-shopware-dal` from `^0.9` to `^0.9.1`

Fixed

- Ensure missing query parameters in the request's URI passed on to the HTTP handler in `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Web\Http\HttpHandlerController`
- Interpret `entity-type` option in `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\MappingNode\ListMappingNodeSiblings` as filter criteria for identities
- Show an empty result if first search did not find a mapping node to search for its siblings `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\MappingNode\ListMappingNodeSiblings`

7.1.17 [0.9.2.0] - 2022-11-26

Added

- Add composer dependency `kor3k/flysystem-stream-wrapper: ^1.0.11` to register flysystem filesystems to a stream wrapper

- Add service definition for implementation `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\File\PortalNodeFilesystemStreamProtocolProvider` described by `\Heptacom\HeptaConnect\Core\Bridge\File\PortalNodeFilesystemStreamProtocolProviderInterface` to provide stream wrapper protocol and register flysystem filesystems for portal nodes
- Add service definition `Heptacom\HeptaConnect\Core\Portal\File\Filesystem\Contract\FilesystemFactoryInterface` for class `\Heptacom\HeptaConnect\Core\Portal\File\Filesystem\FilesystemFactory`
- Add dependency `Heptacom\HeptaConnect\Core\Portal\File\Filesystem\Contract\FilesystemFactoryInterface` to service `Heptacom\HeptaConnect\Core\Portal\Contract\PortalStackServiceContainerBuilderInterface`
- Add command `heptaconnect:emit` to emit one or more entities
- Add composer suggestion `psy/psysh` for an interactive read–eval–print loop in the scope of a portal-node
- Add command `heptaconnect:repl` for an interactive read–eval–print loop in the scope of a portal-node

Fixed

- Change base filesystem for portal nodes in `Heptacom\HeptaConnect\Core\Storage\Filesystem\FilesystemFactory` from the Shopware bundle provided private filesystem to a custom prefixed filesystem based on the Shopware instance private filesystem to keep the same default directory but to support adapter access on the file system

7.1.18 [0.9.1.1] - 2022-10-03

Added

- Show progress-bar in command `heptaconnect:job:cleanup-finished`

Fixed

- Remove service `Shopware\Core\Framework\MessageQueue\Monitoring\MonitoringBusDecorator` from container as it has been renamed from `Shopware\Core\Framework\MessageQueue\MonitoringBusDecorator`.
- Fix command `heptaconnect:portal-node:status:list-topics` when there are no topics

7.1.19 [0.9.1.0] - 2022-07-19

Added

- Add service `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\FrameworkX\XAppFactoryInterface` to initialize a framework-x app. Requires optional dependency `clue/framework-x`.

7.1.20 [0.9.0.3] - 2022-06-08

Fixed

- Fix command `heptaconnect:portal-node:status:list-topics` by using the `Heptacom\HeptaConnect\Core\Portal\FlowComponentRegistry` from the portal container
- Fix command `heptaconnect:job:cleanup-finished` by using only the job-keys of the `Heptacom\HeptaConnect\Storage\Base\Action\Job\Listing\JobListFinishedResult` objects

7.1.21 [0.9.0.2] - 2022-04-27

Fixed

- Create lock tables `heptaconnect_core_reception_lock` and `heptaconnect_portal_node_resource_lock` manually as `Symfony\Component\Lock\Store\PdoStore` does not create them automatically for MySQL driver

7.1.22 [0.9.0.1] - 2022-04-19

Fixed

- Use different locking implementation to follow Shopware master-slave database setup warning in `\Shopware\Core\Profiling\Doctrine\DebugStack`

7.1.23 [0.9.0.0] - 2022-04-02

Added

- Add service definition `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Job\JobGetActionInterface` provided by `Heptacom\HeptaConnect\Storage\Base\Bridge\Contract\StorageFacadeInterface`
- Add service definition `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Job\JobCreateActionInterface` provided by `Heptacom\HeptaConnect\Storage\Base\Bridge\Contract\StorageFacadeInterface`
- Add service definition `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Job\JobListFinishedActionInterface` provided by `Heptacom\HeptaConnect\Storage\Base\Bridge\Contract\StorageFacadeInterface`
- Add service definition `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Job\JobDeleteActionInterface` provided by `Heptacom\HeptaConnect\Storage\Base\Bridge\Contract\StorageFacadeInterface`
- Add service definition `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Job\JobStartActionInterface` provided by `Heptacom\HeptaConnect\Storage\Base\Bridge\Contract\StorageFacadeInterface`
- Add service definition `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Job\JobFinishActionInterface` provided by `Heptacom\HeptaConnect\Storage\Base\Bridge\Contract\StorageFacadeInterface`
- Add service definition `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Job\JobFailActionInterface` provided by `Heptacom\HeptaConnect\Storage\Base\Bridge\Contract\StorageFacadeInterface`
- Add service definition `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Job\JobScheduleActionInterface` provided by `Heptacom\HeptaConnect\Storage\Base\Bridge\Contract\StorageFacadeInterface`
- Add service definition `Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNode\PortalNodeCreateActionInterface` provided by `Heptacom\HeptaConnect\Storage\Base\Bridge\Contract\StorageFacadeInterface`
- Add service definition `Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNode\PortalNodeDeleteActionInterface` provided by `Heptacom\HeptaConnect\Storage\Base\Bridge\Contract\StorageFacadeInterface`
- Add service definition `Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNode\PortalNodeListActionInterface` provided by `Heptacom\HeptaConnect\Storage\Base\Bridge\Contract\StorageFacadeInterface`
- Add service definition `Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNode\PortalNodeGetActionInterface` provided by `Heptacom\HeptaConnect\Storage\Base\Bridge\Contract\StorageFacadeInterface`
- Add service definition `Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNode\PortalNodeOverviewActionInterface` provided by `Heptacom\HeptaConnect\Storage\Base\Bridge\Contract\StorageFacadeInterface`
- Add command `heptaconnect:portal-node:extensions:activate` in service definition `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\PortalNode\Extension\ActivateExtension` to activate a portal extension on a portal node
- Add command `heptaconnect:portal-node:extensions:deactivate` in service definition `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\PortalNode\Extension\DeactivateExtension` to deactivate a portal extension on a portal node
- Add command `heptaconnect:portal-node:extensions:list` in service definition `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\PortalNode\Extension\ListExtensions` to list activity state of portal extensions on a portal node
- Add service definition `Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalExtension\PortalExtensionFindActionInterface` provided by `Heptacom\HeptaConnect\Storage\Base\Bridge\Contract\StorageFacadeInterface`
- Add service definition `Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalExtension\PortalExtensionActivateActionInterface` provided by `Heptacom\HeptaConnect\Storage\Base\Bridge\Contract\StorageFacadeInterface`

- Add service definition `Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalExtension\PortalExtensionDeactivateActionInterface` provided by `Heptacom\HeptaConnect\Storage\Base\Bridge\Contract\StorageFacadeInterface`
- Add option `--bidirectional` and its functionality to `heptaconnect:router:add-route` defined in class `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\Router\AddRoute` to automate creation of the route back
- Add service definition `\Heptacom\HeptaConnect\Core\Component\Logger\FlowComponentCodeOriginFinderLogger` for decorating `heptacom_heptaconnect.logger` to stringify flow component into human readable code origins in log messages
- Add command `heptaconnect:portal-node:list-flow-components` in service definition `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\PortalNode\ListFlowComponentsForPortalNode` to list all flow components for a given entity type, job type (by base class) and portal node
- Add service definition based upon class `\Heptacom\HeptaConnect\Core\Web\Http\HttpHandlerCodeOriginFinder` as `Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\HttpHandlerCodeOriginFinderInterface`
- Add service definition based upon class `\Heptacom\HeptaConnect\Core\Emission\EmitterCodeOriginFinder` as `Heptacom\HeptaConnect\Portal\Base\Emission\Contract\EmitterCodeOriginFinderInterface`
- Add service definition based upon class `\Heptacom\HeptaConnect\Core\Exploration\ExplorerCodeOriginFinder` as `Heptacom\HeptaConnect\Portal\Base\Exploration\Contract\ExplorerCodeOriginFinderInterface`
- Add service definition based upon class `\Heptacom\HeptaConnect\Core\Reception\ReceiverCodeOriginFinder` as `Heptacom\HeptaConnect\Portal\Base\Reception\Contract\ReceiverCodeOriginFinderInterface`
- Add service definition based upon class `\Heptacom\HeptaConnect\Core\StatusReporting\StatusReporterCodeOriginFinder` as `Heptacom\HeptaConnect\Portal\Base\StatusReporting\Contract\StatusReporterCodeOriginFinderInterface`
- Add service definition based upon class `\Heptacom\HeptaConnect\Storage\ShopwareDal\Bridge\StorageFacade` as `Heptacom\HeptaConnect\Storage\Base\Bridge\Contract\StorageFacadeInterface` that is used to create all storage based service
- Add service definition `Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNodeConfiguration\PortalNodeConfigurationGetActionInterface` provided by `Heptacom\HeptaConnect\Storage\Base\Bridge\Contract\StorageFacadeInterface`
- Add service definition `Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNodeConfiguration\PortalNodeConfigurationSetActionInterface` provided by `Heptacom\HeptaConnect\Storage\Base\Bridge\Contract\StorageFacadeInterface`
- Add service definition `Heptacom\HeptaConnect\Core\Component\Logger\ExceptionCodeLogger` for decorating `heptacom_heptaconnect.logger` to add exception codes in log messages
- Add service definition `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Identity\IdentityMapActionInterface` provided by `Heptacom\HeptaConnect\Storage\Base\Bridge\Contract\StorageFacadeInterface`
- Add service definition `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Identity\IdentityPersistActionInterface` provided by `Heptacom\HeptaConnect\Storage\Base\Bridge\Contract\StorageFacadeInterface`
- Add service definition `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Identity\IdentityOverviewActionInterface` provided by `Heptacom\HeptaConnect\Storage\Base\Bridge\Contract\StorageFacadeInterface`
- Add service definition `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Identity\IdentityReflectActionInterface` provided by `Heptacom\HeptaConnect\Storage\Base\Bridge\Contract\StorageFacadeInterface`
- Add service definition `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\RouteDeleteActionInterface` provided by `Heptacom\HeptaConnect\Storage\Base\Bridge\Contract\StorageFacadeInterface`
- Add service definition `Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNodeStorage\PortalNodeStorageClearActionInterface` provided by `Heptacom\HeptaConnect\Storage\Base\Bridge\Contract\StorageFacadeInterface`
- Add service definition `Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNodeStorage\PortalNodeStorageDeleteActionInterface` provided by `Heptacom\HeptaConnect\Storage\Base\Bridge\Contract\StorageFacadeInterface`
- Add service definition `Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNodeStorage\PortalNodeStorageGetActionInterface` provided by `Heptacom\HeptaConnect\Storage\Base\Bridge\Contract\StorageFacadeInterface`

- Add service definition `Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNodeStorage\PortalNodeStorageListActionInterface` provided by `Heptacom\HeptaConnect\Storage\Base\Bridge\Contract\StorageFacadeInterface`
- Add service definition `Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNodeStorage\PortalNodeStorageSetActionInterface` provided by `Heptacom\HeptaConnect\Storage\Base\Bridge\Contract\StorageFacadeInterface`
- Add service definition `Heptacom\HeptaConnect\Storage\Base\Contract\Action\IdentityError\IdentityErrorCreateActionInterface` provided by `Heptacom\HeptaConnect\Storage\Base\Bridge\Contract\StorageFacadeInterface`
- Add command `heptaconnect:router:remove-route` in service definition `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\Router\RemoveRoute` to remove a route by id seen on `heptaconnect:router:list-routes`
- Implement `\Heptacom\HeptaConnect\Core\Bridge\File\FileContentsUrlProviderInterface` in `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\File\FileContentsUrlProvider`
- Implement `\Heptacom\HeptaConnect\Core\Bridge\File\FileRequestUrlProviderInterface` in `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\File\FileRequestUrlProvider`
- Add HTTP route `heptaconnect.file.request` in `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\File\FileReferenceController::request` to send a stored request of a file reference and pass the response through to the client
- Add HTTP route `heptaconnect.file.contents` in `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\File\FileReferenceController::contents` to read a normalized stream of a file reference and respond with its contents and an arbitrary mime type
- Add service definition `Heptacom\HeptaConnect\Portal\Base\File\FileReferenceResolverContract`
- Add service definition `Heptacom\HeptaConnect\Core\Storage\Contract\RequestStorageContract`
- Add service definition `Heptacom\HeptaConnect\Core\Storage\Normalizer\Psr7RequestDenormalizer`
- Add service definition `Heptacom\HeptaConnect\Core\Storage\Normalizer\Psr7RequestNormalizer`
- Add service definition `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Support\RequestContextHelper`
- Add service definition `Heptacom\HeptaConnect\Storage\Base\Contract\Action\FileReference\FileReferenceGetRequestActionInterface` provided by `Heptacom\HeptaConnect\Storage\Base\Bridge\Contract\StorageFacadeInterface`
- Add service definition `Heptacom\HeptaConnect\Storage\Base\Contract\Action\FileReference\FileReferencePersistRequestActionInterface` provided by `Heptacom\HeptaConnect\Storage\Base\Bridge\Contract\StorageFacadeInterface`
- Add service definition `Heptacom\HeptaConnect\Core\Bridge\File\FileContentsUrlProviderInterface`
- Add service definition `Heptacom\HeptaConnect\Core\Bridge\File\FileRequestUrlProviderInterface`
- Add service definition `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\File\FileReferenceController`
- Add class `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Support\RequestContextHelper` to scope a request context to a base URL
- Add service definition `Heptacom\HeptaConnect\Core\Configuration\PortalNodeConfigurationInstructionProcessor` with dependency onto `heptacom_heptaconnect.logger`, `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Support\AliasStorageKeyGenerator`, `Heptacom\HeptaConnect\Core\Portal\PortalRegistry` and all tagged services by tag `heptaconnect_core.portal_node_configuration.instruction_file_loader` tagged as `heptaconnect_core.portal_node_configuration.processor`
- Add service definition `Heptacom\HeptaConnect\Core\Configuration\PortalNodeConfigurationCacheProcessor` with dependency onto `cache.system` and `Heptacom\HeptaConnect\Storage\Base\Contract\StorageKeyGeneratorContract` tagged as `heptaconnect_core.portal_node_configuration.processor`
- Add service and definition `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Support\AliasValidator` to validate portal node aliases
- Add command `heptaconnect:portal-node:alias:find` in service definition `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\PortalNode\Alias\Find` to resolve alias to a portal node key
- Add command `heptaconnect:portal-node:alias:get` in service definition `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\PortalNode\Alias\Get` to get an alias by a portal node key

Changed

- Change dependency in `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\Job\CleanupFinished` from `Heptacom\HeptaConnect\Storage\Base\Contract\Repository\JobRepositoryContract` into `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Job\JobListFinishedActionInterface` and `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Job\JobDeleteActionInterface`
- Change dependency in `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\Job\Run` from `Heptacom\HeptaConnect\Storage\Base\Contract\Repository\JobRepositoryContract` and `Heptacom\HeptaConnect\Storage\Base\Contract\Repository\JobPayloadRepositoryContract` into `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Job\JobGetActionInterface`
- Change dependency in `Heptacom\HeptaConnect\Core\Flow\MessageQueueFlow\MessageHandler` from `Heptacom\HeptaConnect\Storage\Base\Contract\Repository\JobRepositoryContract` and `Heptacom\HeptaConnect\Storage\Base\Contract\Repository\JobPayloadRepositoryContract` into `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Job\JobGetActionInterface`
- Change dependency in `Heptacom\HeptaConnect\Core\Job\Contract\JobDispatcherContract` from `Heptacom\HeptaConnect\Storage\Base\Contract\Repository\JobRepositoryContract` into `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Job\JobStartActionInterface` and `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Job\JobFinishActionInterface`
- Change dependency in `Heptacom\HeptaConnect\Core\Job\Contract\ReceptionHandlerInterface` from `Heptacom\HeptaConnect\Storage\Base\Contract\Repository\JobRepositoryContract` into `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Job\JobStartActionInterface` and `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Job\JobFinishActionInterface`
- Change dependency in `Heptacom\HeptaConnect\Core\Job\Contract\ExplorationHandlerInterface` from `Heptacom\HeptaConnect\Storage\Base\Contract\Repository\JobRepositoryContract` into `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Job\JobStartActionInterface` and `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Job\JobFinishActionInterface`
- Switch dependency in `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\MappingNode\ListMappingNodeSiblings` from `Heptacom\HeptaConnect\Storage\Base\Contract\Repository\PortalNodeRepositoryContract` into `Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNode\PortalNodeListActionInterface`
- Switch dependency in `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\PortalNode\AddPortalNode` from `Heptacom\HeptaConnect\Storage\Base\Contract\Repository\PortalNodeRepositoryContract` into `Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNode\PortalNodeCreateActionInterface`
- Switch dependency in `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\PortalNode\ListPortalNodes` from `Heptacom\HeptaConnect\Storage\Base\Contract\Repository\PortalNodeRepositoryContract` into `Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNode\PortalNodeOverviewActionInterface`
- Switch dependency in `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\PortalNode\RemovePortalNode` from `Heptacom\HeptaConnect\Storage\Base\Contract\Repository\PortalNodeRepositoryContract` into `Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNode\PortalNodeDeleteActionInterface`
- Switch dependency in `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\Web\HttpHandler\ListHandlers` from `Heptacom\HeptaConnect\Storage\Base\Contract\Repository\PortalNodeRepositoryContract` into `Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNode\PortalNodeListActionInterface`
- Switch dependency in `Heptacom\HeptaConnect\Core\Portal\PortalRegistry` from `Heptacom\HeptaConnect\Storage\Base\Contract\Repository\PortalNodeRepositoryContract` into `Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNode\PortalNodeGetActionInterface`
- Remove argument `Heptacom\HeptaConnect\Portal\Base\Builder\FlowComponent` from service definition `Heptacom\HeptaConnect\Core\Portal\Contract\PortalStackServiceContainerBuilderInterface`
- Add dependency `Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalExtension\PortalExtensionFindActionInterface` to the service definition `Heptacom\HeptaConnect\Core\Portal\PortalRegistry`
- Change service id from `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Listing\ReceptionRouteListActionInterface` to `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\ReceptionRouteListActionInterface`
- Change service id from `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Overview\RouteOverviewActionInterface` to `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\RouteOverviewActionInterface`

- Change service id from `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Find\RouteFindActionInterface` to `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\RouteFindActionInterface`
- Change service id from `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Get\RouteGetActionInterface` to `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\RouteGetActionInterface`
- Change service id from `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Get\RouteCreateActionInterface` to `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\RouteCreateActionInterface`
- Change service id from `Heptacom\HeptaConnect\Storage\Base\Contract\Action\RouteCapability\Overview\RouteCapabilityOverviewActionInterface` to `Heptacom\HeptaConnect\Storage\Base\Contract\Action\RouteCapability\RouteCapabilityOverviewActionInterface`
- Change service id from `Heptacom\HeptaConnect\Storage\Base\Contract\Action\WebHttpHandlerConfiguration\Find\WebHttpHandlerConfigurationFindActionInterface` to `Heptacom\HeptaConnect\Storage\Base\Contract\Action\WebHttpHandlerConfiguration\WebHttpHandlerConfigurationFindActionInterface`
- Change service id from `Heptacom\HeptaConnect\Storage\Base\Contract\Action\WebHttpHandlerConfiguration\Set\WebHttpHandlerConfigurationSetActionInterface` to `Heptacom\HeptaConnect\Storage\Base\Contract\Action\WebHttpHandlerConfiguration\WebHttpHandlerConfigurationSetActionInterface`
- Change behavior of command `heptaconnect:portal-node:config:get` to throw an exception when the output cannot be converted to JSON
- Change output of command `heptaconnect:portal-node:config:get` to not escape slashes in JSON
- Change output of command `heptaconnect:portal-node:status:report` to not escape slashes in JSON
- Change behavior of command `heptaconnect:http-handler:get-configuration` to throw an exception when the output cannot be converted to JSON
- Change output of command `heptaconnect:http-handler:get-configuration` to not escape slashes in JSON
- Change service id from `Heptacom\HeptaConnect\Core\Configuration\ConfigurationService` to `Heptacom\HeptaConnect\Core\Configuration\Contract\ConfigurationServiceInterface` to prioritize service interface as id
- Switch dependency in `Heptacom\HeptaConnect\Core\Configuration\Contract\ConfigurationServiceInterface` from `Heptacom\HeptaConnect\Storage\ShopwareDal\ConfigurationStorage` into `Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNodeConfiguration\PortalNodeConfigurationGetActionInterface` and `Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNodeConfiguration\PortalNodeConfigurationSetActionInterface`
- Switch dependency in `Heptacom\HeptaConnect\Core\Job\Contract\ReceptionHandlerInterface` from `Heptacom\HeptaConnect\Storage\Base\Contract\EntityMapperContract` into `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Identity\IdentityMapActionInterface`
- Switch dependency in `Heptacom\HeptaConnect\Core\Job\Contract\ReceptionHandlerInterface` from `Heptacom\HeptaConnect\Storage\Base\Contract\EntityReflectorContract` into `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Identity\IdentityReflectActionInterface`
- Switch dependency in `Heptacom\HeptaConnect\Core\Exploration\ExplorationActor` from `Heptacom\HeptaConnect\Core\Mapping\MappingService` into `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Identity\IdentityMapActionInterface`
- Switch dependency in `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\MappingNode\ListMappingNodes` from `Heptacom\HeptaConnect\Storage\Base\Contract\Repository\MappingRepositoryContract` into `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Identity\IdentityOverviewActionInterface`
- Switch dependency in `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\MappingNode\ListMappingNodeSiblings` from `Heptacom\HeptaConnect\Core\Portal\ComposerPortalLoader`, `Heptacom\HeptaConnect\Storage\Base\Contract\Repository\MappingNodeRepositoryContract`, `Heptacom\HeptaConnect\Storage\Base\Contract\Repository\MappingRepositoryContract`, `Heptacom\HeptaConnect\Core\Portal\PortalStackServiceContainerFactory` and `Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNode\PortalNodeListActionInterface` into `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Identity\IdentityOverviewActionInterface`

- Switch dependency in `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\MappingNode\MergeMappingNodes` from `Heptacom\HeptaConnect\Core\Mapping\Contract\MappingServiceInterface` into `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Identity\IdentityOverviewActionInterface`
- Switch dependency in `Heptacom\HeptaConnect\Core\Reception\PostProcessing\SaveMappingsPostProcessor` from `Heptacom\HeptaConnect\Storage\Base\MappingPersister\Contract\MappingPersisterContract` into `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Identity\IdentityPersistActionInterface`
- Switch implementation of `Heptacom\HeptaConnect\Core\Router\Router.lock_factory` from `Symfony\Component\Lock\Store\FlockStore` to `Symfony\Component\Lock\Store\PdoStore` to support horizontally scaled setups out of the box
- Switch implementation of `Heptacom\HeptaConnect\Storage\ShopwareDal\ResourceLockStorage.lock_factory` from `Symfony\Component\Lock\Store\FlockStore` to `Symfony\Component\Lock\Store\PdoStore` to support horizontally scaled setups out of the box
- Change id and implementation of `Heptacom\HeptaConnect\Storage\ShopwareDal\ResourceLockStorage` to `Heptacom\HeptaConnect\Core\Parallelization\Contract\ResourceLockStorageContract` implemented by `Heptacom\HeptaConnect\Core\Parallelization\ResourceLockStorage`
- Switch dependency in `Heptacom\HeptaConnect\Core\Portal\PortalStorageFactory` from `Heptacom\HeptaConnect\Storage\ShopwareDal\PortalStorage` into `Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNodeStorage\PortalNodeStorageClearActionInterface`, `Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNodeStorage\PortalNodeStorageDeleteActionInterface`, `Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNodeStorage\PortalNodeStorageListActionInterface`, `Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNodeStorage\PortalNodeStorageSetActionInterface` and `Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNodeStorage\PortalNodeStorageGetActionInterface`
- Switch dependency in `Heptacom\HeptaConnect\Core\Emission\EmitContextFactory` from `Heptacom\HeptaConnect\Storage\Core\Mapping\Contract\MappingServiceInterface` and `Heptacom\HeptaConnect\Storage\Base\Contract\Repository\MappingNodeRepositoryContract` to `Heptacom\HeptaConnect\Storage\Base\Contract\Action\IdentityError\IdentityErrorCreateActionInterface` as previous services are removed
- Switch dependency in `Heptacom\HeptaConnect\Core\Reception\PostProcessing\MarkAsFailedPostProcessor` from `Heptacom\HeptaConnect\Storage\Core\Mapping\Contract\MappingServiceInterface` to `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\IdentityError\IdentityErrorCreateActionInterface` as previous service is removed
- Remove argument `Heptacom\HeptaConnect\Storage\Base\Contract\Repository\MappingNodeRepositoryContract` from service definition `Heptacom\HeptaConnect\Core\Job\Contract\ReceptionHandlerInterface`
- Rename route `heptaconnect.http.handler` to `api.heptaconnect.http.handler`
- Change usage of deprecated `Heptacom\HeptaConnect\Portal\Base\Publication\Contract\PublisherInterface::publish` to `Heptacom\HeptaConnect\Portal\Base\Publication\Contract\PublisherInterface::publishBatch` in `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\Explore::execute` and `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Core\Mapping\PublisherDecorator::flushBuffer`
- Add final modifier to `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Core\Mapping\PublisherDecorator`, `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\DependencyInjection\CompilerPass\RemoveBusMonitoring`, `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\DependencyInjection\CompilerPass\RemoveEntityCache`, `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Profiling\Profiler`, `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Profiling\ProfilerFactory`, `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Support\CommandsPrintLogsSubscriber`, `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Web\Http\HttpHandlerUrlProvider` and `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Web\Http\HttpHandlerUrlProviderFactory` to ensure correct usage of implementation. Decoration by their interfaces or base classes is still possible
- Add argument `Heptacom\HeptaConnect\Core\Storage\Contract\RequestStorageContract` to service definition `Heptacom\HeptaConnect\Core\Portal\PortalStackServiceContainerBuilder`
- Add call to `\Heptacom\HeptaConnect\Core\Portal\PortalStackServiceContainerBuilder::setFileReferenceResolver` with argument `Heptacom\HeptaConnect\Portal\Base\File\FileReferenceResolverContract` to service definition `Heptacom\HeptaConnect\Core\Portal\PortalStackServiceContainerBuilder`

- Add argument `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Support\RequestContextHelper` to service definition `Heptacom\HeptaConnect\Core\Web\Http\Contract\HttpHandlerUrlProviderFactoryInterface`
- Switch dependency in `Heptacom\HeptaConnect\Core\Configuration\Contract\ConfigurationServiceInterface` from `cache.system`, `Heptacom\HeptaConnect\Storage\ShopwareDal\StorageKeyGenerator` and `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Support\AliasStorageKeyGenerator` to all tagged services by tag `heptaconnect_core.portal_node_configuration.processor`
- Change service id from `Heptacom\HeptaConnect\Storage\ShopwareDal\StorageKeyGenerator` to `Heptacom\HeptaConnect\Storage\Base\Contract\StorageKeyGeneratorContract` and provide by `Heptacom\HeptaConnect\Storage\Base\Bridge\Contract\StorageFacadeInterface`
- Add argument `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Support\AliasValidator` to service definition `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\PortalNode\AddPortalNode`
- Replace `heptaconnect:support:alias:list` in `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\Support\Alias\ListAliases` with new command `heptaconnect:portal-node:alias:overview` in service definition `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\PortalNode\Alias\Overview` to list all portal node keys and their aliases
- Replace `heptaconnect:support:alias:reset` in `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\Support\Alias\Reset` with new command `heptaconnect:portal-node:alias:reset` in service definition `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\PortalNode\Alias\Reset` to remove an alias from a portal node key
- Replace `heptaconnect:support:alias:set` in `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\Support\Alias\Set` with new command `heptaconnect:portal-node:alias:set` in service definition `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\PortalNode\Alias\Set` to set an alias to a portal node key
- Change implementation of `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Support\CommandsPrintLogsSubscriber` to support decoration of the logger. Replace argument `\Psr\Log\LoggerInterface` with `\Monolog\Handler\StreamHandler`.

Removed

- Remove command `heptaconnect:job:cleanup-payloads` and service `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\Job\CleanupPayloads` in favour of storages removing unused payloads with their jobs
- Remove service definition `Heptacom\HeptaConnect\Storage\Base\Contract\Repository\JobRepositoryContract`
- Remove service definition `Heptacom\HeptaConnect\Storage\Base\Contract\Repository\JobPayloadRepositoryContract`
- Remove service definition `Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\PortalNodeRepository` and its alias `Heptacom\HeptaConnect\Storage\Base\Contract\Repository\PortalNodeRepositoryContract`
- Remove unused service `Heptacom\HeptaConnect\Portal\Base\Builder\FlowComponent`
- Remove service definition `Heptacom\HeptaConnect\Storage\ShopwareDal\ConfigurationStorage` in favour of `Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNodeConfiguration\PortalNodeConfigurationGetActionInterface` and `Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNodeConfiguration\PortalNodeConfigurationSetActionInterface`
- Remove command `heptaconnect:cronjob:ensure-queue` and service `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\Cronjob\EnsureQueue` as the feature of cronjobs in its current implementation is removed
- Remove command `heptaconnect:cronjob:queue` and service `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\Cronjob\Queue` as the feature of cronjobs in its current implementation is removed
- Remove class and its service `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Messaging\Cronjob\CronjobRunHandler` and `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Messaging\Cronjob\CronjobRunMessageHandler` as the feature of cronjobs in its current implementation is removed
- Remove class `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Messaging\Cronjob\CronjobRunMessage` as the feature of cronjobs in its current implementation is removed
- Remove service `\Heptacom\HeptaConnect\Core\Cronjob\CronjobService`, `Heptacom\HeptaConnect\Core\Cronjob\CronjobContextFactory`, `Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\CronjobRepository`, `Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\CronjobRunRepository`, `Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Cronjob\CronjobDefinition` and

`Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Cronjob\CronjobRunDefinition` as the feature of cronjobs in its current implementation is removed

- Remove service `Heptacom\HeptaConnect\Storage\Base\Contract\EntityMapperContract` in favour of storage action `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Identity\IdentityMapActionInterface`
- Remove service `Heptacom\HeptaConnect\Storage\Base\MappingPersister\Contract\MappingPersisterContract` in favour of storage action `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Identity\IdentityPersistActionInterface`
- Remove service `Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\MappingRepository` and `Heptacom\HeptaConnect\Storage\Base\Contract\Repository\MappingRepositoryContract` in favour of storage action `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Identity\IdentityPersistActionInterface`, `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Identity\IdentityMapActionInterface` and `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Identity\IdentityOverviewActionInterface`
- Remove service `Heptacom\HeptaConnect\Core\Mapping\MappingService`
- Remove service `Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\MappingExceptionRepository`
- Remove service `Heptacom\HeptaConnect\Storage\ShopwareDal\EntityReflector`
- Remove service `Heptacom\HeptaConnect\Storage\Base\Contract\Repository\MappingNodeRepositoryContract`
- Remove service `Heptacom\HeptaConnect\Storage\ShopwareDal\DalAccess`
- Remove service `Heptacom\HeptaConnect\Storage\Base\MappingPersister\Contract\MappingPersisterContract`
- Remove service `Heptacom\HeptaConnect\Storage\ShopwareDal\EntityTypeAccessor`
- Remove composer dependency `dragonmantank/cron-expression`
- Integrate service definition `Heptacom\HeptaConnect\Core\Router\Router.lock_store` as anonymous service parameter directly into the definition of `Heptacom\HeptaConnect\Core\Router\Router.lock_factory`
- Integrate service definition `Heptacom\HeptaConnect\Storage\ShopwareDal\ResourceLockStorage.lock_store` as anonymous service parameter directly into the definition of `Heptacom\HeptaConnect\Core\Parallelization\Contract\ResourceLockStorageContract.lock_factory`
- Remove support for `symfony/lock: >=4 <5.2` so the `Symfony\Component\Lock\Store\PdoStore` will automatically create the lock tables
- Remove support for `shopware/core: 6.3.*`
- Remove service definition `Heptacom\HeptaConnect\Storage\ShopwareDal\PortalStorage` in favour of storage actions `Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNodeStorage\PortalNodeStorageClearActionInterface`, `Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNodeStorage\PortalNodeStorageDeleteActionInterface`, `Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNodeStorage\PortalNodeStorageListActionInterface`, `Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNodeStorage\PortalNodeStorageSetActionInterface` and `Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNodeStorage\PortalNodeStorageGetActionInterface`
- Remove service definitions for classes `\Heptacom\HeptaConnect\Storage\ShopwareDal\ContextFactory`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\EntityType\EntityTypeDefinition`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Mapping\MappingDefinition`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Mapping\MappingErrorMessageDefinition`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Mapping\MappingNodeDefinition`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\PortalNode\PortalNodeDefinition`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\PortalNode\PortalNodeStorageDefinition`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Job\JobDefinition`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Job\JobPayloadDefinition`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Job\JobTypeDefinition` and `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Route\RouteDefinition` as well as their generated services `heptaconnect_entity_type.repository`, `heptaconnect_mapping.repository`, `heptaconnect_mapping_error_message.repository`, `heptaconnect_mapping_node.repository`, `heptaconnect_portal_node.repository`, `heptaconnect_portal_node_storage.repository`, `heptaconnect_job.repository`, `heptaconnect_job_payload.repository`, `heptaconnect_job_type.repository` and `heptaconnect_route.repository` as DAL usage is removed in `heptacom/heptaconnect-storage-shopware-dal`
- Remove deprecated `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Core\Mapping\PublisherDecorator::publish` inherited by `Heptacom\HeptaConnect\Portal\Base\Publication\Contract\PublisherInterface::publish`

- Remove support for `doctrine/dbal: >=2.1 <2.11`
- Remove implementation `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Support\AliasStorageKeyGenerator` as portal node alias support is integrated into `heptacom/heptaconnect-core`
- Remove Shopware entity classes `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Content\KeyAlias\KeyAliasCollection`, `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Content\KeyAlias\KeyAliasDefinition` and `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Content\KeyAlias\KeyAliasEntity` for table `heptaconnect_bridge_key_alias`

Fixed

- Change hardcoded `prod` environment in `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\AbstractIntegration::getLifecycleContainer` to using the current one
- Add tag `console.command` to service definition of `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\Job\CleanupFinished` to make the command available
- Add tag `console.command` to service definition of `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\Job\Run` to make the command available

st changes

7.1.24 [0.8.1] - 2022-03-04

Fixed

- Add missing service tag for command `heptaconnect:job:run`
- Add missing service tag for command `heptaconnect:job:cleanup-finished`

7.1.25 [0.8.0] - 2021-11-22

Added

- Add command `heptaconnect:job:run` in service definition `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\Job\Run` to run jobs by key from the commandline
- Add command `heptaconnect:job:cleanup-finished` in service definition `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\Job\CleanupFinished` to remove finished jobs from the storage
- Add command `heptaconnect:job:cleanup-payloads` in service definition `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\Job\CleanupPayloads` to remove unused job data from the storage
- Add service definition based upon class `\Heptacom\HeptaConnect\Core\Storage\Contract\StreamPathContract`
- Add service definition based upon class `\Heptacom\HeptaConnect\Storage\ShopwareDal\Support\Query\QueryIterator`
- Add service definition based upon class `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Route\ReceptionRouteList` as `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Listing\ReceptionRouteListActionInterface`
- Add service definition based upon class `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Route\RouteOverview` as `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Overview\RouteOverviewActionInterface`
- Add service definition based upon class `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Route\RouteFind` as `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Find\RouteFindActionInterface`
- Add service definition based upon class `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Route\RouteGet` as `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Get\RouteGetActionInterface`
- Add service definition based upon class `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Route\RouteCreate` as `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Create\RouteCreateActionInterface`
- Add service definition based upon class `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\RouteCapability\RouteCapabilityOverview` as `Heptacom\HeptaConnect\Storage\Base\Contract\Action\RouteCapability\Overview\RouteCapabilityOverviewActionInterface`
- Add command `heptaconnect:router:list-capabilities` in service definition `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\Router\ListRouteCapabilities` to list available route capabilities
- Add service definition based upon class `\Heptacom\HeptaConnect\Storage\ShopwareDal\RouteCapabilityAccessor`
- Add column for route primary key and route capabilities to the output of `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\Router\ListRoutes` named `heptaconnect:router:list-routes`
- Add command `heptaconnect:http-handler:set-configuration` in service definition `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\Web\HttpHandler\Set` to set http handler configuration
- Add command `heptaconnect:http-handler:get-configuration` in service definition `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\Web\HttpHandler\Get` to read http handler configuration
- Add service definition based upon class `\Heptacom\HeptaConnect\Storage\ShopwareDal\WebHttpHandlerAccessor`
- Add service definition based upon class `\Heptacom\HeptaConnect\Storage\ShopwareDal\WebHttpHandlerPathAccessor`
- Add service definition based upon class `\Heptacom\HeptaConnect\Storage\ShopwareDal\WebHttpHandlerPathIdResolver`
- Add command `heptaconnect:config:base-url:get` in service definition `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\Config\GetBaseUrlCommand` to get base url for http handlers
- Add command `heptaconnect:config:base-url:set` in service definition `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\Config\SetBaseUrlCommand` to set base url for http handlers
- Add service definition `Psr\Http\Message\ResponseFactoryInterface.heptaconnect` factorized by `\Http\Discovery\Psr17FactoryDiscovery::findResponseFactory`

- Add service definition based upon class `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\WebHttpHandlerConfiguration\WebHttpHandlerConfigurationFind` as `Heptacom\HeptaConnect\Storage\Base\Contract\Action\WebHttpHandlerConfiguration\Find\WebHttpHandlerConfigurationFindActionInterface`
- Add service definition based upon class `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\WebHttpHandlerConfiguration\WebHttpHandlerConfigurationSet` as `Heptacom\HeptaConnect\Storage\Base\Contract\Action\WebHttpHandlerConfiguration\Set\WebHttpHandlerConfigurationSetActionInterface`
- Add service definition based upon class `\Heptacom\HeptaConnect\Core\Web\Http\HttpHandleContextFactory` as `Heptacom\HeptaConnect\Core\Web\Http\Contract\HttpHandleContextFactoryInterface`
- Add service definition based upon class `\Heptacom\HeptaConnect\Core\Web\Http\HttpHandlerStackBuilderFactory` as `Heptacom\HeptaConnect\Core\Web\Http\Contract\HttpHandlerStackBuilderFactoryInterface`
- Add service definition based upon class `\Heptacom\HeptaConnect\Core\Web\Http\HttpHandleService` as `Heptacom\HeptaConnect\Core\Web\Http\Contract\HttpHandleServiceInterface`
- Add service definition based upon class `\Heptacom\HeptaConnect\Core\Web\Http\HttpHandlingActor` as `Heptacom\HeptaConnect\Core\Web\Http\Contract\HttpHandlingActorInterface`
- Add service definition based upon class `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Web\Http\HttpHandlerUrlProviderFactory` as `Heptacom\HeptaConnect\Core\Web\Http\Contract\HttpHandlerUrlProviderFactoryInterface`
- Add service definition based upon class `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Web\Http\HttpHandlerController` and http handling implementation
- Add service definition based upon class `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Web\Http\HttpHostProviderContract` and implementation to simplify base URL configuration for integrators
- Add command `heptaconnect:http-handler:list-handlers` in service definition `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\Web\HttpHandler\ListHandlers` to list available HTTP handlers
- Add command `heptaconnect:portal-node:status:list-topics` in service definition `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\PortalNode\ListStatusReportTopics` to list all supported status topics

Changed

- Change service definition id from `Heptacom\HeptaConnect\Storage\ShopwareDal\DatasetEntityTypeAccessor` to `Heptacom\HeptaConnect\Storage\ShopwareDal\EntityTypeAccessor` and set new id for definitions of services `Heptacom\HeptaConnect\Storage\Base\Contract\EntityMapperContract`, `Heptacom\HeptaConnect\Storage\Base\Contract\Repository\JobRepositoryContract`, `Heptacom\HeptaConnect\Storage\Base\Contract\Repository\MappingNodeRepositoryContract`, `Heptacom\HeptaConnect\Storage\Base\Contract\Repository\RouteRepositoryContract`
- Change parameter name of `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Core\Mapping\PublisherDecorator::publish` from `$datasetEntityClassName` to `$entityType`
- Change name of service `heptaconnect_dataset_entity_type.repository.patched` to `heptaconnect_entity_type.repository.patched`
- Change `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\DatasetEntityType\DatasetEntityTypeDefinition` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\EntityType\EntityTypeDefinition`
- Change argument and variable names in `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\MappingNode\ListMappingNodes::configure`, `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\MappingNode\ListMappingNodes::execute` and `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\MappingNode\ListMappingNodeSiblings::configure`, `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\MappingNode\ListMappingNodeSiblings::execute`
- Add dependency `Heptacom\HeptaConnect\Storage\Base\Contract\Repository\JobRepositoryContract` to the service definition `Heptacom\HeptaConnect\Core\Job\Handler\EmissionHandler`
- Add dependency `Heptacom\HeptaConnect\Storage\Base\Contract\Repository\JobRepositoryContract` to the service definition `Heptacom\HeptaConnect\Core\Job\Handler\ExplorationHandler`
- Add dependency `Heptacom\HeptaConnect\Storage\Base\Contract\Repository\JobRepositoryContract` to the service definition `Heptacom\HeptaConnect\Core\Job\Handler\ReceptionHandler`
- Add dependency `cache.system` in the service definition `Heptacom\HeptaConnect\Core\Configuration\ConfigurationService`
- Add service definition `Heptacom\HeptaConnect\Core\Reception\PostProcessing\MarkAsFailedPostProcessor` with dependencies on `Heptacom\HeptaConnect\Core\Mapping\MappingService` and `heptacom_heptaconnect.logger`

- Add service definition `Heptacom\HeptaConnect\Core\Reception\PostProcessing\SaveMappingsPostProcessor` with dependencies on `Heptacom\HeptaConnect\Portal\Base\Support\Contract\DeepObjectIteratorContract` and `Heptacom\HeptaConnect\Storage\Base\MappingPersister\Contract\MappingPersisterContract`
- Add dependency to tagged services of tag `heptaconnect.postprocessor` to service definition `Heptacom\HeptaConnect\Core\Reception\Contract\ReceiveContextFactoryInterface`. The service that are tagged like this are `Heptacom\HeptaConnect\Core\Reception\PostProcessing\MarkAsFailedPostProcessor` and `Heptacom\HeptaConnect\Core\Reception\PostProcessing\SaveMappingsPostProcessor`
- Remove argument `Heptacom\HeptaConnect\Core\Mapping\MappingService` from service definition `Heptacom\HeptaConnect\Core\Reception\Contract\ReceiveContextFactoryInterface`
- Remove argument `Heptacom\HeptaConnect\Storage\Base\MappingPersister\Contract\MappingPersisterContract` from service definition `Heptacom\HeptaConnect\Core\Reception\ReceptionActor`
- Add dependency `Heptacom\HeptaConnect\Core\Storage\Contract\StreamPathContract` in the service definition `Heptacom\HeptaConnect\Core\Storage\Normalizer\StreamDenormalizer`
- Add dependency `Heptacom\HeptaConnect\Core\Storage\Contract\StreamPathContract` in the service definition `Heptacom\HeptaConnect\Core\Storage\Normalizer\StreamNormalizer`
- Add dependency `heptacom_heptaconnect.logger` in the service definition `Heptacom\HeptaConnect\Core\Storage\Normalizer\StreamNormalizer`
- Change dependency in `Heptacom\HeptaConnect\Core\Emission\EmissionActor` from `Heptacom\HeptaConnect\Storage\Base\Contract\Repository\RouteRepositoryContract` into `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Listing\ReceptionRouteListActionInterface`
- Change service definition id based upon class `Heptacom\HeptaConnect\Core\Emission\EmissionActor` to match its interface `Heptacom\HeptaConnect\Core\Emission\Contract\EmissionActorInterface`
- Change dependency in `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\Router\ListRoutes` from `Heptacom\HeptaConnect\Storage\Base\Contract\Repository\RouteRepositoryContract`, `Heptacom\HeptaConnect\Storage\Base\Contract\Repository\PortalNodeRepositoryContract` and `Heptacom\HeptaConnect\Core\Portal\PortalStackServiceContainerFactory` into `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Overview\RouteOverviewActionInterface`
- Add dependency `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Find\RouteFindActionInterface` in the service definition `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\Router\AddRoute`
- Change dependency in `Heptacom\HeptaConnect\Core\Job\Contract\ReceptionHandlerInterface` from `Heptacom\HeptaConnect\Storage\Base\Contract\Repository\RouteRepositoryContract` into `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Get\RouteGetActionInterface`
- Change dependency in `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\Router\AddRoute` from `Heptacom\HeptaConnect\Storage\Base\Contract\Repository\RouteRepositoryContract` and `Heptacom\HeptaConnect\Storage\Base\Contract\Repository\PortalNodeRepositoryContract` into `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Create\RouteCreateActionInterface` and `Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Get\RouteGetActionInterface`
- Change output from `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\Router\AddRoute` named `heptaconnect:router:add-route` to show all route information like `heptaconnect:router:list-routes`
- Add dependency `Heptacom\HeptaConnect\Core\Web\Http\Contract\HttpHandlerUrlProviderFactoryInterface` in the service definition `Heptacom\HeptaConnect\Core\Portal\Contract\PortalStackServiceContainerBuilderInterface`
- Change dependency in `Heptacom\HeptaConnect\Storage\ShopwareDal\EntityReflector` from `heptaconnect_mapping.repository.patched` to `heptaconnect_mapping.repository`
- Change dependency in `Heptacom\HeptaConnect\Storage\ShopwareDal\PortalStorage` from `heptaconnect_portal_node_storage.repository.patched` to `heptaconnect_portal_node_storage.repository`
- Change dependency in `Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\CronjobRepository` from `heptaconnect_cronjob.repository.patched` to `heptaconnect_cronjob.repository`
- Change dependency in `Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\CronjobRunRepository` from `heptaconnect_cronjob.repository.patched` to `heptaconnect_cronjob.repository`
- Change dependency in `Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\CronjobRunRepository` from `heptaconnect_cronjob_run.repository.patched` to `heptaconnect_cronjob_run.repository`
- Change dependency in `Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\MappingExceptionRepository` from `heptaconnect_mapping_error_message.repository.patched` to `heptaconnect_mapping_error_message.repository`

- Change dependency in `Heptacom\HeptaConnect\Storage\Base\Contract\Repository\MappingRepositoryContract` from `heptaconnect_mapping.repository.patched` to `heptaconnect_mapping.repository`
- Change dependency in `Heptacom\HeptaConnect\Storage\Base\Contract\Repository\PortalNodeRepositoryContract` from `heptaconnect_portal_node.repository.patched` to `heptaconnect_portal_node.repository`
- Change dependency in `Heptacom\HeptaConnect\Storage\ShopwareDal\StorageKeyGenerator` from `heptaconnect_mapping_node.repository.patched` to `heptaconnect_mapping_node.repository`
- Change dependency in `Heptacom\HeptaConnect\Storage\ShopwareDal\StorageKeyGenerator` from `heptaconnect_mapping.repository.patched` to `heptaconnect_mapping.repository`
- Change dependency in `Heptacom\HeptaConnect\Storage\Base\MappingPersister\Contract\MappingPersisterContract` from `heptaconnect_mapping.repository.patched` to `heptaconnect_mapping.repository`
- Change dependency in `Heptacom\HeptaConnect\Storage\Base\Contract\Repository\JobRepositoryContract` from `heptaconnect_job.repository.patched` to `heptaconnect_job.repository`
- Change dependency in `Heptacom\HeptaConnect\Storage\Base\Contract\Repository\JobRepositoryContract` from `heptaconnect_job_type.repository.patched` to `heptaconnect_job_type.repository`
- Change dependency in `Heptacom\HeptaConnect\Storage\Base\Contract\Repository\JobPayloadRepositoryContract` from `heptaconnect_job_payload.repository.patched` to `heptaconnect_job_payload.repository`
- Change dependency in `Heptacom\HeptaConnect\Storage\Base\Contract\Repository\MappingNodeRepositoryContract` from `heptaconnect_mapping_node.repository.patched` to `heptaconnect_mapping_node.repository`
- Change dependency in `Heptacom\HeptaConnect\Storage\Base\Contract\Repository\MappingNodeRepositoryContract` from `heptaconnect_mapping.repository.patched` to `heptaconnect_mapping.repository`
- Change dependency in `Heptacom\HeptaConnect\Storage\ShopwareDal\EntityTypeAccessor` from `heptaconnect_entity_type.repository.patched` to `heptaconnect_entity_type.repository`
- Move route annotation registration from `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Webhook\WebhookController` to `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Web\Http\HttpHandlerController`
- Change command name from `heptaconnect:portal-node:status` to `heptaconnect:portal-node:status:report`
- Change option from `--dataset-entity-class` (`-d`) to `--entity-type` (`-t`) in `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\MappingNode\ListMappingNodeSiblings` (`heptaconnect:mapping-node:siblings-list`)
- Add dependency `heptacom_heptaconnect.logger` in the service definition `Heptacom\HeptaConnect\Core\Reception\PostProcessing\SaveMappingsPostProcessor`
- Add dependency `heptacom_heptaconnect.logger` in the service definition `Heptacom\HeptaConnect\Core\Job\Contract\ReceptionHandlerInterface`

Removed

- Remove service definition `Heptacom\HeptaConnect\Storage\Base\Contract\Repository\RouteRepositoryContract`
- Remove service definition `Heptacom\HeptaConnect\Core\Webhook\WebhookContextFactory`
- Remove service definition `Heptacom\HeptaConnect\Core\Webhook\WebhookService`
- Remove service definition `Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Webhook\WebhookDefinition`
- Remove service definition `heptaconnect_webhook.repository.patched`
- Remove service definition `Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\WebhookRepository`
- Remove class and its service definition `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Webhook\WebhookController` in favour of `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Web\Http\HttpHandlerController`
- Remove class and its service definition `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Webhook\UrlProvider` in favour of `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Web\Http\HttpHandlerUrlProviderFactory`
- Remove patched entity repository services `heptaconnect_mapping_node.repository.patched`, `heptaconnect_mapping.repository.patched`, `heptaconnect_job.repository.patched`, `heptaconnect_job_type.repository.patched`, `heptaconnect_job_payload.repository.patched`, `heptaconnect_entity_type.repository.patched`, `heptaconnect_route.repository.patched`, `heptaconnect_portal_node_storage.repository.patched`, `heptaconnect_portal_node.repository.patched`, `heptaconnect_mapping_error_message.repository.patched`, `heptaconnect_cronjob_run.repository.patched` and `heptaconnect_cronjob.repository.patched`

- Remove support for `shopware/core: 6.2.*` and therefore the compatibility patching process with `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\PatchProvider\EntityRepository` and `\Heptacom\HeptaConnect\Bridge\ShopwarePlatform\PatchProvider\EntityRepositoryPatch587`

Fixed

- Change behaviour of command `heptaconnect:router:list-routes` in `Heptacom\HeptaConnect\Bridge\ShopwarePlatform\Command\Router\ListRoutes` to also list created routes that do not have supported flow components (anymore)

7.1.26 [0.7.0] - 2021-09-25

Added

- Add service definition `Heptacom\HeptaConnect\Storage\Base\MappingPersister\Contract\MappingPersisterContract`

Changed

- Add dependency `heptacom_heptaconnect.logger` to service definition `\Heptacom\HeptaConnect\Core\Portal\PortalStorageFactory`
- Change service definition id based upon class `\Heptacom\HeptaConnect\Core\Emission\EmitContextFactory` to match its interface `\Heptacom\HeptaConnect\Core\Emission\Contract\EmitContextFactoryInterface`
- Change service definition id based upon class `\Heptacom\HeptaConnect\Core\Job\Handler\EmissionHandler` to match its interface `\Heptacom\HeptaConnect\Core\Job\Contract\EmissionHandlerInterface`
- Change service definition id based upon class `\Heptacom\HeptaConnect\Core\Job\Handler\ExplorationHandler` to match its interface `\Heptacom\HeptaConnect\Core\Job\Contract\ExplorationHandlerInterface`
- Change service definition id based upon class `\Heptacom\HeptaConnect\Core\Job\Handler\ReceptionHandler` to match its interface `\Heptacom\HeptaConnect\Core\Job\Contract\ReceptionHandlerInterface`
- Change service definition id based upon class `\Heptacom\HeptaConnect\Core\Reception\ReceiveContextFactory` to match its interface `\Heptacom\HeptaConnect\Core\Reception\Contract\ReceiveContextFactoryInterface`
- Change service definition id based upon class `\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\MappingRepository` to match its contract `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\MappingRepositoryContract`
- Remove argument `Heptacom\HeptaConnect\Core\Mapping\MappingService` from service definition `Heptacom\HeptaConnect\Portal\Base\Flow\DirectEmission\DirectEmissionFlowContract`

7.1.27 [0.9.7.0] - 2024-02-10

Added

- Add exception code `1693671570` in `\Heptacom\HeptaConnect\Core\Portal\ServiceContainerCompilerPass\BuildDefinitionForFlowComponentRegistryCompilerPass::getServiceReference` when a flow component service is missing a source attribute on its tag
- Add exception code `1693698154` in `\Heptacom\HeptaConnect\Core\Portal\ServiceContainerCompilerPass\BuildDefinitionForFlowComponentRegistryCompilerPass::getSourcePackage` when a referenced flow component package is not found in known packages

Changed

- Sort flow components by priority, if the service definition tag has a priority attribute

Deprecated

- Deprecate parameter `$source` in method `\Heptacom\HeptaConnect\Core\Portal\FlowComponentRegistry::getExplorers`
- Deprecate parameter `$source` in method `\Heptacom\HeptaConnect\Core\Portal\FlowComponentRegistry::getEmitters`
- Deprecate parameter `$source` in method `\Heptacom\HeptaConnect\Core\Portal\FlowComponentRegistry::getReceivers`
- Deprecate parameter `$source` in method `\Heptacom\HeptaConnect\Core\Portal\FlowComponentRegistry::getStatusReporters`
- Deprecate parameter `$source` in method `\Heptacom\HeptaConnect\Core\Portal\FlowComponentRegistry::getWebHttpHandlers`
- Deprecate method `\Heptacom\HeptaConnect\Core\Portal\FlowComponentRegistry::getOrderedSources`

7.1.28 [0.9.6.0] - 2023-07-10

Added

- Add log message code `1686752874` when handling of job failed in `\Heptacom\HeptaConnect\Core\Job\Handler\EmissionHandler`
- Add log message code `1686752879` when handling of job failed in `\Heptacom\HeptaConnect\Core\Job\Handler\ExplorationHandler`
- Add log message code `1686752889` when handling of job failed in `\Heptacom\HeptaConnect\Core\Job\Handler\ReceptionHandler`

Fixed

- Fix a bug in `\Heptacom\HeptaConnect\Core\Storage\Filesystem\AbstractFilesystem` that occurred when adapters don't populate the `path` key in metadata.
- Fix a bug in `\Heptacom\HeptaConnect\Core\Web\Http\HttpKernel` that broke sub-requests when the request contains no `Cookie` header.
- Fix order of packages when building a portal-container in `\Heptacom\HeptaConnect\Core\Portal\PortalStackServiceContainerBuilder`. Packages can now access services of other packages in their service definition, if they list that package in `\Heptacom\HeptaConnect\Portal\Base\Portal\Contract\PackageContract::getAdditionalPackages`.
- Fix emission check in `\Heptacom\HeptaConnect\Core\Support\EntityStatus::isMappedByEmitter` by validating external id instead of foreign key

7.1.29 [0.9.5.0] - 2023-05-27

Added

- Add service `Heptacom\HeptaConnect\Portal\Base\Portal\PackageCollection` to portal-container, containing the portal, all portal-extensions and all packages involved in building the container
- Add service `Psr\Http\Message\ServerRequestFactoryInterface` to portal-container

- Add service `Psr\Http\Message\UploadedFileFactoryInterface` to portal-container
- Add service `Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\HttpKernelInterface` to portal-container to execute a `\Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\HttpHandlerStackInterface` from inside a portal
- Add implementation `\Heptacom\HeptaConnect\Core\Web\Http\HttpHandleContext::forward` for `\Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\HttpHandleContextInterface::forward`
- Add composer dependency `riverline/multipart-parser:^2.1` to support parsing body-data of `\Psr\Http\Message\ServerRequestInterface` in `\Heptacom\HeptaConnect\Core\Web\Http\HttpKernel`
- Add implementation `\Heptacom\HeptaConnect\Core\Web\Http\Psr7MessageMultiPartFormDataBuilder` for `Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\Psr7MessageMultiPartFormDataBuilderInterface` to build HTTP payloads for multipart messages
- Add exception code `1682806294` in `\Heptacom\HeptaConnect\Core\Web\Http\Psr7MessageMultiPartFormDataBuilder::build` when an input parameter is of an illegal type

Changed

- Use `\Psr\Http\Message\StreamInterface::__toString` instead of `\Psr\Http\Message\StreamInterface::getContents` to retrieve stream contents in `\Heptacom\HeptaConnect\Core\File\ResolvedReference\ResolvedContentsFileReference`, `\Heptacom\HeptaConnect\Core\File\ResolvedReference\ResolvedPublicUrlFileReference` and `\Heptacom\HeptaConnect\Core\File\ResolvedReference\ResolvedRequestFileReference`. This way, all stream contents are retrieved, regardless of the position of the stream pointer.
- Add dependency on `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Job\JobFailActionInterface` into `\Heptacom\HeptaConnect\Core\Job\Handler\ExplorationHandler`, `\Heptacom\HeptaConnect\Core\Job\Handler\EmissionHandler` and `\Heptacom\HeptaConnect\Core\Job\Handler\ReceptionHandler` to set job-states to `failed` in case of an error
- Allow handling of HTTP requests, even when no HTTP handler exists for the requested path. This means, middlewares for HTTP handlers will run for every request.
- Add argument `bool $isStackEmpty` to `\Heptacom\HeptaConnect\Core\Web\Http\Handler\HttpMiddlewareChainHandler` to indicate whether the related instance of `\Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\HttpHandlerStackInterface` is empty.
- Change log level of code `1636845086` from `critical` to `notice`

Fixed

- Remove a step in building a portal-container that would remove all services that extend `\Heptacom\HeptaConnect\Portal\Base\Portal\Contract\PackageContract`
- Catch exceptions when running jobs and setting the affected jobs to failed state. Also change behavior in `\Heptacom\HeptaConnect\Core\Job\Handler\ExplorationHandler`, `\Heptacom\HeptaConnect\Core\Job\Handler\EmissionHandler` and `\Heptacom\HeptaConnect\Core\Job\Handler\ReceptionHandler` to continue with the remaining jobs.

7.1.30 [0.9.4.0] - 2023-03-04

Added

- Add `\Heptacom\HeptaConnect\Core\Web\Http\Formatter\Support\HeaderUtility` described by `\Heptacom\HeptaConnect\Core\Web\Http\Formatter\Support\Contract\HeaderUtilityInterface` to work with PSR-7 message headers
- Add implementation `\Heptacom\HeptaConnect\Core\Web\Http\Formatter\Psr7MessageRawHttpFormatter` extending `\Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\Psr7MessageRawHttpFormatter` to provide raw HTTP message formatting
- Add implementation `\Heptacom\HeptaConnect\Core\Web\Http\Formatter\Psr7MessageCurlShellFormatter` extending `\Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\Psr7MessageCurlShellFormatterContract` to provide cURL shell command formatting
- Add exception code `1674950000` in `\Heptacom\HeptaConnect\Core\Web\Http\Formatter\Psr7MessageRawHttpFormatter::formatMessage` when the given message is neither a request nor a response

- Add exception code `1674950001` in `\Heptacom\HeptaConnect\Core\Web\Http\Formatter\Psr7MessageRawHttpFormatter::getFileExtension` when the given message is neither a request nor a response
- Add exception code `1674950002` in `\Heptacom\HeptaConnect\Core\Web\Http\Formatter\Psr7MessageCurlShellFormatter::formatMessage` when the given message is neither a request nor a response
- Add exception code `1674950003` in `\Heptacom\HeptaConnect\Core\Web\Http\Formatter\Psr7MessageCurlShellFormatter::getFileExtension` when the given message is neither a request nor a response
- Add interface `\Heptacom\HeptaConnect\Core\Bridge\File\HttpHandlerDumpPathProviderInterface`, that needs to be implemented by bridges and integrations, to return the path for placing HTTP handler dumps
- Add constant `\Heptacom\HeptaConnect\Core\Web\Http\Contract\HttpHandleServiceInterface::REQUEST_ATTRIBUTE_PREFIX` to identify all request attributes, that can be used as value holders for additional parameters attached to requests to the core layer
- Add constant `\Heptacom\HeptaConnect\Core\Web\Http\Contract\HttpHandleServiceInterface::REQUEST_ATTRIBUTE_ORIGINAL_REQUEST` as request attribute key holding an instance of `\Psr\Http\Message\ServerRequestInterface` of the original inbound HTTP request used for debugging purposes
- Add `\Heptacom\HeptaConnect\Core\Web\Http\Dump\ServerRequestCycleDumper` described by `\Heptacom\HeptaConnect\Core\Web\Http\Dump\Contract\ServerRequestCycleDumperInterface` to dump a request cycle in a way, that they can be associated, when accessing the dumps
- Add sample rate strategy implementation `\Heptacom\HeptaConnect\Core\Web\Http\Dump\SampleRateServerRequestCycleDumpChecker` for new interface `\Heptacom\HeptaConnect\Core\Web\Http\Dump\Contract\ServerRequestCycleDumpCheckerInterface` using configuration `dump-sample-rate` for HTTP handlers, which can be an integer between 0 and 100, that will be used to determine whether a request-cycle will be dumped. Use value 100 for a request-response dump on every request

Changed

- Add dependency on `\Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\Psr7MessageCurlShellFormatterContract` into `\Heptacom\HeptaConnect\Core\Portal\PortalStackServiceContainerBuilder` to provide service for raw HTTP message formatting
- Add dependency on `\Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\Psr7MessageRawHttpFormatterContract` into `\Heptacom\HeptaConnect\Core\Portal\PortalStackServiceContainerBuilder` to provide service for cURL shell command formatting
- Add dependency on `\Heptacom\HeptaConnect\Core\Web\Http\Dump\Contract\ServerRequestCycleDumpCheckerInterface` and `\Heptacom\HeptaConnect\Core\Web\Http\Dump\Contract\ServerRequestCycleDumperInterface` into `\Heptacom\HeptaConnect\Core\Web\Http\HttpHandleService` to dump requests and responses from HTTP handling

Deprecated

- Deprecate class `\Heptacom\HeptaConnect\Core\Portal\Exception\DelegatingLoaderLoadException`. Use `\Heptacom\HeptaConnect\Portal\Base\Portal\Exception\DelegatingLoaderLoadException` instead.

Fixed

- Fix issue in `\Heptacom\HeptaConnect\Core\Portal\PortalStackServiceContainerBuilder` when a composer package with a portal has multiple PSR-4 entries in its `composer.json`
- Fix container compile error when an excluded service has an automatic alias from its interface.

7.1.31 [0.9.3.0] - 2022-11-26

Added

- Add `\Heptacom\HeptaConnect\Core\File\Filesystem\StreamUriSchemePathConverter` described by `\Heptacom\HeptaConnect\Core\File\Filesystem\Contract\StreamUriSchemePathConverterInterface` to convert between URIs and paths when using paths as contextualized URIs
- Add exception code `1666942800` in `\Heptacom\HeptaConnect\Core\File\Filesystem\StreamUriSchemePathConverter::convertToUri` when the path is not a compatible URI

- Add exception code `1666942801` in `\Heptacom\HeptaConnect\Core\File\Filesystem\StreamUriSchemePathConverter::convertToUri` when the path already has a protocol
- Add exception code `1666942802` in `\Heptacom\HeptaConnect\Core\File\Filesystem\StreamUriSchemePathConverter::convertToUri` when the path has a port
- Add exception code `1666942803` in `\Heptacom\HeptaConnect\Core\File\Filesystem\StreamUriSchemePathConverter::convertToUri` when the path has query parameters
- Add exception code `1666942804` in `\Heptacom\HeptaConnect\Core\File\Filesystem\StreamUriSchemePathConverter::convertToUri` when the path has a URI fragment
- Add exception code `1666942810` in `\Heptacom\HeptaConnect\Core\File\Filesystem\StreamUriSchemePathConverter::convertToPath` when the URI is not a URI
- Add exception code `1666942811` in `\Heptacom\HeptaConnect\Core\File\Filesystem\StreamUriSchemePathConverter::convertToPath` when the URI has no host
- Add exception code `1666942812` in `\Heptacom\HeptaConnect\Core\File\Filesystem\StreamUriSchemePathConverter::convertToPath` when the URI has a port
- Add exception code `1666942813` in `\Heptacom\HeptaConnect\Core\File\Filesystem\StreamUriSchemePathConverter::convertToPath` when the URI has query parameters
- Add exception code `1666942814` in `\Heptacom\HeptaConnect\Core\File\Filesystem\StreamUriSchemePathConverter::convertToPath` when the URI has a URI fragment
- Add `\Heptacom\HeptaConnect\Core\Portal\File\Filesystem\Filesystem` as implementation of `\Heptacom\HeptaConnect\Portal\Base\File\Filesystem\Contract\FilesystemInterface` to provide a path conversion for portals
- Add interface `\Heptacom\HeptaConnect\Core\File\Filesystem\Contract\StreamWrapperInterface` to describe, what the PHP documentation describes as signatures for a class to use as a stream wrapper
- Add interface `\Heptacom\HeptaConnect\Core\Bridge\File\PortalNodeFilesystemStreamProtocolProviderInterface`, that needs to be implemented by bridges and integrations, to create portal node specific stream protocols
- Add `\Heptacom\HeptaConnect\Core\Portal\File\Filesystem\FilesystemFactory` described by `\Heptacom\HeptaConnect\Core\Portal\File\Filesystem\Contract\FilesystemFactoryInterface` to create portal node specific instances of `\Heptacom\HeptaConnect\Portal\Base\File\Filesystem\Contract\FilesystemInterface`
- Add stream wrapper implementation `\Heptacom\HeptaConnect\Core\File\Filesystem\RewritePathStreamWrapper` to rewrite requested paths to a new protocol while changing the protocol and the path itself

Changed

- Add `\Heptacom\HeptaConnect\Core\Storage\Filesystem\AbstractFilesystem::getConfig` to forward the decorated filesystem config
- Wrap result of `\Heptacom\HeptaConnect\Core\Storage\Filesystem\PrefixFilesystem::getAdapter` into an adaptor decorator of `\Heptacom\HeptaConnect\Core\Storage\Filesystem\PrefixAdapter` to ensure that adapter usage will apply same path rewrites like the filesystem itself
- Add dependency on `\Heptacom\HeptaConnect\Core\Portal\File\Filesystem\Contract\FilesystemFactoryInterface` into `\Heptacom\HeptaConnect\Core\Portal\PortalStackServiceContainerBuilder` to provide a `Heptacom\HeptaConnect\Portal\Base\File\Filesystem\Contract\FilesystemInterface` service for portal nodes

Fixed

- Changed return type of `\Heptacom\HeptaConnect\Core\Storage\Filesystem\AbstractFilesystem::getAdapter` from `\League\Flysystem\FilesystemInterface` to `\League\Flysystem\AdapterInterface` by returning the decorated filesystem adapter instead of the filesystem itself

7.1.32 [0.9.2.0] - 2022-10-16

Added

- Add `\Heptacom\HeptaConnect\Core\Web\Http\HttpMiddlewareClient` to execute a chain of `\Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\HttpClientMiddlewareInterface` services for outbound HTTP requests via `\Psr\Http\Client\ClientInterface` from a portal-node context.
- Add `\Heptacom\HeptaConnect\Core\Portal\ServiceContainerCompilerPass\AddHttpMiddlewareClientCompilerPass` to automatically tag services implementing `\Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\HttpClientMiddlewareInterface` with `heptaconnect.http.client.middleware`.
- Execute a chain of `\Psr\Http\Server\MiddlewareInterface` services for inbound HTTP request via `\Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\HttpHandlerContract`
- Add `\Heptacom\HeptaConnect\Core\Portal\ServiceContainerCompilerPass\AddHttpMiddlewareCollectorCompilerPass` to automatically tag services implementing `\Psr\Http\Server\MiddlewareInterface` with `heptaconnect.http.handler.middleware`.
- Add `\Heptacom\HeptaConnect\Core\Support\HttpMiddlewareCollector` as a service in the portal-node container. It is used to retrieve tagged middleware services from the container.
- Add `\Heptacom\HeptaConnect\Core\Web\Http\Handler\HttpMiddlewareChainHandler` and `\Heptacom\HeptaConnect\Core\Web\Http\HttpMiddlewareHandler` to wrap execution of middleware chain
- Add composer dependency `psr/http-server-handler: ^1.0` and `psr/http-server-middleware: ^1.0` to support PSR-15 middlewares for HTTP handlers
- Add exception code `1651338559` in `\Heptacom\HeptaConnect\Core\Portal\PortalStorage::list` when unpacking a single entry fails
- Add exception code `1651338621` in `\Heptacom\HeptaConnect\Core\Portal\PortalStorage` when denormalizing any stored value fails

Fixed

- Only load dev-packages from `composer.lock` file when dev-mode is active in `\Heptacom\HeptaConnect\Core\Component\Composer\PackageConfigurationLoader`
- Only check for dev-mode in `\Heptacom\HeptaConnect\Core\Component\Composer\PackageConfigurationLoader`, if the installed version of composer supports it.
- Skip broken entries in `\Heptacom\HeptaConnect\Core\Portal\PortalStorage::list` instead of returning an empty list

7.1.33 [0.9.1.1] - 2022-09-28

Added

- Load composer packages also from `require-dev` section of `composer.lock` file in `\Heptacom\HeptaConnect\Core\Component\Composer\PackageConfigurationLoader`

7.1.34 [0.9.1.0] - 2022-08-15

Changed

- Move decision of exclusion by class for automatically created portal node container services from `\Heptacom\HeptaConnect\Core\Portal\ServiceContainerCompilerPass\RemoveAutoPrototypedDefinitionsCompilerPass` into `\Heptacom\HeptaConnect\Portal\Base\Portal\Contract\PackageContract::getContainerExcludedClasses`

Fixed

- Fix reception of multiple entities with the same identity within a single batch in `\Heptacom\HeptaConnect\Core\Job\Handler\ReceptionHandler`
- Add fallback value for the reported topic in `\Heptacom\HeptaConnect\Core\StatusReporting\StatusReportingService::reportSingleTopic`

- Prevent parallelization lock from being released immediately after creating or checking it in `\Heptacom\HeptaConnect\Core\Parallelization\ResourceLockStorage`

7.1.35 [0.9.0.2] - 2022-04-23

Fixed

- Portal instances and portal extension instances are not shared across multiple portal node service containers anymore. If these instances are used stateful, portal node A can affect portal node B. All packages we provide have been checked negative against stateful usage of portal and portal extension instances.
- Portal extension stacks are now built for each portal node instead for each portal. This resulted in portal node service containers with active portal extension that have not been set active for the stack's portal node. It only occurs when more than one portal node service containers of the same portal is created in a single PHP process e.g. a message consumption process.
- Portal node configuration for preview portal nodes are now loaded statically again
- Portal node service container for preview portal nodes are now loaded statically again

7.1.36 [0.9.0.1] - 2022-04-19

Fixed

- Fix order of portals and portal extensions in `\Heptacom\HeptaConnect\Core\Portal\FlowComponentRegistry`
- Fix creating identity error messages in `\Heptacom\HeptaConnect\Core\Reception\PostProcessing\MarkAsFailedPostProcessor::handle`
- Fix a critical error when writing portal node configuration

7.1.37 [0.9.0.0] - 2022-04-02

Added

- Implement `\Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\HttpHandlerCodeOriginFinderInterface` in `\Heptacom\HeptaConnect\Core\Web\Http\HttpHandlerCodeOriginFinder`
- Add exception code `1637607699` in `\Heptacom\HeptaConnect\Core\Web\Http\HttpHandlerCodeOriginFinder::findOrigin` when HTTP handler is a short-notation HTTP handler and has no configured callback
- Add exception code `1637607700` in `\Heptacom\HeptaConnect\Core\Web\Http\HttpHandlerCodeOriginFinder::findOrigin` when HTTP handler class cannot be read via reflection
- Add exception code `1637607701` in `\Heptacom\HeptaConnect\Core\Web\Http\HttpHandlerCodeOriginFinder::findOrigin` when HTTP handler class does not belong to a physical file
- Implement `\Heptacom\HeptaConnect\Portal\Base\Emission\Contract\EmitterCodeOriginFinderInterface` in `\Heptacom\HeptaConnect\Core\Emission\EmitterCodeOriginFinder`
- Add exception code `1637607653` in `\Heptacom\HeptaConnect\Core\Emission\EmitterCodeOriginFinder::findOrigin` when emitter is a short-notation emitter and has no configured callback
- Add exception code `1637607654` in `\Heptacom\HeptaConnect\Core\Emission\EmitterCodeOriginFinder::findOrigin` when emitter class cannot be read via reflection
- Add exception code `1637607655` in `\Heptacom\HeptaConnect\Core\Emission\EmitterCodeOriginFinder::findOrigin` when emitter class does not belong to a physical file
- Implement `\Heptacom\HeptaConnect\Portal\Base\Exploration\Contract\ExplorerCodeOriginFinderInterface` in `\Heptacom\HeptaConnect\Core\Exploration\ExplorerCodeOriginFinder`
- Add exception code `1637421327` in `\Heptacom\HeptaConnect\Core\Exploration\ExplorerCodeOriginFinder::findOrigin` when explorer is a short-notation explorer and has no configured callback
- Add exception code `1637421328` in `\Heptacom\HeptaConnect\Core\Exploration\ExplorerCodeOriginFinder::findOrigin` when explorer class cannot be read via reflection
- Add exception code `1637421329` in `\Heptacom\HeptaConnect\Core\Exploration\ExplorerCodeOriginFinder::findOrigin` when explorer class does not belong to a physical file

- Implement `\Heptacom\HeptaConnect\Portal\Base\Reception\Contract\ReceiverCodeOriginFinderInterface` in `\Heptacom\HeptaConnect\Core\Reception\ReceiverCodeOriginFinder`
- Add exception code `1641079368` in `\Heptacom\HeptaConnect\Core\Reception\ReceiverCodeOriginFinder::findOrigin` when receiver is a short-notation receiver and has no configured callback
- Add exception code `1641079369` in `\Heptacom\HeptaConnect\Core\Reception\ReceiverCodeOriginFinder::findOrigin` when receiver class cannot be read via reflection
- Add exception code `1641079370` in `\Heptacom\HeptaConnect\Core\Reception\ReceiverCodeOriginFinder::findOrigin` when receiver class does not belong to a physical file
- Implement `\Heptacom\HeptaConnect\Portal\Base\StatusReporting\Contract\StatusReporterCodeOriginFinderInterface` in `\Heptacom\HeptaConnect\Core\StatusReporting\StatusReporterCodeOriginFinder`
- Add exception code `1641079371` in `\Heptacom\HeptaConnect\Core\StatusReporting\StatusReporterCodeOriginFinder::findOrigin` when status reporter is a short-notation status reporter and has no configured callback
- Add exception code `1641079372` in `\Heptacom\HeptaConnect\Core\StatusReporting\StatusReporterCodeOriginFinder::findOrigin` when status reporter class cannot be read via reflection
- Add exception code `1641079373` in `\Heptacom\HeptaConnect\Core\StatusReporting\StatusReporterCodeOriginFinder::findOrigin` when status reporter class does not belong to a physical file
- Add logger decorator `\Heptacom\HeptaConnect\Core\Component\Logger\FlowComponentCodeOriginFinderLogger` that replaces instances of `\Heptacom\HeptaConnect\Portal\Base\Emission\Contract\EmitterContract`, `\Heptacom\HeptaConnect\Portal\Base\Exploration\Contract\ExplorerContract`, `\Heptacom\HeptaConnect\Portal\Base\Reception\Contract\ReceiverContract`, `\Heptacom\HeptaConnect\Portal\Base\StatusReporting\Contract\StatusReporterContract` and `\Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\HttpHandlerContract` within the context with their code origin
- Add new service `Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\HttpClientContract` to portal node container as an alternative to `Psr\Http\Client\ClientInterface` with behaviour by configuration e.g. that can throw `\Heptacom\HeptaConnect\Portal\Base\Web\Http\Exception\HttpException` on certain status code
- Add class `\Heptacom\HeptaConnect\Core\Component\Logger\ExceptionCodeLogger` intended as a decorator to prepend the exception code to log messages if available
- Add log message code `1647396033` in `\Heptacom\HeptaConnect\Core\Flow\MessageQueueFlow\MessageHandler::handleJob` when jobs from message cannot be loaded
- Add log message code `1647396034` in `\Heptacom\HeptaConnect\Core\Flow\MessageQueueFlow\MessageHandler::handleJob` when jobs from message cannot be processed
- Add contract `\Heptacom\HeptaConnect\Core\Parallelization\Contract\ResourceLockStorageContract` migrated from `\Heptacom\HeptaConnect\Storage\Base\Contract\ResourceLockStorageContract`
- Add implementation `\Heptacom\HeptaConnect\Core\Parallelization\ResourceLockStorage` for `\Heptacom\HeptaConnect\Core\Parallelization\Contract\ResourceLockStorageContract` that depends on `symfony/lock` which is already required
- Add log message code `1646383738` in `\Heptacom\HeptaConnect\Core\Portal\PortalStorage::list` when reading portal node storage entries fails
- Add implementation `\Heptacom\HeptaConnect\Core\Portal\PreviewPortalNodeStorage` for the interface `\Heptacom\HeptaConnect\Portal\Base\Portal\Contract\PortalStorageInterface` to support interactions on `\Heptacom\HeptaConnect\Storage\Base\PreviewPortalNodeKey`
- Add interface `\Heptacom\HeptaConnect\Core\Bridge\File\FileContentsUrlProviderInterface` to provide public URLs for normalized streams
- Add interface `\Heptacom\HeptaConnect\Core\Bridge\File\FileRequestUrlProviderInterface` to prode public URLs for serialized requests
- Add class `\Heptacom\HeptaConnect\Core\File\FileReferenceFactory` to create file references from public URLs, request objects or file contents
- Add class `\Heptacom\HeptaConnect\Core\File\FileReferenceResolver` to resolve file references for read operations

- Add class `\Heptacom\HeptaConnect\Core\File\Reference\ContentsFileReference` as implementation of `\Heptacom\HeptaConnect\Dataset\Base\File\FileReferenceContract` that is created from file contents
- Add class `\Heptacom\HeptaConnect\Core\File\Reference\PublicUrlFileReference` as implementation of `\Heptacom\HeptaConnect\Dataset\Base\File\FileReferenceContract` that is created from a public URL
- Add class `\Heptacom\HeptaConnect\Core\File\Reference\RequestFileReference` as implementation of `\Heptacom\HeptaConnect\Dataset\Base\File\FileReferenceContract` that is created from a PSR-7 request object
- Add class `\Heptacom\HeptaConnect\Core\File\ResolvedReference\ResolvedContentsFileReference` as implementation of `\Heptacom\HeptaConnect\Portal\Base\File\ResolvedFileReferenceContract` for file references that were created from file contents
- Add class `\Heptacom\HeptaConnect\Core\File\ResolvedReference\ResolvedPublicUrlFileReference` as implementation of `\Heptacom\HeptaConnect\Portal\Base\File\ResolvedFileReferenceContract` for file references that were created from a public URL
- Add class `\Heptacom\HeptaConnect\Core\File\ResolvedReference\ResolvedRequestFileReference` as implementation of `\Heptacom\HeptaConnect\Portal\Base\File\ResolvedFileReferenceContract` for file references that were created from a PSR-7 request object
- Add class `\Heptacom\HeptaConnect\Core\Storage\Normalizer\Psr7RequestDenormalizer` to deserialize instances of `\Psr\Http\Message\RequestInterface`
- Add class `\Heptacom\HeptaConnect\Core\Storage\Normalizer\Psr7RequestNormalizer` to serialize instances of `\Psr\Http\Message\RequestInterface`
- Add contract `\Heptacom\HeptaConnect\Core\Storage\Contract\RequestStorageContract` with implementation in `\Heptacom\HeptaConnect\Core\Storage\RequestStorage` to persist and load instances of `\Psr\Http\Message\RequestInterface`
- Add exception code `1647788744` in `\Heptacom\HeptaConnect\Core\File\FileReferenceFactory::fromContents` when the NormalizationRegistry is missing a normalizer for streams
- Add exception code `1648315863` in `\Heptacom\HeptaConnect\Core\File\FileReferenceFactory::fromContents` when the normalizer is unable to serialize the given file contents
- Add exception code `1647788896` in `\Heptacom\HeptaConnect\Core\File\FileReferenceResolver::resolve` when the NormalizationRegistry is missing a denormalizer for streams
- Add exception code `1647789133` in `\Heptacom\HeptaConnect\Core\File\FileReferenceResolver::resolve` when the FileReference has an unsupported source
- Add exception code `1647789503` in `\Heptacom\HeptaConnect\Core\File\ResolvedReference\ResolvedContentsFileReference::getContents` when denormalizing a normalized stream fails
- Add exception code `1647789809` in `\Heptacom\HeptaConnect\Core\Storage\Normalizer\Psr7RequestNormalizer::normalize` when trying to normalize anything other than a request object
- Add exception code `1647790094` in `\Heptacom\HeptaConnect\Core\Storage\RequestStorage::load` when denormalizing a serialized request fails
- Add exception code `1647791094` in `\Heptacom\HeptaConnect\Core\Storage\RequestStorage::load` when a serialized request is not found
- Add exception code `1647791390` in `\Heptacom\HeptaConnect\Core\Storage\RequestStorage::persist` when persisting a serialized request fails
- Add `\Heptacom\HeptaConnect\Core\Bridge\PortalNode\Configuration\Contract\InstructionTokenContract` to define a contract for changing portal node configurations
- Add `\Heptacom\HeptaConnect\Core\Bridge\PortalNode\Configuration\ClosureInstructionToken` that changes portal node configuration by the given closure
- Add `\Heptacom\HeptaConnect\Core\Bridge\PortalNode\Configuration\PortalNodeConfigurationHelper` to generate closures for processing configuration sources like json files and environment variables
- Add exception code `1647801828` in return callable from `\Heptacom\HeptaConnect\Core\Bridge\PortalNode\Configuration\PortalNodeConfigurationHelper::ini` when the ini file can not be loaded and parsed

- Add exception code `1647801829` in return callable from `\Heptacom\HeptaConnect\Core\Bridge\PortalNode\Configuration\PortalNodeConfigurationHelper::json` when the JSON file can not be loaded and parsed
- Add `\Heptacom\HeptaConnect\Core\Bridge\PortalNode\Configuration\Config` to collect `\Heptacom\HeptaConnect\Core\Bridge\PortalNode\Configuration\Contract\InstructionTokenContract` in a short-notation manner
- Add `\Heptacom\HeptaConnect\Core\Bridge\PortalNode\Configuration\Contract\InstructionLoaderInterface` to identify services that provide `\Heptacom\HeptaConnect\Core\Bridge\PortalNode\Configuration\Contract\InstructionTokenContract`
- Add `\Heptacom\HeptaConnect\Core\Bridge\PortalNode\Configuration\InstructionFileLoader` to provide `\Heptacom\HeptaConnect\Core\Bridge\PortalNode\Configuration\Contract\InstructionTokenContract` using `\Heptacom\HeptaConnect\Core\Bridge\PortalNode\Configuration\Config`
- Add exception code `1645611612` in `\Heptacom\HeptaConnect\Core\Bridge\PortalNode\Configuration\InstructionFileLoader::loadInstructions` when referenced file can not be loaded
- Add `\Heptacom\HeptaConnect\Core\Configuration\PortalNodeConfigurationInstructionProcessor` as `\Heptacom\HeptaConnect\Core\Configuration\Contract\PortalNodeConfigurationProcessorInterface` to change portal node configuration by instructions from given `\Heptacom\HeptaConnect\Core\Bridge\PortalNode\Configuration\Contract\InstructionLoaderInterface` instances
- Add log message code `1647826121` in `\Heptacom\HeptaConnect\Core\Configuration\PortalNodeConfigurationInstructionProcessor` when an error happens during instruction loading

Changed

- Replace dependencies in `\Heptacom\HeptaConnect\Core\Flow\MessageQueueFlow\MessageHandler` from `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\JobRepositoryContract` and `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\JobPayloadRepositoryContract` to `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Job\JobGetActionInterface` to improve performance by batching job reading
- Replace dependencies in `\Heptacom\HeptaConnect\Core\Job\Handler\EmissionHandler` from `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\JobRepositoryContract` to `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Job\JobStartActionInterface` and `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Job\JobFinishActionInterface` to improve performance by batching job state changes
- Replace dependencies in `\Heptacom\HeptaConnect\Core\Job\Handler\ExplorationHandler` from `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\JobRepositoryContract` to `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Job\JobStartActionInterface` and `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Job\JobFinishActionInterface` to improve performance by batching job state changes
- Replace dependencies in `\Heptacom\HeptaConnect\Core\Job\Handler\ReceptionHandler` from `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\JobRepositoryContract` to `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Job\JobStartActionInterface` and `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Job\JobFinishActionInterface` to improve performance by batching job state changes
- Replace dependencies in `\Heptacom\HeptaConnect\Core\Job\JobDispatcher` from `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\JobRepositoryContract` and `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\JobPayloadRepositoryContract` to `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Job\JobCreateActionInterface` to improve performance by batching job insertion
- Switch storage access in `\Heptacom\HeptaConnect\Core\Portal\PortalRegistry` from `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\PortalNodeRepositoryContract` to `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNode\PortalNodeGetActionInterface`
- Use portal node container tags `heptaconnect.flow_component.status_reporter_source`, `heptaconnect.flow_component.emitter_source`, `heptaconnect.flow_component.explorer_source`, `heptaconnect.flow_component.receiver_source`, `heptaconnect.flow_component.web_http_handler_source` instead of

`heptaconnect.flow_component.emitter`, `heptaconnect.flow_component.emitter_decorator`, `heptaconnect.flow_component.explorer`, `heptaconnect.flow_component.explorer_decorator`, `heptaconnect.flow_component.receiver`, `heptaconnect.flow_component.receiver_decorator` and `heptaconnect.flow_component.web_http_handler` to collect flow component services

- Short-noted flow components by portals load on first flow component usage instead of on container building using `\Heptacom\HeptaConnect\Core\Portal\FlowComponentRegistry`
- Add dependency onto `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalExtension\PortalExtensionFindActionInterface` into `\Heptacom\HeptaConnect\Core\Portal\PortalRegistry` for loading portal extension availability
- Use instance of `\Heptacom\HeptaConnect\Portal\Base\Emission\Contract\EmitterContract` in log context instead of its class in the message in `\Heptacom\HeptaConnect\Core\Emission\EmitterStackBuilder` logger usage
- Use instance of `\Heptacom\HeptaConnect\Portal\Base\Exploration\Contract\ExplorerContract` in log context instead of its class in the message in `\Heptacom\HeptaConnect\Core\Exploration\ExplorerStackBuilder` logger usage
- Use instance of `\Heptacom\HeptaConnect\Portal\Base\Reception\Contract\ReceiverContract` in log context instead of its class in the message in `\Heptacom\HeptaConnect\Core\Reception\ReceiverStackBuilder` logger usage
- Use instance of `\Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\HttpHandlerContract` in log context instead of its class in the message in `\Heptacom\HeptaConnect\Core\Web\Http\HttpHandlerStackBuilder` logger usage
- Replace dependencies in `\Heptacom\HeptaConnect\Core\Configuration\ConfigurationService` from `\Heptacom\HeptaConnect\Storage\Base\Contract\ConfigurationStorageContract` to `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNodeConfiguration\PortalNodeConfigurationGetActionInterface` and `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNodeConfiguration\PortalNodeConfigurationSetActionInterface` to improve performance on reading and writing portal node configuration
- Replace dependencies in `\Heptacom\HeptaConnect\Core\Job\Handler\ReceptionHandler` from `\Heptacom\HeptaConnect\Storage\Base\Contract\EntityMapperContract` to `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Identity\IdentityMapActionInterface` as previous service is renamed
- Replace dependencies in `\Heptacom\HeptaConnect\Core\Job\Handler\ReceptionHandler` from `\Heptacom\HeptaConnect\Storage\Base\Contract\EntityReflectorContract` to `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Identity\IdentityReflectActionInterface` as previous service is renamed
- Replace dependencies in `\Heptacom\HeptaConnect\Core\Exploration\ExplorationActor` from `\Heptacom\HeptaConnect\Core\Mapping\Contract\MappingServiceInterface` to `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Identity\IdentityMapActionInterface`
- Replace dependencies in `\Heptacom\HeptaConnect\Core\Reception\PostProcessing\SaveMappingsPostProcessor` from `\Heptacom\HeptaConnect\Storage\Base\MappingPersister\Contract\MappingPersisterContract` to `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Identity\IdentityPersistActionInterface` as previous service is renamed
- Replace dependencies in `\Heptacom\HeptaConnect\Core\Portal\PortalStorageFactory` and `\Heptacom\HeptaConnect\Core\Portal\PortalStorage` from `\Heptacom\HeptaConnect\Storage\Base\Contract\PortalStorageContract` to `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNodeStorage\PortalNodeStorageClearActionInterface`, `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNodeStorage\PortalNodeStorageDeleteActionInterface`, `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNodeStorage\PortalNodeStorageGetActionInterface` and `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNodeStorage\PortalNodeStorageListActionInterface`, `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNodeStorage\PortalNodeStorageSetActionInterface`
- Replace dependencies in `\Heptacom\HeptaConnect\Core\Emission\EmitContext` from `\Heptacom\HeptaConnect\Storage\Core\Mapping\Contract\MappingServiceInterface` and `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\MappingNodeRepositoryContract` to `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\IdentityError\IdentityErrorCreateActionInterface` as previous services are removed
- Replace dependencies in `\Heptacom\HeptaConnect\Core\Emission\EmitContextFactory` from `\Heptacom\HeptaConnect\Storage\Core\Mapping\Contract\MappingServiceInterface` and `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\MappingNodeRepositoryContract` to `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\IdentityError\IdentityErrorCreateActionInterface` as previous services are removed

- Replace dependencies in `\Heptacom\HeptaConnect\Core\Reception\PostProcessing\MarkAsFailedPostProcessor` from `\Heptacom\HeptaConnect\Storage\Core\Mapping\Contract\MappingServiceInterface` to `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\IdentityError\IdentityErrorCreateActionInterface` as previous service is removed
- Split argument in `\Heptacom\HeptaConnect\Core\Reception\Contract\ReceiveServiceInterface::receive` of type `\Heptacom\HeptaConnect\Portal\Base\Mapping\TypedMappedDatasetEntityCollection` into `\Heptacom\HeptaConnect\Dataset\Base\TypedDatasetEntityCollection` and `\Heptacom\HeptaConnect\Portal\Base\StorageKey\Contract\PortalNodeKeyInterface` to state target portal clearly
- Extract caching of `\Heptacom\HeptaConnect\Core\Configuration\ConfigurationService` into new class `\Heptacom\HeptaConnect\Core\Configuration\PortalNodeConfigurationCacheProcessor` using the `\Heptacom\HeptaConnect\Core\Configuration\Contract\PortalNodeConfigurationProcessorInterface` interface
- Make classes final: `\Heptacom\HeptaConnect\Core\Component\Composer\PackageConfigurationLoader`, `\Heptacom\HeptaConnect\Core\Configuration\ConfigurationService`, `\Heptacom\HeptaConnect\Core\Emission\EmissionActor`, `\Heptacom\HeptaConnect\Core\Emission\EmitContext`, `\Heptacom\HeptaConnect\Core\Emission\EmitContextFactory`, `\Heptacom\HeptaConnect\Core\Emission\EmitService`, `\Heptacom\HeptaConnect\Core\Emission\EmitterStackBuilder`, `\Heptacom\HeptaConnect\Core\Emission\EmitterStackBuilderFactory`, `\Heptacom\HeptaConnect\Core\Exploration\DirectEmitter`, `\Heptacom\HeptaConnect\Core\Exploration\ExplorationActor`, `\Heptacom\HeptaConnect\Core\Exploration\ExploreContext`, `\Heptacom\HeptaConnect\Core\Exploration\ExploreContextFactory`, `\Heptacom\HeptaConnect\Core\Exploration\ExplorerStackBuilder`, `\Heptacom\HeptaConnect\Core\Exploration\ExplorerStackBuilderFactory`, `\Heptacom\HeptaConnect\Core\Exploration\ExploreService`, `\Heptacom\HeptaConnect\Core\Flow\DirectEmissionFlow\DirectEmissionFlow`, `\Heptacom\HeptaConnect\Core\Job\Handler\EmissionHandler`, `\Heptacom\HeptaConnect\Core\Job\Handler\ExplorationHandler`, `\Heptacom\HeptaConnect\Core\Job\Handler\ReceptionHandler`, `\Heptacom\HeptaConnect\Core\Job\DelegatingJobActor`, `\Heptacom\HeptaConnect\Core\Job\JobDispatcher`, `\Heptacom\HeptaConnect\Core\Mapping\MappingNodeStruct`, `\Heptacom\HeptaConnect\Core\Mapping\MappingStruct`, `\Heptacom\HeptaConnect\Core\Mapping\Publisher`, `\Heptacom\HeptaConnect\Core\Portal\ServiceContainerCompilerPass\AddPortalConfigurationBindingsCompilerPass`, `\Heptacom\HeptaConnect\Core\Portal\ServiceContainerCompilerPass\AllDefinitionDefaultsCompilerPass`, `\Heptacom\HeptaConnect\Core\Portal\ServiceContainerCompilerPass\BuildDefinitionForFlowComponentRegistryCompilerPass`, `\Heptacom\HeptaConnect\Core\Portal\ServiceContainerCompilerPass\RemoveAutoPrototypedDefinitionsCompilerPass`, `\Heptacom\HeptaConnect\Core\Portal\PortalConfiguration`, `\Heptacom\HeptaConnect\Core\Portal\PortalFactory`, `\Heptacom\HeptaConnect\Core\Portal\PortalRegistry`, `\Heptacom\HeptaConnect\Core\Portal\PortalStackServiceContainerBuilder`, `\Heptacom\HeptaConnect\Core\Portal\PortalStorage`, `\Heptacom\HeptaConnect\Core\Portal\PreviewPortalNodeStorage`, `\Heptacom\HeptaConnect\Core\Reception\PostProcessing\MarkAsFailedPostProcessor`, `\Heptacom\HeptaConnect\Core\Reception\PostProcessing\SaveMappingsPostProcessor`, `\Heptacom\HeptaConnect\Core\Reception\ReceiveContext`, `\Heptacom\HeptaConnect\Core\Reception\ReceiveContextFactory`, `\Heptacom\HeptaConnect\Core\Reception\ReceiverStackBuilder`, `\Heptacom\HeptaConnect\Core\Reception\ReceiverStackBuilderFactory`, `\Heptacom\HeptaConnect\Core\Reception\ReceiveService`, `\Heptacom\HeptaConnect\Core\Reception\ReceptionActor`, `\Heptacom\HeptaConnect\Core\StatusReporting\StatusReportingContext`, `\Heptacom\HeptaConnect\Core\StatusReporting\StatusReportingContextFactory`, `\Heptacom\HeptaConnect\Core\StatusReporting\StatusReportingService`, `\Heptacom\HeptaConnect\Core\Storage\Normalizer\ScalarDenormalizer`, `\Heptacom\HeptaConnect\Core\Storage\Normalizer\ScalarNormalizer`, `\Heptacom\HeptaConnect\Core\Storage\Normalizer\SerializableCompressDenormalizer`, `\Heptacom\HeptaConnect\Core\Storage\Normalizer\SerializableDenormalizer`, `\Heptacom\HeptaConnect\Core\Storage\Normalizer\SerializableNormalizer`, `\Heptacom\HeptaConnect\Core\Storage\Normalizer\StreamDenormalizer`, `\Heptacom\HeptaConnect\Core\Storage\Normalizer\StreamNormalizer`, `\Heptacom\HeptaConnect\Core\Storage\NormalizationRegistry`, `\Heptacom\HeptaConnect\Core\Support\EntityStatus`, `\Heptacom\HeptaConnect\Core\Web\Http\HttpHandleContext`, `\Heptacom\HeptaConnect\Core\Web\Http\HttpHandleContextFactory`, `\Heptacom\HeptaConnect\Core\Web\Http\HttpHandlerStackBuilder`, `\Heptacom\HeptaConnect\Core\Web\Http\HttpHandlerStackBuilderFactory`, `\Heptacom\HeptaConnect\Core\Web\Http\HttpHandleService` and `\Heptacom\HeptaConnect\Core\Web\Http\HttpHandlingActor`

Removed

- Remove separation of source flow components and decorator flow components in `\Heptacom\HeptaConnect\Core\Emission\EmitterStackBuilder`, `\Heptacom\HeptaConnect\Core\Exploration\ExplorerStackBuilder`,

`\Heptacom\HeptaConnect\Core\Reception\ReceiverStackBuilder` and `\Heptacom\HeptaConnect\Core\Web\Http\HttpHandlerStackBuilder`. First flow component in list is always the source

- Remove portal node container service ids `Heptacom\HeptaConnect\Portal\Base\Emission\EmitterCollection`, `Heptacom\HeptaConnect\Portal\Base\Emission\EmitterCollection.decorator`, `Heptacom\HeptaConnect\Portal\Base\Exploration\ExplorerCollection`, `Heptacom\HeptaConnect\Portal\Base\Exploration\ExplorerCollection.decorator`, `Heptacom\HeptaConnect\Portal\Base\StatusReporting\StatusReporterCollection`, `Heptacom\HeptaConnect\Portal\Base\Reception\ReceiverCollection`, `Heptacom\HeptaConnect\Portal\Base\Reception\ReceiverCollection.decorator`, `Heptacom\HeptaConnect\Portal\Base\Web\Http\HttpHandlerCollection` and `Heptacom\HeptaConnect\Portal\Base\Web\Http\HttpHandlerCollection.decorator` due to refactoring of flow component stack building
- Remove dependency on `\Heptacom\HeptaConnect\Portal\Base\Builder\FlowComponent` in `\Heptacom\HeptaConnect\Core\Portal\PortalStackServiceContainerBuilder`
- Remove classes `\Heptacom\HeptaConnect\Core\Cronjob\CronjobContext`, `\Heptacom\HeptaConnect\Core\Cronjob\CronjobContextFactory` and `\Heptacom\HeptaConnect\Core\Cronjob\CronjobService` as the feature of cronjobs in its current implementation is removed
- Remove composer dependency `dragonmantank/cron-expression`
- Remove unused implementation `\Heptacom\HeptaConnect\Core\Mapping\MappingService::get` of `\Heptacom\HeptaConnect\Core\Mapping\Contract\MappingServiceInterface::get`
- Remove unused implementation `\Heptacom\HeptaConnect\Core\Mapping\MappingService::save` of `\Heptacom\HeptaConnect\Core\Mapping\Contract\MappingServiceInterface::save`
- Remove unused implementation `\Heptacom\HeptaConnect\Core\Mapping\MappingService::reflect` of `\Heptacom\HeptaConnect\Core\Mapping\Contract\MappingServiceInterface::reflect`
- Remove unused implementation `\Heptacom\HeptaConnect\Core\Mapping\MappingService::addException` of `\Heptacom\HeptaConnect\Core\Mapping\Contract\MappingServiceInterface::addException`
- Remove `\Heptacom\HeptaConnect\Core\Mapping\Contract\MappingServiceInterface::getListByExternalIds` in favour of `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Identity\IdentityMapActionInterface::map`
- Remove `\Heptacom\HeptaConnect\Core\Mapping\Contract\MappingServiceInterface::merge`, `\Heptacom\HeptaConnect\Core\Mapping\Exception\MappingNodeAreUnmergableException` and `\Heptacom\HeptaConnect\Core\Mapping\Exception\MappingNodeNotCreatedException` in favour of `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Identity\IdentityOverviewActionInterface` and `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Identity\IdentityPersistActionInterface`
- Remove log message code `1631563639`, `1631563699`, `1631565446` and `1631565376` from `\Heptacom\HeptaConnect\Core\Portal\PortalStorage`
- Remove deprecated methods `\Heptacom\HeptaConnect\Core\Portal\PortalStorage::canSet` and `\Heptacom\HeptaConnect\Core\Portal\PortalStorage::canGet`
- Remove unused `\Heptacom\HeptaConnect\Core\Router\CumulativeMappingException`
- Remove dependency on `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\MappingNodeRepositoryContract` in `\Heptacom\HeptaConnect\Core\Job\Handler\ReceptionHandler`
- Remove deprecated method `Heptacom\HeptaConnect\Core\Mapping\Publisher::publish` inherited from `Heptacom\HeptaConnect\Portal\Base\Publication\Contract\PublisherInterface::publish`

Fixed

- Portal node extensions can supply source flow components for data types that have not been introduced by the decorated portal
- All aliases in the dependency-injection container for portals are now public. This enables injection of aliased services in short-notation flow-components.

st changes

7.1.38 [0.8.6] - 2022-03-07

Fixed

- Prevent leak of `\Heptacom\HeptaConnect\Portal\Base\Reception\Support\PostProcessorDataBag` into subsequent iterations of `\Heptacom\HeptaConnect\Core\Reception\ReceptionActor::performReception`. Every entry of `\Heptacom\HeptaConnect\Core\Reception\PostProcessing\MarkAsFailedData` is now only handled once.

7.1.39 [0.8.5] - 2021-12-28

Fixed

- Change composer dependency `bentools/iterable-functions: >=1 <2` to `bentools/iterable-functions: >=1.4 <2` to ensure availability of `\iterable_map` in a lowest-dependency-version installation
- Change composer dependency `composer/composer: >=1` to `composer/composer: >=1.9` to ensure correct composer project and library parsing in a lowest-dependency-version installation
- Change composer dependency `php-http/discovery: ^1.0` to `php-http/discovery: ^1.11` to ensure availability of `\Http\Discovery\Psr17FactoryDiscovery` and `\Http\Discovery\Psr17FactoryDiscovery::findUriFactory` in a lowest-dependency-version installation
- Add composer dependency `symfony/event-dispatcher-contracts: >=1.1` to ensure availability of `\Symfony\Contracts\EventDispatcher\Event` in a lowest-dependency-version installation
- Change composer dependency `symfony/polyfill-php80: >=1.15` to `symfony/polyfill-php80: >=1.16` to ensure availability of `\str_starts_with` a php 7.4 and lowest-dependency-version installation
- Amend signature of `\Heptacom\HeptaConnect\Core\Storage\Normalizer\ScalarDenormalizer::denormalize`, `\Heptacom\HeptaConnect\Core\Storage\Normalizer\ScalarDenormalizer::supportsDenormalization`, `\Heptacom\HeptaConnect\Core\Storage\Normalizer\ScalarNormalizer::normalize`, `\Heptacom\HeptaConnect\Core\Storage\Normalizer\ScalarNormalizer::supportsNormalization`, `\Heptacom\HeptaConnect\Core\Storage\Normalizer\SerializableCompressDenormalizer::denormalize`, `\Heptacom\HeptaConnect\Core\Storage\Normalizer\SerializableCompressDenormalizer::supportsDenormalization`, `\Heptacom\HeptaConnect\Core\Storage\Normalizer\SerializableCompressNormalizer::normalize`, `\Heptacom\HeptaConnect\Core\Storage\Normalizer\SerializableDenormalizer::denormalize`, `\Heptacom\HeptaConnect\Core\Storage\Normalizer\SerializableDenormalizer::supportsDenormalization`, `\Heptacom\HeptaConnect\Core\Storage\Normalizer\SerializableNormalizer::normalize`, `\Heptacom\HeptaConnect\Core\Storage\Normalizer\SerializableNormalizer::supportsNormalization`, `\Heptacom\HeptaConnect\Core\Storage\Normalizer\StreamDenormalizer::denormalize`, `\Heptacom\HeptaConnect\Core\Storage\Normalizer\StreamDenormalizer::supportsDenormalization`, `\Heptacom\HeptaConnect\Core\Storage\Normalizer\StreamNormalizer::normalize` and `\Heptacom\HeptaConnect\Core\Storage\Normalizer\StreamNormalizer::supportsNormalization` to allow installations of `symfony/serializer: >=4` and `symfony/serializer: >= 5`

7.1.40 [0.8.4] - 2021-12-16

Removed

- Remove the code for unit tests, configuration for style checks as well as the Makefile

Fixed

- Provide portal node container services as definition instead of synthetic service to allow decoration for service ids `Heptacom\HeptaConnect\Portal\Base\Support\Contract\DeepCloneContract`, `Heptacom\HeptaConnect\Portal\Base\Support\Contract\DeepObjectIteratorContract`, `Psr\Http\Client\ClientInterface`, `Psr\Http\Message\RequestFactoryInterface`, `Psr\Http\Message\UriFactoryInterface`, `Psr\Http\Message\ResponseFactoryInterface` and `Psr\Http\Message\StreamFactoryInterface`
- Remove expired keys from the result of `\Heptacom\HeptaConnect\Core\Portal\PortalStorage::getMultiple`

7.1.41 [0.8.3] - 2021-12-02

Fixed

- Fix auto-wiring array values from portal configuration

7.1.42 [0.8.2] - 2021-11-25

Fixed

- Fix type error during reception when entity with numeric primary key is received

7.1.43 [0.8.1] - 2021-11-22

Fixed

- Fix stack building to allow for decorators. Previously when a portal extension had provided a decorator for a flow component, the stack would only contain the decorator and would miss the source component.
  (`\Heptacom\HeptaConnect\Core\Emission\EmitterStackBuilder::pushSource`,
  `\Heptacom\HeptaConnect\Core\Exploration\ExplorerStackBuilder::pushSource`,
  `\Heptacom\HeptaConnect\Core\Reception\ReceiverStackBuilder::pushSource`,
  `\Heptacom\HeptaConnect\Core\Web\Http\HttpHandlerStackBuilder::pushSource`)

7.1.44 [0.8.0] - 2021-11-22

Added

- Add calls to `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\JobRepositoryContract::start` and `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\JobRepositoryContract::finish` in `\Heptacom\HeptaConnect\Core\Job\Handler\EmissionHandler::triggerEmission`, `\Heptacom\HeptaConnect\Core\Job\Handler\ExplorationHandler::triggerExplorations` and `\Heptacom\HeptaConnect\Core\Job\Handler\ReceptionHandler::triggerReception` to track job states
- Add caching layer to `\Heptacom\HeptaConnect\Core\Configuration\ConfigurationService::getPortalNodeConfiguration`
- Add composer dependency `symfony/event-dispatcher: ^4.0 || ^5.0`
- Add log message `\Heptacom\HeptaConnect\Core\Component\LogMessage::MARK_AS_FAILED_ENTITY_IS_UNMAPPED` with log message code `1637456198` for issues during logging error messages during reception
- Add log message `\Heptacom\HeptaConnect\Core\Component\LogMessage::RECEIVE_NO_SAVE_MAPPINGS_NOT_PROCESSED` for issues after saving mappings after a reception
- Introduce `\Heptacom\HeptaConnect\Core\Event\PostReceptionEvent` for reception new event dispatcher in reception
- Add post-processing type `\Heptacom\HeptaConnect\Portal\Base\Reception\PostProcessing\MarkAsFailedData`
- Implement new method `\Heptacom\HeptaConnect\Portal\Base\Reception\Contract\ReceiveContextInterface::getEventDispatcher` in `\Heptacom\HeptaConnect\Core\Reception\ReceiveContext::getEventDispatcher`
- Implement new method `\Heptacom\HeptaConnect\Portal\Base\Reception\Contract\ReceiveContextInterface::getPostProcessingBag` in `\Heptacom\HeptaConnect\Core\Reception\ReceiveContext::getEventDispatcher`
- Add post-processor base class `\Heptacom\HeptaConnect\Core\Reception\Contract\PostProcessorContract`
- Add post-processing for failed receptions using `\Heptacom\HeptaConnect\Core\Reception\PostProcessing\MarkAsFailedData` and handled in `\Heptacom\HeptaConnect\Core\Reception\PostProcessing\MarkAsFailedPostProcessor`
- Add post-processing for saving mappings after receptions using `\Heptacom\HeptaConnect\Core\Reception\PostProcessing\SaveMappingsData` and handled in `\Heptacom\HeptaConnect\Core\Reception\PostProcessing\SaveMappingsPostProcessor`
- Extract path building from `\Heptacom\HeptaConnect\Core\Storage\Normalizer\StreamNormalizer` and `\Heptacom\HeptaConnect\Core\Storage\Normalizer\StreamDenormalizer` into new service `\Heptacom\HeptaConnect\Core\Storage\Contract\StreamPathContract`
- Add log messages codes `1634868818`, `1634868819` to `\Heptacom\HeptaConnect\Core\Storage\Normalizer\StreamDenormalizer`

- Add log message `\Heptacom\HeptaConnect\Core\Component\LogMessage::STORAGE_STREAM_NORMALIZER_CONVERTS_HINT_TO_FILENAME` with the message code `1635462690` to track generated filenames from the stream file storage in `\Heptacom\HeptaConnect\Core\Storage\Normalizer\StreamNormalizer`
- Add log exception code `1636503503` to `\Heptacom\HeptaConnect\Core\Job\Handler\ReceptionHandler::triggerReception` when job has no related route
- Add log exception code `1636503504` to `\Heptacom\HeptaConnect\Core\Job\Handler\ReceptionHandler::triggerReception` when job has no entity
- Add log exception code `1636503505` to `\Heptacom\HeptaConnect\Core\Job\Handler\ReceptionHandler::triggerReception` when job refers a non-existing route
- Add log exception code `1636503506` to `\Heptacom\HeptaConnect\Core\Job\Handler\ReceptionHandler::triggerReception` when job refers to a route that is not configured to allow receptions
- Add log exception code `1636503507` to `\Heptacom\HeptaConnect\Core\Job\Handler\ReceptionHandler::triggerReception` when job has an entity, that is of a different type than the route's entity type
- Add log exception code `1636503508` to `\Heptacom\HeptaConnect\Core\Job\Handler\ReceptionHandler::triggerReception` when job has an entity, that has a different primary key than the one saved on the job
- Add web HTTP handler context factory interface `\Heptacom\HeptaConnect\Core\Web\Http\Contract\HttpHandleContextFactoryInterface` and implementation `\Heptacom\HeptaConnect\Core\Web\Http\HttpHandleContextFactory` as well as `\Heptacom\HeptaConnect\Core\Web\Http\HttpHandleContext`
- Add web HTTP stack building interfaces `\Heptacom\HeptaConnect\Core\Web\Http\Contract\HttpHandlerStackBuilderFactoryInterface`, `\Heptacom\HeptaConnect\Core\Web\Http\Contract\HttpHandlerStackBuilderInterface` and implementations `\Heptacom\HeptaConnect\Core\Web\Http\HttpHandlerStackBuilderFactory`, `\Heptacom\HeptaConnect\Core\Web\Http\HttpHandlerStackBuilder` for acting with web HTTP handlers
- Add web HTTP service interface `\Heptacom\HeptaConnect\Core\Web\Http\Contract\HttpHandleServiceInterface` and implementation `\Heptacom\HeptaConnect\Core\Web\Http\HttpHandleService` to validate and handle requests
- Add web HTTP actor interface `\Heptacom\HeptaConnect\Core\Web\Http\Contract\HttpHandlingActorInterface` and implementation `\Heptacom\HeptaConnect\Core\Web\Http\HttpHandlingActor` to process any request through a web HTTP handler stack
- Add interface `\Heptacom\HeptaConnect\Core\Web\Http\Contract\HttpHandlerUrlProviderFactoryInterface` for bridges to provide implementation as bridges implement routing
- Add log message `\Heptacom\HeptaConnect\Core\Component\LogMessage::WEB_HTTP_HANDLE_NO_THROW` used with log message code `1636845126` when handling the web request triggered an exception in the flow component
- Add log message `\Heptacom\HeptaConnect\Core\Component\LogMessage::WEB_HTTP_HANDLE_NO_HANDLER_FOR_PATH` used with log message code `1636845086` when handling the web request could not match any flow component
- Add log message `\Heptacom\HeptaConnect\Core\Component\LogMessage::WEB_HTTP_HANDLE_DISABLED` used with log message code `1636845085` when route is disabled and still called
- Add `\Heptacom\HeptaConnect\Core\Storage\Exception\GzipCompressException` for cases when gzip related methods fail
- Add exception code `1637432095` in `\Heptacom\HeptaConnect\Core\Storage\Normalizer\SerializableCompressNormalizer::normalize` when `gzcompress` fails to compress
- Add exception code `1637101289` in `\Heptacom\HeptaConnect\Core\Storage\Normalizer\StreamDenormalizer::denormalize` when file to denormalize does not exist
- Add exception code `1637432853` in `\Heptacom\HeptaConnect\Core\Storage\Normalizer\StreamNormalizer::normalize` when object is no `\Heptacom\HeptaConnect\Portal\Base\Serialization\Contract\SerializableStream`
- Add exception code `1637432854` in `\Heptacom\HeptaConnect\Core\Storage\Normalizer\StreamNormalizer::normalize` when object does not hold a valid stream
- Add exception code `1637433403` in `\Heptacom\HeptaConnect\Core\Portal\ServiceContainerCompilerPass\AddPortalConfigurationBindingsCompilerPass::process` when an array_combine call fails that logically should not be able to fail
- Add log message `\Heptacom\HeptaConnect\Core\Component\LogMessage::EMIT_NO_PRIMARY_KEY` used with log message code `1637434358` when emitted entity has no primary key
- Add parameter `$jobKey` in `\Heptacom\HeptaConnect\Core\Job\JobData::__construct`
- Add method `\Heptacom\HeptaConnect\Core\Job\JobData::getJobKey`
- Add service `Heptacom\HeptaConnect\Portal\Base\Web\Http\HttpHandlerUrlProviderInterface` to portal container
- Add service `Heptacom\HeptaConnect\Portal\Base\Web\Http\HttpHandlerCollection` to portal container

- Add service `Heptacom\HeptaConnect\Portal\Base\Web\Http\HttpHandlerCollection.decorator` to portal container
- Add log message code `1637527920` in `\Heptacom\HeptaConnect\Core\Reception\PostProcessing\SaveMappingsPostProcessor::handle` when an entity has been received with a primary key but has no mapping data
- Add log message code `1637527921` in `\Heptacom\HeptaConnect\Core\Reception\PostProcessing\SaveMappingsPostProcessor::handle` when an entity has been received with a primary key but has invalid mapping data

Changed

- Change parameter name of `\Heptacom\HeptaConnect\Core\Emission\EmitContext::markAsFailed` from `$datasetEntityClassName` to `$entityType`
- Change parameter name of `\Heptacom\HeptaConnect\Core\Emission\Contract\EmitterStackBuilderFactoryInterface::createEmitterStackBuilder` from `$entityClassName` to `$entityType`, respective change in its implementing class `\Heptacom\HeptaConnect\Core\Emission\EmitterStackBuilderFactory::createEmitterStackBuilder`
- Change parameter name of `\Heptacom\HeptaConnect\Core\Emission\EmitterStackBuilder::__construct` from `$entityClassName` to `$entityType`. Change the field name in corresponding functions that use the field (`\Heptacom\HeptaConnect\Core\Emission\EmitterStackBuilder::push`, `\Heptacom\HeptaConnect\Core\Emission\EmitterStackBuilder::pushSource`, `\Heptacom\HeptaConnect\Core\Emission\EmitterStackBuilder::pushDecorators`)
- Change parameter name of `\Heptacom\HeptaConnect\Core\Emission\EmitService::getEmitterStack` from `$entityClassName` to `$entityType`
- Change parameter name of `\Heptacom\HeptaConnect\Core\Exploration\Contract\ExplorerStackBuilderFactoryInterface::createExplorerStackBuilder` from `$entityClassName` to `$entityType`, respective change in its implementing class `\Heptacom\HeptaConnect\Core\Exploration\ExplorerStackBuilderFactory::createExplorerStackBuilder`
- Change parameter name of `\Heptacom\HeptaConnect\Core\Exploration\Contract\ExplorationActorInterface::performExploration` from `$entityClassName` to `$entityType`, respective change in its implementing class `\Heptacom\HeptaConnect\Core\Exploration\ExplorationActor::performExploration`
- Change parameter name of `\Heptacom\HeptaConnect\Core\Exploration\ExplorerStackBuilder::__construct` from `$entityClassName` to `$entityType`. Change the field name in corresponding functions that use the field (`\Heptacom\HeptaConnect\Core\Exploration\ExplorerStackBuilder::push`, `\Heptacom\HeptaConnect\Core\Exploration\ExplorerStackBuilder::pushSource`, `\Heptacom\HeptaConnect\Core\Exploration\ExplorerStackBuilder::pushDecorators`)
- Change parameter name of `\Heptacom\HeptaConnect\Core\Reception\Contract\ReceiverStackBuilderFactoryInterface::createReceiverStackBuilder` from `$entityClassName` to `$entityType`, respective change in its implementing class `\Heptacom\HeptaConnect\Core\Reception\ReceiverStackBuilderFactory::createReceiverStackBuilder`
- Change parameter name of `\Heptacom\HeptaConnect\Core\Reception\ReceiverStackBuilder::__construct` from `$entityClassName` to `$entityType`. Change the field name in corresponding functions that use the field (`\Heptacom\HeptaConnect\Core\Reception\ReceiverStackBuilder::push`, `\Heptacom\HeptaConnect\Core\Reception\ReceiverStackBuilder::pushSource`, `\Heptacom\HeptaConnect\Core\Reception\ReceiverStackBuilder::pushDecorators`)
- Change parameter name of `\Heptacom\HeptaConnect\Core\Reception\ReceiveService::getReceiverStack` from `$entityClassName` to `$entityType`
- Change parameter name of `\Heptacom\HeptaConnect\Core\Mapping\Contract\MappingServiceInterface::get` from `$datasetEntityClassName` to `$entityType`, respective change in its implementing class for `\Heptacom\HeptaConnect\Core\Mapping\MappingService::get`
- Change parameter name of `\Heptacom\HeptaConnect\Core\Mapping\Contract\MappingServiceInterface::getListByExternalIds` from `$datasetEntityClassName` to `$entityType`, respective change in its implementing class for `\Heptacom\HeptaConnect\Core\Mapping\MappingService::getListByExternalIds`
- Change parameter name of `\Heptacom\HeptaConnect\Core\Mapping\MappingNodeStruct::__construct` from `$datasetEntityClassName` to `$entityType`
- Change parameter name of `\Heptacom\HeptaConnect\Core\Mapping\Publisher::publish` from `$datasetEntityClassName` to `$entityType`
- Change parameter name of `\Heptacom\HeptaConnect\Core\Reception\Support\PrimaryKeyChangesAttachable::__construct` from `$datasetEntityClassName` to `$entityType`

- Change method name from `\Heptacom\HeptaConnect\Portal\Base\Mapping\Contract\MappingComponentStructContract::getDatasetEntityClassName` to `\Heptacom\HeptaConnect\Portal\Base\Mapping\Contract\MappingComponentStructContract::getEntityType`
- Change method name from `\Heptacom\HeptaConnect\Core\Mapping\MappingStruct::getDatasetEntityClassName` to `\Heptacom\HeptaConnect\Core\Mapping\MappingStruct::getEntityType`
- Change method name from `\Heptacom\HeptaConnect\Core\Mapping\MappingNodeStruct::getDatasetEntityClassName` to `\Heptacom\HeptaConnect\Core\Mapping\MappingNodeStruct::getEntityType`
- Change method name from `\Heptacom\HeptaConnect\Core\Mapping\MappingNodeStruct::setDatasetEntityClassName` to `\Heptacom\HeptaConnect\Core\Mapping\MappingNodeStruct::setEntityType`
- Change method name from `\Heptacom\HeptaConnect\Core\Reception\Support\PrimaryKeyChangesAttachable::getForeignDatasetEntityClassName` to `\Heptacom\HeptaConnect\Core\Reception\Support\PrimaryKeyChangesAttachable::getForeignEntityType`
- Change method name from `\Heptacom\HeptaConnect\Core\Reception\Support\PrimaryKeyChangesAttachable::setForeignDatasetEntityClassName` to `\Heptacom\HeptaConnect\Core\Reception\Support\PrimaryKeyChangesAttachable::setForeignEntityType`
- Add dependency onto `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\JobRepositoryContract` into `\Heptacom\HeptaConnect\Core\Job\Handler\EmissionHandler`, `\Heptacom\HeptaConnect\Core\Job\Handler\ExplorationHandler` and `\Heptacom\HeptaConnect\Core\Job\Handler\ReceptionHandler` for job tracking
- Add dependency onto `\Psr\Cache\CacheItemPoolInterface` into `\Heptacom\HeptaConnect\Core\Configuration\ConfigurationService` for configuration caching
- Remove parameter `$mappingService` from `\Heptacom\HeptaConnect\Core\Reception\ReceiveContext::__construct` and `\Heptacom\HeptaConnect\Core\Reception\ReceiveContextFactory::__construct` as it is no longer needed
- Add parameter `$postProcessors` to `\Heptacom\HeptaConnect\Core\Reception\ReceiveContext::__construct` and `\Heptacom\HeptaConnect\Core\Reception\ReceiveContextFactory::__construct` to contain every post-processing handler for this context
- Change `\Heptacom\HeptaConnect\Core\Reception\ReceiveContext::markAsFailed` to add `\Heptacom\HeptaConnect\Portal\Base\Reception\PostProcessing\MarkAsFailedData` to the post-processing data bag instead of directly passing to `\Heptacom\HeptaConnect\Core\Mapping\Contract\MappingServiceInterface::addException`
- Remove parameter `$mappingPersister` from `\Heptacom\HeptaConnect\Core\Reception\ReceptionActor::__construct` as its usage has been moved into `\Heptacom\HeptaConnect\Core\Reception\PostProcessing\SaveMappingsPostProcessor`
- Move of saving mappings from `\Heptacom\HeptaConnect\Core\Reception\ReceptionActor::performReception` into `\Heptacom\HeptaConnect\Core\Reception\PostProcessing\SaveMappingsPostProcessor::handle`
- Add dependency onto `\Psr\Log\LoggerInterface` into `\Heptacom\HeptaConnect\Core\Storage\Normalizer\StreamNormalizer` for logging filename conversions
- Change dependency in `\Heptacom\HeptaConnect\Core\Emission\EmissionActor` from `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\RouteRepositoryContract` into `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Listing\ReceptionRouteListActionInterface` for more performant route lookup
- Change dependency in `\Heptacom\HeptaConnect\Core\Job\Handler\ReceptionHandler` from `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\RouteRepositoryContract` into `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Get\RouteGetActionInterface` for more performant route reading
- Allow `\Heptacom\HeptaConnect\Core\Job\Contract\ReceptionHandlerInterface::triggerReception` to throw `\Heptacom\HeptaConnect\Core\Job\Exception\ReceptionJobHandlingException`
- Add dependency onto `\Psr\Log\LoggerInterface` into `\Heptacom\HeptaConnect\Core\Job\Handler\ReceptionHandler` for logging exceptions
- Add dependency onto `\Psr\Log\LoggerInterface` into `\Heptacom\HeptaConnect\Core\Reception\PostProcessing\SaveMappingsPostProcessor` for logging unclearmapping scenarios

Deprecated

- Move `\Heptacom\HeptaConnect\Core\Storage\Normalizer\StreamNormalizer::STORAGE_LOCATION` into `\Heptacom\HeptaConnect\Core\Storage\Contract\StreamPathContract::STORAGE_LOCATION`

Removed

- Remove `\Heptacom\HeptaConnect\Core\Webhook\Contract\UrlProviderInterface`

- Remove `\Heptacom\HeptaConnect\Core\Webhook\WebhookContext` in favour of `\Heptacom\HeptaConnect\Core\Web\Http\HttpHandleContext`
- Remove `\Heptacom\HeptaConnect\Core\Webhook\WebhookContextFactory` in favour of `\Heptacom\HeptaConnect\Core\Web\Http\HttpHandleContextFactory`
- Remove `\Heptacom\HeptaConnect\Core\Webhook\WebhookService`
- Remove interface `\Heptacom\HeptaConnect\Core\Mapping\Contract\MappingServiceInterface::ensurePersistence` and implementation `\Heptacom\HeptaConnect\Core\Mapping\MappingService::ensurePersistence` in favour of `\Heptacom\HeptaConnect\Storage\Base\MappingPersister\Contract\MappingPersisterContract`

Fixed

- Provide callback-function to \array_filter in `Heptacom\HeptaConnect\Core\Flow\DirectEmissionFlow\DirectEmissionFlow::run` to only filter out primary keys with null and not 0
- `\Heptacom\HeptaConnect\Core\Storage\Normalizer\StreamDenormalizer` rejects null and empty string as data
- Usage of `\Ramsey\Uuid\Uuid` in `\Heptacom\HeptaConnect\Core\Storage\Normalizer\StreamNormalizer` only supported `ramsey/uuid: 3` but composer configuration allowed installation of `ramsey/uuid: 4`. Now it is used cross-compatible to work with `ramsey/uuid: 3 || 4`
- `\Heptacom\HeptaConnect\Core\Configuration\ConfigurationService::setPortalNodeConfiguration` removes nested `null` values and does not store `null` anymore
- Fix automatic prototyping when a portal provides an interface in `\Heptacom\HeptaConnect\Core\Portal\ServiceContainerCompilerPass\RemoveAutoPrototypedDefinitionsCompilerPass::isPrototypable`

7.1.45 [0.7.0] - 2021-09-25

Added

- Change implementation for `\Heptacom\HeptaConnect\Portal\Base\Portal\Contract\PortalStorageInterface` in `\Heptacom\HeptaConnect\Core\Portal\PortalStorage` to allow PSR simple cache compatibility
- Add log messages codes `1631387202`, `1631387363`, `1631387430`, `1631387448`, `1631387470`, `1631387510`, `1631561839`, `1631562097`, `1631562285`, `1631562928`, `1631563058`, `1631563639`, `1631563699`, `1631565257`, `1631565376`, `1631565446` to `\Heptacom\HeptaConnect\Core\Portal\PortalStorage`
- Add interface `\Heptacom\HeptaConnect\Core\Reception\Contract\ReceiveContextFactoryInterface` to `\Heptacom\HeptaConnect\Core\Reception\ReceiveContextFactory`
- Add interface `\Heptacom\HeptaConnect\Core\Job\Contract\ReceptionHandlerInterface` to `\Heptacom\HeptaConnect\Core\Job\Handler\ReceptionHandler`
- Add interface `\Heptacom\HeptaConnect\Core\Job\Contract\ExplorationHandlerInterface` to `\Heptacom\HeptaConnect\Core\Job\Handler\ExplorationHandler`
- Add interface `\Heptacom\HeptaConnect\Core\Job\Contract\EmissionHandlerInterface` to `\Heptacom\HeptaConnect\Core\Job\Handler\EmissionHandler`
- Add interface `\Heptacom\HeptaConnect\Core\Emission\Contract\EmitContextFactoryInterface` to `\Heptacom\HeptaConnect\Core\Emission\EmitContextFactory`
- Add method `\Heptacom\HeptaConnect\Core\Exploration\DirectEmitter::batch` for better performance in direct emissions

Changed

- `\Heptacom\HeptaConnect\Core\Portal\PortalStorage::get` and `\Heptacom\HeptaConnect\Core\Portal\PortalStorage::set` will now throw exceptions when normalization could not happen
- Add parameter for `\Psr\Log\LoggerInterface` dependency in `\Heptacom\HeptaConnect\Core\Portal\PortalStorage::__construct` and `\Heptacom\HeptaConnect\Core\Portal\PortalStorageFactory::__construct`
- Change type of parameter `\Heptacom\HeptaConnect\Core\Reception\ReceiveContextFactory` to its new interface `\Heptacom\HeptaConnect\Core\Reception\Contract\ReceiveContextFactoryInterface` in `\Heptacom\HeptaConnect\Core\Reception\ReceiveService::__construct`
- Change type of parameter `\Heptacom\HeptaConnect\Core\Job\Handler\EmissionHandler` to its new interface `\Heptacom\HeptaConnect\Core\Job\Contract\EmissionHandlerInterface` in `\Heptacom\HeptaConnect\Core\Job\DelegatingJobActor::__construct`

- Change type of parameter `\Heptacom\HeptaConnect\Core\Job\Handler\ReceptionHandler` to its new interface `\Heptacom\HeptaConnect\Core\Job\Contract\ReceptionHandlerInterface` in `\Heptacom\HeptaConnect\Core\Job\DelegatingJobActor::__construct`
- Change type of parameter `\Heptacom\HeptaConnect\Core\Job\Handler\ExplorationHandler` to its new interface `\Heptacom\HeptaConnect\Core\Job\Contract\ExplorationHandlerInterface` in `\Heptacom\HeptaConnect\Core\Job\DelegatingJobActor::__construct`
- Change type of parameter `\Heptacom\HeptaConnect\Core\Emission\EmitContextFactory` to its new interface `\Heptacom\HeptaConnect\Core\Emission\Contract\EmitContextFactoryInterface` in `\Heptacom\HeptaConnect\Core\Emission\EmitService::__construct`
- Change behavior of service `\Heptacom\HeptaConnect\Core\Flow\DirectEmissionFlow\DirectEmissionFlow` to not create mappings anymore
- Remove parameter `\Heptacom\HeptaConnect\Core\Mapping\Contract\MappingServiceInterface` from `\Heptacom\HeptaConnect\Core\Flow\DirectEmissionFlow\DirectEmissionFlow::__construct`
- Change method `\Heptacom\HeptaConnect\Core\Reception\ReceptionActor::saveMappings` to use new service `\Heptacom\HeptaConnect\Storage\Base\MappingPersister\Contract\MappingPersisterContract`
- `\Heptacom\HeptaConnect\Core\Exploration\ExplorerStackBuilder::pushSource` and `\Heptacom\HeptaConnect\Core\Exploration\ExplorerStackBuilder::pushDecorators` don't push explorers onto the stack when they are already in the stack
- `\Heptacom\HeptaConnect\Core\Emission\EmitterStackBuilder::pushSource` and `\Heptacom\HeptaConnect\Core\Emission\EmitterStackBuilder::pushDecorators` don't push emitters onto the stack when they already in the stack
- `\Heptacom\HeptaConnect\Core\Reception\ReceiverStackBuilder::pushSource` and `\Heptacom\HeptaConnect\Core\Reception\ReceiverStackBuilder::pushDecorators` don't push receivers onto the stack when they already in the stack

Removed

- Remove method `\Heptacom\HeptaConnect\Core\Exploration\DirectEmitter::run` as it became obsolete

7.1.46 [0.6.0] - 2021-07-26

Added

- Add `\Heptacom\HeptaConnect\Core\Exploration\Contract\ExploreServiceInterface::dispatchExploreJob` to start an exploration as a job via `\Heptacom\HeptaConnect\Core\Job\Contract\JobDispatcherContract::dispatch`
- Add `\Heptacom\HeptaConnect\Core\Job\Handler\ExplorationHandler` to handle exploration jobs `\Heptacom\HeptaConnect\Core\Job\Type\Exploration`
- Add support for handling exploration jobs in `\Heptacom\HeptaConnect\Core\Job\DelegatingJobActor` with using `\Heptacom\HeptaConnect\Core\Job\Handler\ExplorationHandler`
- Add `\Psr\Http\Message\ResponseFactoryInterface` service to the portal containers in `\Heptacom\HeptaConnect\Core\Portal\PortalStackServiceContainerBuilder` for better HTTP and messaging PSR support for portal developers
- Add `\Psr\Http\Message\StreamFactoryInterface` service to the portal containers in `\Heptacom\HeptaConnect\Core\Portal\PortalStackServiceContainerBuilder` for better HTTP and messaging PSR support for portal developers

Changed

- Direct emission and explorations create mappings via `\Heptacom\HeptaConnect\Core\Mapping\Contract\MappingServiceInterface::getListByExternalIds` on `\Heptacom\HeptaConnect\Core\Exploration\Contract\ExplorationActorInterface::performExploration` when implemented by `\Heptacom\HeptaConnect\Core\Exploration\ExplorationActor::performExploration`

7.1.47 [0.5.1] - 2021-07-13

Fixed

- Remove impact of entity primary keys on lock keys in `\Heptacom\HeptaConnect\Core\Job\Handler\ReceptionHandler::triggerReception`

7.1.48 [0.5.0] - 2021-07-11

Added

- Add composer dependency `symfony/yaml: ^4.4|^5.0`
- Add base class `\Heptacom\HeptaConnect\Portal\Base\Flow\DirectEmission\DirectEmissionFlowContract` to `\Heptacom\HeptaConnect\Core\Flow\DirectEmissionFlow` to expose service for portals
- Add classes to hold job data for batch processing `\Heptacom\HeptaConnect\Core\Job\JobData` and `\Heptacom\HeptaConnect\Core\Job\JobDataCollection`
- Add class `\Heptacom\HeptaConnect\Core\Portal\PortalLogger` that can decorate any `\Psr\Log\LoggerInterface` to apply any additional message prefix and context to all logs
- Add `\Heptacom\HeptaConnect\Portal\Base\Publication\Contract\PublisherInterface` to portal node service container
- Add `\Heptacom\HeptaConnect\Portal\Base\Flow\DirectEmission\DirectEmissionFlowContract` to portal node service container

Changed

- The acting to jobs in `\Heptacom\HeptaConnect\Core\Job\Contract\DelegatingJobActorContract::performJob` will now happen in batches in `\Heptacom\HeptaConnect\Core\Job\Contract\DelegatingJobActorContract::performJobs` and expects different parameters
- The trigger on emission jobs in `\Heptacom\HeptaConnect\Core\Job\Handler\EmissionHandler::triggerEmission` will now happen in batches and expects different parameters
- The trigger on reception jobs in `\Heptacom\HeptaConnect\Core\Job\Handler\ReceptionHandler::triggerReception` will now happen in batches and expects different parameters
- Change signature of `\Heptacom\HeptaConnect\Core\Reception\Contract\ReceptionActorInterface::performReception` to not rely on previously entities bound to `\Heptacom\HeptaConnect\Portal\Base\Mapping\Contract\MappingInterface` objects
- Change signature of `\Heptacom\HeptaConnect\Core\Reception\ReceiveContext::markAsFailed` to not rely on previously entities bound to `\Heptacom\HeptaConnect\Portal\Base\Mapping\Contract\MappingInterface` objects
- Do most of the business logic for reception in `\Heptacom\HeptaConnect\Core\Job\Handler\ReceptionHandler` to have job related logic less bound to reception processes in general

Deprecated

- Deprecate cronjobs and therefore mark `\Heptacom\HeptaConnect\Core\Cronjob\CronjobContext`, `\Heptacom\HeptaConnect\Core\Cronjob\CronjobContextFactory`, `\Heptacom\HeptaConnect\Core\Cronjob\CronjobService` as internal
- Deprecate webhooks and therefore mark `\Heptacom\HeptaConnect\Core\Webhook\WebhookContext`, `\Heptacom\HeptaConnect\Core\Webhook\WebhookContextFactory`, `\Heptacom\HeptaConnect\Core\Webhook\WebhookService`, `\Heptacom\HeptaConnect\Core\Webhook\Contact\UrlProviderInterface` as internal

Removed

- Move `\Heptacom\HeptaConnect\Core\Flow\DirectEmissionFlow\DirectEmissionResult` into the portal base package as `\Heptacom\HeptaConnect\Portal\Base\Flow\DirectEmission\DirectEmissionResult`
- Move `\Heptacom\HeptaConnect\Core\Flow\DirectEmissionFlow\Exception\UnidentifiedEntityException` into the portal base package as `\Heptacom\HeptaConnect\Portal\Base\Flow\DirectEmission\Exception\UnidentifiedEntityException`
- The handling of jobs in `\Heptacom\HeptaConnect\Core\Flow\MessageQueueFlow\MessageHandler::handleJob` does not republish failed jobs anymore. That feature will be added back again in a different form
- The trigger on emission jobs in `\Heptacom\HeptaConnect\Core\Job\Handler\EmissionHandler::triggerEmission` will no longer report back success

- The trigger on reception jobs in `\Heptacom\HeptaConnect\Core\Job\Handler\ReceptionHandler::triggerReception` will no longer report back success
- Remove automatically registered services in `\Heptacom\HeptaConnect\Core\Portal\ServiceContainerCompilerPass\RemoveAutoPrototypedDefinitionsCompilerPass` that is based on `\Throwable`, `\Heptacom\HeptaConnect\Dataset\Base\Contract\AttachableInterface`, `\Heptacom\HeptaConnect\Dataset\Base\Contract\CollectionInterface` and `\Heptacom\HeptaConnect\Dataset\Base\Contract\DatasetEntityContract`

7.1.49 [0.9.4.0] - 2023-03-04

Deprecated

- Deprecate and discourage usage of `\Heptacom\HeptaConnect\Dataset\Base\Contract\DeferralAwareInterface` and `\Heptacom\HeptaConnect\Dataset\Base\Support\DeferralAwareTrait` as it has not been a practical solution to defer closure execution in a different process

7.1.50 [0.9.1.1] - 2022-09-28

Added

- Add method `\Heptacom\HeptaConnect\Dataset\Base\Support\AbstractCollection::withoutItems` to create safely new instances of the same type but without content
- Add method `\Heptacom\HeptaConnect\Dataset\Base\Support\AbstractCollection::chunk` to iterate over the items prepared in a buffer of a certain size
- Add method `\Heptacom\HeptaConnect\Dataset\Base\Support\AbstractCollection::asArray` to access the items of the collection as fixed size array
- Add method `\Heptacom\HeptaConnect\Dataset\Base\Support\AbstractCollection::reverse` to reverse the order of the collection items
- Add method `\Heptacom\HeptaConnect\Dataset\Base\Support\AbstractCollection::isEmpty` to check whether the collection is empty without counting
- Add aggregation methods `\Heptacom\HeptaConnect\Dataset\Base\ScalarCollection\FloatCollection::sum`, `\Heptacom\HeptaConnect\Dataset\Base\ScalarCollection\FloatCollection::max` and `\Heptacom\HeptaConnect\Dataset\Base\ScalarCollection\FloatCollection::min` to reduce boilerplate code when aggregating a float collection
- Add aggregation methods `\Heptacom\HeptaConnect\Dataset\Base\ScalarCollection\IntegerCollection::sum`, `\Heptacom\HeptaConnect\Dataset\Base\ScalarCollection\IntegerCollection::max` and `\Heptacom\HeptaConnect\Dataset\Base\ScalarCollection\IntegerCollection::min` to reduce boilerplate code when aggregating an integer collection

7.1.51 [0.9.1.0] - 2022-08-15

Added

- Add method `\Heptacom\HeptaConnect\Dataset\Base\ScalarCollection\StringCollection::join` to implode strings

7.1.52 [0.9.0.0] - 2022-04-02

Added

- Add `\Heptacom\HeptaConnect\Dataset\Base\Support\AttachmentAwareTrait::isAttached` to check for a specific instance of an object in the attachment list
- Add `\Heptacom\HeptaConnect\Dataset\Base\Support\AttachmentAwareTrait::detach` to remove a specific instance from the attachment list
- Add `\Heptacom\HeptaConnect\Dataset\Base\Contract\AttachmentAwareInterface` to match the trait `\Heptacom\HeptaConnect\Dataset\Base\Support\AttachmentAwareTrait` and add it to `\Heptacom\HeptaConnect\Dataset\Base\Contract\DatasetEntityContract`
- Add class `\Heptacom\HeptaConnect\Dataset\Base\File\FileReferenceContract` as a base class for various file reference implementations
- Add class `\Heptacom\HeptaConnect\Dataset\Base\File\FileReferenceCollection` as a collection for `\Heptacom\HeptaConnect\Dataset\Base\File\FileReferenceContract`

Changed

- Implement possible usage of interface FQCNs as parameter in the methods `\Heptacom\HeptaConnect\Dataset\Base\Support\AttachmentAwareTrait::hasAttached`, `\Heptacom\HeptaConnect\Dataset\Base\Support\AttachmentAwareTrait::getAttachment`, `\Heptacom\HeptaConnect\Dataset\Base\Support\AttachmentAwareTrait::detachByType`
- Set `array-key` type on iterating over collections that implement the `\Heptacom\HeptaConnect\Dataset\Base\Contract\CollectionInterface` to `int` as they only accept iterables keyed by `int`
- Add final modifier to `\Heptacom\HeptaConnect\Dataset\Base\ScalarCollection\BooleanCollection`, `\Heptacom\HeptaConnect\Dataset\Base\ScalarCollection\DateCollection`, `\Heptacom\HeptaConnect\Dataset\Base\ScalarCollection\DateTimeCollection`, `\Heptacom\HeptaConnect\Dataset\Base\ScalarCollection\FloatCollection`, `\Heptacom\HeptaConnect\Dataset\Base\ScalarCollection\IntegerCollection`, `\Heptacom\HeptaConnect\Dataset\Base\ScalarCollection\StringCollection`, `\Heptacom\HeptaConnect\Dataset\Base\TaggedCollection\TaggedBooleanCollection`, `\Heptacom\HeptaConnect\Dataset\Base\TaggedCollection\TaggedDateCollection`, `\Heptacom\HeptaConnect\Dataset\Base\TaggedCollection\TaggedDateTimeCollection`, `\Heptacom\HeptaConnect\Dataset\Base\TaggedCollection\TaggedFloatCollection`, `\Heptacom\HeptaConnect\Dataset\Base\TaggedCollection\TaggedIntegerCollection`, `\Heptacom\HeptaConnect\Dataset\Base\TaggedCollection\TaggedStringCollection`, `\Heptacom\HeptaConnect\Dataset\Base\TaggedCollection\TagItem`, `\Heptacom\HeptaConnect\Dataset\Base\Translatable\ScalarCollection\TranslatableBooleanCollection`, `\Heptacom\HeptaConnect\Dataset\Base\Translatable\ScalarCollection\TranslatableDateCollection`, `\Heptacom\HeptaConnect\Dataset\Base\Translatable\ScalarCollection\TranslatableDateTimeCollection`, `\Heptacom\HeptaConnect\Dataset\Base\Translatable\ScalarCollection\TranslatableFloatCollection`, `\Heptacom\HeptaConnect\Dataset\Base\Translatable\ScalarCollection\TranslatableIntegerCollection`, `\Heptacom\HeptaConnect\Dataset\Base\Translatable\ScalarCollection\TranslatableStringCollection`, `\Heptacom\HeptaConnect\Dataset\Base\Translatable\TranslatableBoolean`, `\Heptacom\HeptaConnect\Dataset\Base\Translatable\TranslatableDate`, `\Heptacom\HeptaConnect\Dataset\Base\Translatable\TranslatableDateTime`, `\Heptacom\HeptaConnect\Dataset\Base\Translatable\TranslatableFloat`, `\Heptacom\HeptaConnect\Dataset\Base\Translatable\TranslatableInteger`, `\Heptacom\HeptaConnect\Dataset\Base\Translatable\TranslatableString`, `\Heptacom\HeptaConnect\Dataset\Base\AttachmentCollection`, `\Heptacom\HeptaConnect\Dataset\Base\Date`, `\Heptacom\HeptaConnect\Dataset\Base\Dependency`, `\Heptacom\HeptaConnect\Dataset\Base\DependencyCollection` and `\Heptacom\HeptaConnect\Dataset\Base\TypedDatasetEntityCollection` to ensure correct usage of implementation. Decoration by their interfaces or base classes is still possible

Deprecated

- Copy and deprecate `\Heptacom\HeptaConnect\Dataset\Base\Support\AttachmentAwareTrait::unattach` to `\Heptacom\HeptaConnect\Dataset\Base\Support\AttachmentAwareTrait::detachByType` for correct usage of English language and distinguish from `\Heptacom\HeptaConnect\Dataset\Base\Support\AttachmentAwareTrait::detach`

st changes

7.1.53 [0.8.5] - 2021-12-28

Fixed

- Change composer dependency `bentools/iterable-functions: >=1 <2` to `bentools/iterable-functions: >=1.4 <2` to ensure availability of `iterable_map`

7.1.54 [0.8.4] - 2021-12-16

Removed

- Remove the code for unit tests, configuration for style checks as well as the Makefile

7.1.55 [0.8.0] - 2021-11-22

Changed

- Change composer dependency `bentools/iterable-functions: >=1` to `bentools/iterable-functions: >=1 <2`
- Change method name of `\Heptacom\HeptaConnect\Dataset\Base\Contract\ForeignKeyAwareInterface::getForeignDatasetEntityClassName` to `\Heptacom\HeptaConnect\Dataset\Base\Contract\ForeignKeyAwareInterface::getForeignEntityType`

7.1.56 [0.7.0] - 2021-09-25

Changed

- Amend typehint for `\Heptacom\HeptaConnect\Dataset\Base\Support\AbstractCollection::__construct`, `\Heptacom\HeptaConnect\Dataset\Base\TypedDatasetEntityCollection::__construct` and `\Heptacom\HeptaConnect\Dataset\Base\Contract\CollectionInterface::push` to improve static code analysis.

Fixed

- Change signature `\Heptacom\HeptaConnect\Dataset\Base\TypedDatasetEntityCollection::__construct` to allow iterables instead of array like other collections

7.1.57 [0.6.0] - 2021-07-26

Added

- New method `\Heptacom\HeptaConnect\Dataset\Base\Contract\CollectionInterface::column` to improve common cases from `\Heptacom\HeptaConnect\Dataset\Base\Contract\CollectionInterface::map` usage

Changed

- Amend `\Heptacom\HeptaConnect\Dataset\Base\Contract\CollectionInterface::map` typehint for callables to improve static code analysis

7.1.58 [0.5.0] - 2021-07-11

Added

- New composer dependency `opis/closure: ^3.6` to allow serialization of closures
- New class `\Heptacom\HeptaConnect\Dataset\Base\TypedDatasetEntityCollection` to have a dataset entity collection that ensures to contain a single type only to improve common case `\Heptacom\HeptaConnect\Dataset\Base\Contract\CollectionInterface::filter`
- New class `\Heptacom\HeptaConnect\Dataset\Base\Translatable\ScalarCollection\AbstractTranslatableScalarCollection` to allow translations of any collections
- New class `\Heptacom\HeptaConnect\Dataset\Base\Translatable\ScalarCollection\TranslatableBooleanCollection` to allow translations of type `\Heptacom\HeptaConnect\Dataset\Base\ScalarCollection\BooleanCollection`

- New class `\Heptacom\HeptaConnect\Dataset\Base\Translatable\ScalarCollection\TranslatableDateCollection` to allow translations of type `\Heptacom\HeptaConnect\Dataset\Base\ScalarCollection\DateCollection`
- New class `\Heptacom\HeptaConnect\Dataset\Base\Translatable\ScalarCollection\TranslatableDateTimeCollection` to allow translations of type `\Heptacom\HeptaConnect\Dataset\Base\ScalarCollection\DateTimeCollection`
- New class `\Heptacom\HeptaConnect\Dataset\Base\Translatable\ScalarCollection\TranslatableFloatCollection` to allow translations of type `\Heptacom\HeptaConnect\Dataset\Base\ScalarCollection\FloatCollection`
- New class `\Heptacom\HeptaConnect\Dataset\Base\Translatable\ScalarCollection\TranslatableIntegerCollection` to allow translations of type `\Heptacom\HeptaConnect\Dataset\Base\ScalarCollection\IntegerCollection`
- New class `\Heptacom\HeptaConnect\Dataset\Base\Translatable\ScalarCollection\TranslatableStringCollection` to allow translations of type `\Heptacom\HeptaConnect\Dataset\Base\ScalarCollection\StringCollection`
- New method in `\Heptacom\HeptaConnect\Dataset\Base\Contract\DeferralAwareInterface::copyDeferrals` to copy deferrals from one deferral aware to another one
- New default implementation of method `\Heptacom\HeptaConnect\Dataset\Base\Contract\DeferralAwareInterface::copyDeferrals` in `\Heptacom\HeptaConnect\Dataset\Base\Support\DeferralAwareTrait`

Changed

- Rename `\Heptacom\HeptaConnect\Dataset\Base\Translatable\GenericTranslatable` to `\Heptacom\HeptaConnect\Dataset\Base\Translatable\AbstractTranslatable`
- Add `\Heptacom\HeptaConnect\Dataset\Base\Contract\DeferralAwareInterface` to `\Heptacom\HeptaConnect\Dataset\Base\Contract\DatasetEntityContract`

7.1.59 [0.9.2.0] - 2023-03-24

Added

- Add `\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\Shipment\ShipmentState::STATE_PARTIALLY_SHIPPED`
- Add `\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\Shipment\ShipmentState::STATE_PARTIALLY_RETURNED`

7.1.60 [0.9.1.0] - 2022-09-21

Added

- Add `\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\Shipment\ShipmentState` with the states unknown, open, cancelled, returned and shipped
- Add `\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\Shipment\Shipment` entity with `\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\Shipment\ShipmentCollection` to group shipment related information like address, tracking code, state and method
- Add `\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\Shipment\ShipmentAwareInterface` to describe entities related to shipments
- Add `\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\Shipment\ShipmentAwareTrait` as supporting implementation for every entity implementing `\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\Shipment\ShipmentAwareInterface`
- Make `\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\LineItem\Shipping` and `\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\LineItem\Product` aware of their related shipments by implementing `\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\Shipment\ShipmentAwareInterface`
- Add `\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\Order::aggregateShipments` to collect shipment information from all line items
- Add `\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\Transaction` entity with `\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\TransactionCollection` to hold payment transaction related data with optional relation to line items to allow payments without context
- Add `\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\Credit` entity based on `\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\Transaction`
- Add `\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\Payment` entity based on `\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\Transaction`
- Add property `manufacturerNumber` to `\Heptacom\HeptaConnect\Dataset\Ecommerce\Product\Product` to store manufacturer numbers

Changed

- Extract `\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\Transaction` entity from `\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\Refund` to represent any type of payment

Deprecated

- Add deprecation warnings to usage of `\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\Order::$deliveryTrackingCode` in `\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\Order::getDeliveryTrackingCode` and `\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\Order::setDeliveryTrackingCode`. Use `\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\Order::aggregateShipments` with `\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\Shipment\Shipment` and implementations of `\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\Shipment\ShipmentAwareInterface` instead
- Add deprecation warnings to usage of the payment related properties `\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\Order::$paymentState`, `\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\Order::$paymentTransactionCode`, `\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\Order::$paymentMethod` and `\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\Order::$refund` in `\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\Order::getPaymentState`, `\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\Order::setPaymentState`,

`\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\Order::getPaymentTransactionCode`,
`\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\Order::setPaymentTransactionCode`,
`\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\Order::getPaymentMethod`,
`\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\Order::setPaymentMethod`,
`\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\Order::isRefunded`,
`\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\Order::getRefund` and
`\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\Order::setRefund`. Use
`\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\Order::getTransactions` with implementations of
`\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\Transaction` instead

7.1.61 [0.9.0.0] - 2022-04-02

Added

- Add property medias to `\Heptacom\HeptaConnect\Dataset\Ecommerce\Product\Product` as `\Heptacom\HeptaConnect\Dataset\Ecommerce\Media\MediaCollection`
- Use `\Heptacom\HeptaConnect\Dataset\Base\File\FileReferenceContract` in `\Heptacom\HeptaConnect\Dataset\Ecommerce\Media\Media` to normalize file usage in media transfer

Removed

- Replace `\Heptacom\HeptaConnect\Dataset\Ecommerce\Media\Media::getNormalizedStream` and `\Heptacom\HeptaConnect\Dataset\Ecommerce\Media\Media::setNormalizedStream` with `\Heptacom\HeptaConnect\Dataset\Ecommerce\Media\Media::getFile` and `\Heptacom\HeptaConnect\Dataset\Ecommerce\Media\Media::setFile`. To migrate stop job dispatching processes and process all jobs to prevent deserialization issue

st changes

7.1.62 [0.8.3] - 2022-02-16

Added

- Add property to `\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\Refund` to flag refund as partial or full refund

Fixed

- Add missing import of parent class for `\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\Refund`

7.1.63 [0.8.2] - 2022-02-15

Added

- Add new entity `\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\Refund` to hold refund information
- Add new property for refunds to `\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\Order`. New methods: `\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\Order::isRefunded`, `\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\Order::getRefund`, `\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\Order::setRefund`

7.1.64 [0.8.1] - 2022-01-08

Added

- Add new property for percentage information to discount line-item. New methods: `\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\LineItem\Discount::isAbsolute`, `\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\LineItem\Discount::getPercentage`, `\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\LineItem\Discount::setPercentage`
- Add new property for relation to affected line-items to discount line-item. New methods: `\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\LineItem\Discount::getRelatedLineItems`, `\Heptacom\HeptaConnect\Dataset\Ecommerce\Order\LineItem\Discount::setRelatedLineItems`

7.1.65 [0.9.7.0] - 2024-02-10

Added

- Add method `\Heptacom\HeptaConnect\Portal\Base\Builder\Builder\EmitterBuilder::priority` to sort flow component within the stack
- Add method `\Heptacom\HeptaConnect\Portal\Base\Builder\Builder\ExplorerBuilder::priority` to sort flow component within the stack
- Add method `\Heptacom\HeptaConnect\Portal\Base\Builder\Builder\HttpHandlerBuilder::priority` to sort flow component within the stack
- Add method `\Heptacom\HeptaConnect\Portal\Base\Builder\Builder\ReceiverBuilder::priority` to sort flow component within the stack
- Add method `\Heptacom\HeptaConnect\Portal\Base\Builder\Builder\StatusReporterBuilder::priority` to sort flow component within the stack
- Add method `\Heptacom\HeptaConnect\Portal\Base\Builder\Token\EmitterToken::getPriority` and `\Heptacom\HeptaConnect\Portal\Base\Builder\Token\EmitterToken::setPriority` to sort flow component within the stack
- Add method `\Heptacom\HeptaConnect\Portal\Base\Builder\Token\ExplorerToken::getPriority` and `\Heptacom\HeptaConnect\Portal\Base\Builder\Token\ExplorerToken::setPriority` to sort flow component within the stack
- Add method `\Heptacom\HeptaConnect\Portal\Base\Builder\Token\HttpHandlerToken::getPriority` and `\Heptacom\HeptaConnect\Portal\Base\Builder\Token\HttpHandlerToken::setPriority` to sort flow component within the stack
- Add method `\Heptacom\HeptaConnect\Portal\Base\Builder\Token\ReceiverToken::getPriority` and `\Heptacom\HeptaConnect\Portal\Base\Builder\Token\ReceiverToken::setPriority` to sort flow component within the stack
- Add method `\Heptacom\HeptaConnect\Portal\Base\Builder\Token\StatusReporterToken::getPriority` and `\Heptacom\HeptaConnect\Portal\Base\Builder\Token\StatusReporterToken::setPriority` to sort flow component within the stack
- Add method `\Heptacom\HeptaConnect\Portal\Base\Builder\FlowComponent::setDefaultPriority` to set default position for flow components within the stack per source package
- Add method `\Heptacom\HeptaConnect\Portal\Base\Portal\Contract\PackageContract::getDefaultFlowComponentPriority` to set default position for flow components within the stack per source package

Changed

- Change `\Heptacom\HeptaConnect\Portal\Base\Support\Contract\DeepObjectIteratorContract::iterate` to resolve nested iterables deferred instead of in the moment they are found to lower memory peaks during larger object inspection

7.1.66 [0.9.6.0] - 2023-07-10

Fixed

- Fix emission check in implementation of `\Heptacom\HeptaConnect\Portal\Base\Support\Contract\EntityStatusContract::isMappedByEmitter`

7.1.67 [0.9.5.0] - 2023-05-27

Added

- Add `\Heptacom\HeptaConnect\Portal\Base\Portal\PackageCollection` as collection class for `\Heptacom\HeptaConnect\Portal\Base\Portal\Contract\PackageContract`
- Add service `Heptacom\HeptaConnect\Portal\Base\Portal\PackageCollection` to portal-container, containing the portal, all portal-extensions and all packages involved in building the container
- Add service `Psr\Http\Message\ServerRequestFactoryInterface` to portal-container
- Add service `Psr\Http\Message\UploadedFileFactoryInterface` to portal-container

- Add service `Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\HttpKernelInterface` to portal-container to execute a `\Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\HttpHandlerStackInterface` from inside a portal
- Add method `\Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\HttpHandleContextInterface::forward` to provide a guided usage of `\Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\HttpKernelInterface`
- Add service `Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\Psr7MessageMultiPartFormDataBuilderInterface` to build HTTP payloads for multipart messages

Changed

- Allow handling of HTTP requests, even when no HTTP handler exists for the requested path. This means, middlewares for HTTP handlers will run for every request.
- Add constant `\Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\HttpHandleContextInterface::REQUEST_ATTRIBUTE_IS_STACK_EMPTY` to identify an attribute in `\Psr\Http\Message\ServerRequestInterface` objects. This attribute holds a value, that indicates whether the related `\Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\HttpHandlerStackInterface` is empty.

Fixed

- Remove a step in building a portal-container that would remove all services that extend `\Heptacom\HeptaConnect\Portal\Base\Portal\Contract\PackageContract`

7.1.68 [0.9.4.0] - 2023-03-04

Added

- Add composer dependency `symfony/config: ^4.4 || ^5.0` and `symfony/dependency-injection: ^4.4 || ^5.0`
- Add `\Heptacom\HeptaConnect\Portal\Base\Portal\Contract\PackageContract` as base class for additional packages, other than portals and portal extensions
- Add `\Heptacom\HeptaConnect\Portal\Base\Portal\Contract\PackageContract::buildContainer` allowing packages to influence the build-process of the portal-container
- Add `\Heptacom\HeptaConnect\Portal\Base\Portal\Contract\PackageContract::getAdditionalPackages` allowing packages to provide additional packages. These packages may also influence the build-process of the portal-container.
- Add `\Heptacom\HeptaConnect\Portal\Base\Portal\Contract\PackageContract::registerContainerFile` allowing packages to automatically register their service definition files (e. g. `Resources/config/services.xml`)
- Add `\Heptacom\HeptaConnect\Portal\Base\Portal\Exception\DelegatingLoaderLoadException` for when a service definition file cannot be loaded
- Add exception code `1674923696` for when a service definition file cannot be loaded
- Add interface `\Heptacom\HeptaConnect\Portal\Base\FlowComponent\Contract\FlowComponentStackIdentifierInterface` to identify flow component stack identifier and all their commonly shared features
- Add class `\Heptacom\HeptaConnect\Portal\Base\Web\Http\HttpHandlerStackIdentifier` to hold the identifying components of an HTTP handler stack being the portal node key and served path
- Add class `\Heptacom\HeptaConnect\Portal\Base\Web\Http\ServerRequestCycle` to hold a server request and response, that correspond to a single HTTP request/response cycle
- Add service of `\Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\Psr7MessageCurlShellFormatterContract` implementing `\Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\Psr7MessageFormatterContract` to format HTTP messages described in PSR-7 into cURL shell commands
- Add service of `\Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\Psr7MessageRawHttpFormatterContract` implementing `\Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\Psr7MessageFormatterContract` as default provider to format HTTP messages described in PSR-7 into raw HTTP traffic, that can be used with TCP networking tools

Deprecated

- Deprecate and discourage usage of `\Heptacom\HeptaConnect\Dataset\Base\Contract\DeferralAwareInterface` as it has not been a practical solution to defer closure execution in a different process
- Deprecate extending method `\Heptacom\HeptaConnect\Portal\Base\Portal\Contract\PackageContract::__construct` as this method will become final in version 0.10

7.1.69 [0.9.3.0] - 2022-11-26

Added

- Add service of `\Heptacom\HeptaConnect\Portal\Base\File\Filesystem\Contract\FilesystemInterface` to the portal node container to interact with filesystem abstraction
- Add exception `\Heptacom\HeptaConnect\Portal\Base\File\Filesystem\Exception\UnexpectedFormatOfUriException` to indicate usage unexpected parameters with `\Heptacom\HeptaConnect\Portal\Base\File\Filesystem\Contract\FilesystemInterface`

Deprecated

- Deprecate service `League\Flysystem\FilesystemInterface` in the portal node container. Use `\Heptacom\HeptaConnect\Portal\Base\File\Filesystem\Contract\FilesystemInterface` in combination with native stream functions like `fopen`, `fread`, `fwrite`, `fclose`, `file_get_contents` and `file_put_contents` instead

7.1.70 [0.9.2.0] - 2022-10-16

Added

- Add `\Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\HttpClientMiddlewareInterface`. Every service implementing this interface will automatically be tagged with `heptaconnect.http.client.middleware`. Middlewares will be executed for every outbound HTTP request via the `\Psr\Http\Client\ClientInterface`.
- Add composer dependency `psr/http-server-handler: ^1.0` and `psr/http-server-middleware: ^1.0` to support PSR-15 middlewares for HTTP handlers. Every service implementing `\Psr\Http\Server\MiddlewareInterface` will automatically be tagged with `heptaconnect.http.handler.middleware`. Middlewares will be executed for every inbound HTTP request via `\Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\HttpHandlerContract`.

Fixed

- Add composer dependency on `psr/http-client: ^1.0`

7.1.71 [0.9.1.0] - 2022-08-15

Added

- Extract similarities of `\Heptacom\HeptaConnect\Portal\Base\Portal\Contract\PortalContract` and `\Heptacom\HeptaConnect\Portal\Base\Portal\Contract\PortalExtensionContract` into a new common base class `\Heptacom\HeptaConnect\Portal\Base\Portal\Contract\PackageContract`
- Add `\Heptacom\HeptaConnect\Portal\Base\Portal\Contract\PackageContract::getContainerExcludedClasses` to allow portals and portal extensions to add and remove automatically excluded classes from container auto-prototyping

Deprecated

- Deprecate `\Heptacom\HeptaConnect\Portal\Base\Portal\Contract\PathMethodsTrait` as content will be moved to `\Heptacom\HeptaConnect\Portal\Base\Portal\Contract\PackageContract` without replacement trait

Fixed

- Change order of stack handling and remove fallback value for the reported topic in `\Heptacom\HeptaConnect\Portal\Base\StatusReporting\Contract\StatusReporterContract::report`

7.1.72 [0.9.0.0] - 2022-04-02

Added

- Add structure to store code origin data in `\Heptacom\HeptaConnect\Portal\Base\FlowComponent\CodeOrigin`
- Add exception `\Heptacom\HeptaConnect\Portal\Base\FlowComponent\Exception\CodeOriginNotFound` to indicate issues when looking for code origins
- Add `\Heptacom\HeptaConnect\Portal\Base\Builder\Component\HttpHandler::getRunMethod` to expose configured callback for origin access reading
- Add `\Heptacom\HeptaConnect\Portal\Base\Builder\Component\HttpHandler::getOptionsMethod` to expose configured callback for origin access reading
- Add `\Heptacom\HeptaConnect\Portal\Base\Builder\Component\HttpHandler::getGetMethod` to expose configured callback for origin access reading
- Add `\Heptacom\HeptaConnect\Portal\Base\Builder\Component\HttpHandler::getPostMethod` to expose configured callback for origin access reading
- Add `\Heptacom\HeptaConnect\Portal\Base\Builder\Component\HttpHandler::getPatchMethod` to expose configured callback for origin access reading
- Add `\Heptacom\HeptaConnect\Portal\Base\Builder\Component\HttpHandler::getPutMethod` to expose configured callback for origin access reading
- Add `\Heptacom\HeptaConnect\Portal\Base\Builder\Component\HttpHandler::getDeleteMethod` to expose configured callback for origin access reading
- Add `\Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\HttpHandlerCodeOriginFinderInterface` to find code origin of `\Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\HttpHandlerContract`
- Add `\Heptacom\HeptaConnect\Portal\Base\Builder\Component\Emitter::getRunMethod` to expose configured callback for origin access reading
- Add `\Heptacom\HeptaConnect\Portal\Base\Builder\Component\Emitter::getBatchMethod` to expose configured callback for origin access reading
- Add `\Heptacom\HeptaConnect\Portal\Base\Builder\Component\Emitter::getExtendMethod` to expose configured callback for origin access reading
- Add `\Heptacom\HeptaConnect\Portal\Base\Emission\Contract\EmitterCodeOriginFinderInterface` to find code origin of `\Heptacom\HeptaConnect\Portal\Base\Emission\Contract\EmitterContract`
- Add `\Heptacom\HeptaConnect\Portal\Base\Builder\Component\Explorer::getRunMethod` to expose configured callback for origin access reading
- Add `\Heptacom\HeptaConnect\Portal\Base\Builder\Component\Explorer::getIsAllowedMethod` to expose configured callback for origin access reading
- Add `\Heptacom\HeptaConnect\Portal\Base\Exploration\Contract\ExplorerCodeOriginFinderInterface` to find code origin of `\Heptacom\HeptaConnect\Portal\Base\Exploration\Contract\ExplorerContract`
- Add `\Heptacom\HeptaConnect\Portal\Base\Builder\Component\Receiver::getRunMethod` to expose configured callback for origin access reading
- Add `\Heptacom\HeptaConnect\Portal\Base\Builder\Component\Receiver::getBatchMethod` to expose configured callback for origin access reading
- Add `\Heptacom\HeptaConnect\Portal\Base\Reception\Contract\ReceiverCodeOriginFinderInterface` to find code origin of `\Heptacom\HeptaConnect\Portal\Base\Reception\Contract\ReceiverContract`
- Add `\Heptacom\HeptaConnect\Portal\Base\Builder\Component\StatusReporter::getRunMethod` to expose configured callback for origin access reading

- Add `\Heptacom\HeptaConnect\Portal\Base\StatusReporting\Contract\StatusReporterCodeOriginFinderInterface` to find code origin of `\Heptacom\HeptaConnect\Portal\Base\StatusReporting\Contract\StatusReporterContract`
- Add method for portal extensions `\Heptacom\HeptaConnect\Portal\Base\Portal\Contract\PortalExtensionContract::isActiveByDefault` to allow for default activity state configuration
- Add supporting filter method `\Heptacom\HeptaConnect\Portal\Base\Portal\PortalExtensionCollection::bySupport` to filter portal extensions by their supported portal class
- Add new service `Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\HttpClientContract` to container as an alternative to `Psr\Http\Client\ClientInterface` with behaviour by configuration with e.g. `\Heptacom\HeptaConnect\Portal\Base\Web\Http\Support\DefaultRequestHeaders`
- Add class `\Heptacom\HeptaConnect\Portal\Base\File\FileReferenceFactoryContract` to create instances of `\Heptacom\HeptaConnect\Dataset\Base\File\FileReferenceContract`
- Add class `\Heptacom\HeptaConnect\Portal\Base\File\FileReferenceResolverContract` to resolve instances of `\Heptacom\HeptaConnect\Dataset\Base\File\FileReferenceContract` to instances of `\Heptacom\HeptaConnect\Portal\Base\File\ResolvedFileReferenceContract`
- Add class `\Heptacom\HeptaConnect\Portal\Base\File\ResolvedFileReferenceContract` to access file references in read operations
- Add new service `\Heptacom\HeptaConnect\Portal\Base\File\FileReferenceFactoryContract` to container to create file references from various sources
- Add new service `\Heptacom\HeptaConnect\Portal\Base\File\FileReferenceResolverContract` to container to resolve file references for read operations
- Add methods `\Heptacom\HeptaConnect\Portal\Base\StorageKey\Contract\PortalNodeKeyInterface::withAlias` and `\Heptacom\HeptaConnect\Portal\Base\StorageKey\Contract\PortalNodeKeyInterface::withoutAlias` to flag a portal node key to prefer the display as alias or storage key
- Make `$this` available in closures for short-notation flow-components with `\Heptacom\HeptaConnect\Portal\Base\Builder\FlowComponent`

Changed

- Use container tags `heptaconnect.flow_component.status_reporter_source`, `heptaconnect.flow_component.emitter_source`, `heptaconnect.flow_component.explorer_source`, `heptaconnect.flow_component.receiver_source`, `heptaconnect.flow_component.web_http_handler_source` instead of `heptaconnect.flow_component.emitter`, `heptaconnect.flow_component.emitter_decorator`, `heptaconnect.flow_component.explorer`, `heptaconnect.flow_component.explorer_decorator`, `heptaconnect.flow_component.receiver`, `heptaconnect.flow_component.receiver_decorator` and `heptaconnect.flow_component.web_http_handler` to collect flow component services
- Short-noted flow components load on first flow component usage instead on container building
- Use instance of `\Heptacom\HeptaConnect\Portal\Base\Emission\Contract\EmitterContract` in log context instead of its class in the message in `\Heptacom\HeptaConnect\Portal\Base\Emission\EmitterStack::next`
- Use instance of `\Heptacom\HeptaConnect\Portal\Base\Exploration\Contract\ExplorerContract` in log context instead of its class in the message in `\Heptacom\HeptaConnect\Portal\Base\Exploration\ExplorerStack::next`
- Use instance of `\Heptacom\HeptaConnect\Portal\Base\Reception\Contract\ReceiverContract` in log context instead of its class in the message in `\Heptacom\HeptaConnect\Portal\Base\Reception\ReceiverStack::next`
- Use instance of `\Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\HttpHandlerContract` in log context instead of its class in the message in `\Heptacom\HeptaConnect\Portal\Base\Web\Http\HttpHandlerStack::next`
- Add dependency to `\Psr\Log\LoggerInterface` into `\Heptacom\HeptaConnect\Portal\Base\StatusReporting\StatusReporterStack` to log instance of `\Heptacom\HeptaConnect\Portal\Base\StatusReporting\Contract\StatusReporterContract::next`
- Set `array-key` type to return of `\Heptacom\HeptaConnect\Portal\Base\Emission\EmitterCollection::bySupport`, `\Heptacom\HeptaConnect\Portal\Base\Exploration\ExplorerCollection::bySupport`, `\Heptacom\HeptaConnect\Portal\Base\Reception\ReceiverCollection::bySupport`, `\Heptacom\HeptaConnect\Portal\Base\StatusReporting\StatusReporterCollection::bySupportedTopic` and `\Heptacom\HeptaConnect\Portal\Base\Web\Http\HttpHandlerCollection::bySupport` to `int`

- Add final modifier to `\Heptacom\HeptaConnect\Portal\Base\Builder\Component\Emitter`, `\Heptacom\HeptaConnect\Portal\Base\Builder\Component\Explorer`, `\Heptacom\HeptaConnect\Portal\Base\Builder\Component\HttpHandler`, `\Heptacom\HeptaConnect\Portal\Base\Builder\Component\Receiver`, `\Heptacom\HeptaConnect\Portal\Base\Builder\Component\StatusReporter`, `\Heptacom\HeptaConnect\Portal\Base\Emission\EmitterStack`, `\Heptacom\HeptaConnect\Portal\Base\Exploration\ExplorerStack`, `\Heptacom\HeptaConnect\Portal\Base\Mapping\MappingComponentStruct`, `\Heptacom\HeptaConnect\Portal\Base\Profiling\NullProfiler`, `\Heptacom\HeptaConnect\Portal\Base\Reception\ReceiverStack`, `\Heptacom\HeptaConnect\Portal\Base\StatusReporting\StatusReporterStack` and `\Heptacom\HeptaConnect\Portal\Base\Web\Http\HttpHandlerStack` to ensure correct usage of implementation. Decoration by their interfaces or base classes is still possible

Removed

- Remove container service ids `Heptacom\HeptaConnect\Portal\Base\Emission\EmitterCollection`, `Heptacom\HeptaConnect\Portal\Base\Emission\EmitterCollection.decorator`, `Heptacom\HeptaConnect\Portal\Base\Exploration\ExplorerCollection`, `Heptacom\HeptaConnect\Portal\Base\Exploration\ExplorerCollection.decorator`, `Heptacom\HeptaConnect\Portal\Base\StatusReporting\StatusReporterCollection`, `Heptacom\HeptaConnect\Portal\Base\Reception\ReceiverCollection`, `Heptacom\HeptaConnect\Portal\Base\Reception\ReceiverCollection.decorator`, `Heptacom\HeptaConnect\Portal\Base\Web\Http\HttpHandlerCollection` and `Heptacom\HeptaConnect\Portal\Base\Web\Http\HttpHandlerCollection.decorator` due to refactoring of flow component stack building
- Remove contracts and exceptions `\Heptacom\HeptaConnect\Portal\Base\Cronjob\Contract\CronjobServiceInterface`, `\Heptacom\HeptaConnect\Portal\Base\Cronjob\Contract\CronjobRunInterface`, `\Heptacom\HeptaConnect\Portal\Base\Cronjob\Contract\CronjobInterface`, `\Heptacom\HeptaConnect\Portal\Base\Cronjob\Contract\CronjobHandlerContract`, `\Heptacom\HeptaConnect\Portal\Base\Cronjob\Contract\CronjobContextInterface`, `\Heptacom\HeptaConnect\Portal\Base\Cronjob\Exception\InvalidCronExpressionException`, `\Heptacom\HeptaConnect\Portal\Base\StorageKey\Contract\CronjobKeyInterface` and `\Heptacom\HeptaConnect\Portal\Base\StorageKey\Contract\CronjobRunKeyInterface` as the feature of cronjobs in its current implementation is removed
- Remove deprecated methods `\Heptacom\HeptaConnect\Portal\Base\Portal\Contract\PortalStorageInterface::canSet` and `\Heptacom\HeptaConnect\Portal\Base\Portal\Contract\PortalStorageInterface::canGet`
- Remove unused `\Heptacom\HeptaConnect\Portal\Base\Mapping\TypedMappedDatasetEntityCollection`
- Remove deprecated method `Heptacom\HeptaConnect\Portal\Base\Publication\Contract\PublisherInterface::publish`
- Remove unused `\Heptacom\HeptaConnect\Portal\Base\StorageKey\Contract\MappingKeyInterface` and `\Heptacom\HeptaConnect\Portal\Base\StorageKey\MappingKeyCollection`
- Move unused `\Heptacom\HeptaConnect\Portal\Base\StorageKey\Contract\RouteKeyInterface` and `\Heptacom\HeptaConnect\Portal\Base\StorageKey\RouteKeyCollection` to package `heptacom/heptaconnect-storage-base` as `\Heptacom\HeptaConnect\Storage\Base\Contract\RouteKeyInterface` and `\Heptacom\HeptaConnect\Storage\Base\RouteKeyCollection`

st changes

7.1.73 [0.8.4] - 2021-12-16

Removed

- Remove the code for unit tests, configuration for style checks as well as the Makefile

7.1.74 [0.8.0] - 2021-11-22

Added

- Add composer dependency on `ext-mbstring:*`
- Add composer dependency on `psr/event-dispatcher:^1.0`
- Add post-processing data bag class `\Heptacom\HeptaConnect\Portal\Base\Reception\Support\PostProcessorDataBag`
- Add method `\Heptacom\HeptaConnect\Portal\Base\Reception\Contract\ReceiveContextInterface::getEventDispatcher` for reception event processing
- Add method `\Heptacom\HeptaConnect\Portal\Base\Reception\Contract\ReceiveContextInterface::getPostProcessingBag` to access post-processing data bag
- Add exception code `1636887426` to `\Heptacom\HeptaConnect\Portal\Base\Serialization\Contract\SerializableStream::copy` when source stream is invalid
- Add exception code `1636887427` to `\Heptacom\HeptaConnect\Portal\Base\Serialization\Contract\SerializableStream::copy` when source stream can't be read from
- Add exception code `1636887428` to `\Heptacom\HeptaConnect\Portal\Base\Serialization\Contract\SerializableStream::copy` when result stream can't be created
- Add exception code `1636887429` to `\Heptacom\HeptaConnect\Portal\Base\Serialization\Contract\SerializableStream::copy` when interim stream can't be created
- Add new flow component `\Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\HttpHandlerContract` and `\Heptacom\HeptaConnect\Portal\Base\Web\Http\HttpHandlerCollection`
- Add interface `\Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\HttpHandleContextInterface` for new flow component
- Add interface `\Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\HttpHandlerStackInterface` and implementation `\Heptacom\HeptaConnect\Portal\Base\Web\Http\HttpHandlerStack` for new flow component
- Add log message code `1636735335` to `\Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\HttpHandlerContract::handleNext` when execution of the next handler failed
- Add log message code `1636735336` to `\Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\HttpHandlerContract::handleCurrent` when execution of the current handler failed
- Add `\Heptacom\HeptaConnect\Portal\Base\Builder\FlowComponent::httpHandler`, `\Heptacom\HeptaConnect\Portal\Base\Builder\FlowComponent::buildHttpHandlers`, `\Heptacom\HeptaConnect\Portal\Base\Builder\Component\HttpHandler`, `\Heptacom\HeptaConnect\Portal\Base\Builder\Token\HttpHandlerToken` and `\Heptacom\HeptaConnect\Portal\Base\Builder\Builder\HttpHandlerBuilder` to allow short notation for new flow component `\Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\HttpHandlerContract`
- Add log message code `1636791700` to `\Heptacom\HeptaConnect\Portal\Base\Builder\FlowComponent::buildHttpHandlers`, `\Heptacom\HeptaConnect\Portal\Base\Builder\FlowComponent::buildReceivers` and `\Heptacom\HeptaConnect\Portal\Base\Builder\FlowComponent::buildEmitters` when building flow components and having a configuration conflict
- Add `\Heptacom\HeptaConnect\Portal\Base\Web\Http\HttpHandlerUrlProviderInterface` to resolve URLs for flow component `\Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\HttpHandlerContract` paths
- Add exception `\Heptacom\HeptaConnect\Portal\Base\Builder\Exception\InvalidResultException` to group cases when short-noted closures are return incorrect values
- Add exception code `1637017868` to `\Heptacom\HeptaConnect\Portal\Base\Builder\Component\Emitter::batch` when short-noted batch method returns an invalid value in iteration
- Add exception code `1637017869` to `\Heptacom\HeptaConnect\Portal\Base\Builder\Component\Emitter::batch` when short-noted batch method returns invalid value

- Add exception code `1637017870` to `\Heptacom\HeptaConnect\Portal\Base\Builder\Component\Emitter::run` when short-noted run method returns invalid value
- Add exception code `1637017871` to `\Heptacom\HeptaConnect\Portal\Base\Builder\Component\Emitter::extend` when short-noted extend method returns invalid value
- Add exception code `1637034100` to `\Heptacom\HeptaConnect\Portal\Base\Builder\Component\Explorer::run` when short-noted run method returns an invalid value in iteration
- Add exception code `1637034101` to `\Heptacom\HeptaConnect\Portal\Base\Builder\Component\Explorer::run` when short-noted run method returns invalid value
- Add exception code `1637034102` to `\Heptacom\HeptaConnect\Portal\Base\Builder\Component\Explorer::isAllowed` when short-noted isAllowed method returns invalid value
- Add exception code `1637440327` to `\Heptacom\HeptaConnect\Portal\Base\Builder\Component\HttpHandler` when any short-noted method returns invalid value
- Add exception code `1637036888` to `\Heptacom\HeptaConnect\Portal\Base\Builder\Component\StatusReporter::run` when short-noted run method returns invalid value

Changed

- Change parameter name of `\Heptacom\HeptaConnect\Portal\Base\Publication\Contract\PublisherInterface::publish` from `$datasetEntityClassName` to `$entityType`
- Change parameter name of `\Heptacom\HeptaConnect\Portal\Base\Emission\Contract\EmitContextInterface::markAsFailed` from `$datasetEntityClassName` to `$entityType`
- Change parameter name of `\Heptacom\HeptaConnect\Portal\Base\Mapping\MappingComponentCollection::filterByEntityType` from `$datasetEntityClassName` to `$entityType`
- Change parameter name of `\Heptacom\HeptaConnect\Portal\Base\Mapping\MappingComponentStruct::__construct` from `$datasetEntityClassName` to `$entityType`
- Change parameter name of `\Heptacom\HeptaConnect\Portal\Base\Emission\EmitterCollection::bySupport` from `$entityClassName` to `$entityType`
- Change parameter name of `\Heptacom\HeptaConnect\Portal\Base\Emission\EmitterStack::__construct` from `$entityClassName` to `$entityType`
- Change parameter name of `\Heptacom\HeptaConnect\Portal\Base\Exploration\ExplorerCollection::bySupport` from `$entityClassName` to `$entityType`
- Change parameter name of `\Heptacom\HeptaConnect\Portal\Base\Reception\ReceiverCollection::bySupport` from `$entityClassName` to `$entityType`
- Change method name from `\Heptacom\HeptaConnect\Portal\Base\Mapping\MappingComponentStruct::getDatasetEntityClassName` to `\Heptacom\HeptaConnect\Portal\Base\Mapping\MappingComponentStruct::getEntityType`
- Change method name from `\Heptacom\HeptaConnect\Portal\Base\Mapping\Contract\MappingComponentStructContract::getDatasetEntityClassName` to `\Heptacom\HeptaConnect\Portal\Base\Mapping\Contract\MappingComponentStructContract::getEntityType`
- Change method name from `\Heptacom\HeptaConnect\Portal\Base\Mapping\Contract\MappingInterface::getDatasetEntityClassName` to `\Heptacom\HeptaConnect\Portal\Base\Mapping\Contract\MappingInterface::getEntityType`
- Change method name from `\Heptacom\HeptaConnect\Portal\Base\Mapping\MappingComponentCollection::getDatasetEntityClassNames` to `\Heptacom\HeptaConnect\Portal\Base\Mapping\MappingComponentCollection::getEntityTypes`
- Change method name from `\Heptacom\HeptaConnect\Portal\Base\Portal\Contract\RouteInterface::getEntityClassName` to `\Heptacom\HeptaConnect\Portal\Base\Portal\Contract\RouteInterface::getEntityType`
- As `\Closure` has a more defined interface for analyzing compared to `callable` and the expected use-case for short-noted flow components are anonymous functions, the return types changed from `callable` to `\Closure` in `\Heptacom\HeptaConnect\Portal\Base\Builder\Token\EmitterToken::getBatch`, `\Heptacom\HeptaConnect\Portal\Base\Builder\Token\EmitterToken::getRun`, `\Heptacom\HeptaConnect\Portal\Base\Builder\Token\EmitterToken::getExtend`, `\Heptacom\HeptaConnect\Portal\Base\Builder\Token\ExplorerToken::getRun`, `\Heptacom\HeptaConnect\Portal\Base\Builder\Token\ExplorerToken::getIsAllowed`, `\Heptacom\HeptaConnect\Portal\Base\Builder\Token\ReceiverToken::getBatch`,

`\Heptacom\HeptaConnect\Portal\Base\Builder\Token\ReceiverToken::getRun` and
`\Heptacom\HeptaConnect\Portal\Base\Builder\Token\StatusReporterToken::getRun`

- As `\Closure` has a more defined interface for analyzing compared to `callable` and the expected use-case for short-noted flow components are anonymous functions, the parameter types changed from `callable` to `\Closure` in
`\Heptacom\HeptaConnect\Portal\Base\Builder\Builder\EmitterBuilder::batch`,
`\Heptacom\HeptaConnect\Portal\Base\Builder\Builder\EmitterBuilder::run`,
`\Heptacom\HeptaConnect\Portal\Base\Builder\Builder\EmitterBuilder::extend`,
`\Heptacom\HeptaConnect\Portal\Base\Builder\Builder\ExplorerBuilder::run`,
`\Heptacom\HeptaConnect\Portal\Base\Builder\Builder\ExplorerBuilder::isAllowed`,
`\Heptacom\HeptaConnect\Portal\Base\Builder\Builder\ReceiverBuilder::batch`,
`\Heptacom\HeptaConnect\Portal\Base\Builder\Builder\ReceiverBuilder::run`,
`\Heptacom\HeptaConnect\Portal\Base\Builder\Builder\StatusReporterBuilder::run`,
`\Heptacom\HeptaConnect\Portal\Base\Builder\FlowComponent::explorer`,
`\Heptacom\HeptaConnect\Portal\Base\Builder\FlowComponent::emitter`,
`\Heptacom\HeptaConnect\Portal\Base\Builder\FlowComponent::receiver`,
`\Heptacom\HeptaConnect\Portal\Base\Builder\FlowComponent::statusReporter`,
`\Heptacom\HeptaConnect\Portal\Base\Builder\Token\EmitterToken::setBatch`,
`\Heptacom\HeptaConnect\Portal\Base\Builder\Token\EmitterToken::setRun`,
`\Heptacom\HeptaConnect\Portal\Base\Builder\Token\EmitterToken::setExtend`,
`\Heptacom\HeptaConnect\Portal\Base\Builder\Token\ExplorerToken::setRun`,
`\Heptacom\HeptaConnect\Portal\Base\Builder\Token\ExplorerToken::setIsAllowed`,
`\Heptacom\HeptaConnect\Portal\Base\Builder\Token\ReceiverToken::setBatch`,
`\Heptacom\HeptaConnect\Portal\Base\Builder\Token\ReceiverToken::setRun`,
`\Heptacom\HeptaConnect\Portal\Base\Builder\Token\StatusReporterToken::setRun` and
`\Heptacom\HeptaConnect\Portal\Base\Builder\ResolveArgumentsTrait::resolveArguments`
- Add throwing of exception `\Heptacom\HeptaConnect\Portal\Base\Serialization\Exception\StreamCopyException` to
`\Heptacom\HeptaConnect\Portal\Base\Serialization\Contract\SerializableStream::copy`

Removed

- Remove `\Heptacom\HeptaConnect\Portal\Base\StorageKey\Contract\WebhookKeyInterface`
- Remove `\Heptacom\HeptaConnect\Portal\Base\StorageKey\WebhookKeyCollection`
- Remove `\Heptacom\HeptaConnect\Portal\Base\Webhook\Contract\WebhookContextInterface`
- Remove `\Heptacom\HeptaConnect\Portal\Base\Webhook\Contract\WebhookHandlerContract`
- Remove `\Heptacom\HeptaConnect\Portal\Base\Webhook\Contract\WebhookInterface` in favour of new flow component
`\Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\HttpHandlerContract`
- Remove `\Heptacom\HeptaConnect\Portal\Base\Webhook\Contract\WebhookServiceInterface` in favour of new flow component
`\Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\HttpHandlerContract` and
`\Heptacom\HeptaConnect\Portal\Base\Web\Http\HttpHandlerUrlProviderInterface`
- Remove unused `\Heptacom\HeptaConnect\Portal\Base\Portal\Contract\RouteInterface`

Fixed

- Change type hint from `string` to `class-string<\Heptacom\HeptaConnect\Dataset\Base\Contract\DatasetEntityContract>` for parameters in
`\Heptacom\HeptaConnect\Portal\Base\Builder\FlowComponent::explorer`,
`\Heptacom\HeptaConnect\Portal\Base\Builder\FlowComponent::emitter`,
`\Heptacom\HeptaConnect\Portal\Base\Builder\FlowComponent::receiver`,
`\Heptacom\HeptaConnect\Portal\Base\Builder\Token\EmitterToken::__construct`,
`\Heptacom\HeptaConnect\Portal\Base\Builder\Token\ExplorerToken::__construct` and
`\Heptacom\HeptaConnect\Portal\Base\Builder\Token\ReceiverToken::__construct`
- Change type hint from `string` to `class-string<\Heptacom\HeptaConnect\Dataset\Base\Contract\DatasetEntityContract>` for return type in
`\Heptacom\HeptaConnect\Portal\Base\Builder\Token\EmitterToken::getType`,
`\Heptacom\HeptaConnect\Portal\Base\Builder\Token\ExplorerToken::getType` and
`\Heptacom\HeptaConnect\Portal\Base\Builder\Token\ReceiverToken::getType`
- Allow missing types in short-noted flow components that are resolved by name by changing `string $parameterType` to `?string $parameterType` in function arguments in `\Heptacom\HeptaConnect\Portal\Base\Builder\ResolveArgumentsTrait` and their usages

- Fixe return type hint on `\Heptacom\HeptaConnect\Portal\Base\Reception\Contract\ReceiverStackInterface::next` to return an iterable of `\Heptacom\HeptaConnect\Dataset\Base\Contract\DatasetEntityContract` instead of `\Heptacom\HeptaConnect\Portal\Base\Mapping\Contract\MappingInterface` and therefore returns like `\Heptacom\HeptaConnect\Portal\Base\Reception\Contract\ReceiverContract::receive`

7.1.75 [0.7.0] - 2021-09-25

Added

- Add `\Heptacom\HeptaConnect\Portal\Base\Portal\Contract\PortalStorageInterface::delete` as replacement for `\Heptacom\HeptaConnect\Portal\Base\Portal\Contract\PortalStorageInterface::unset`. This method returns a boolean instead of throwing exceptions.
- Add composer dependency on `psr/simple-cache:^1.0`

Changed

- `\Heptacom\HeptaConnect\Portal\Base\Support\Contract\DeepObjectIteratorContract::iterate` caches object iteration strategies to improve performance
- `\Heptacom\HeptaConnect\Portal\Base\Portal\Contract\PortalStorageInterface` implements `\Psr\SimpleCache\CacheInterface`.
- `\Heptacom\HeptaConnect\Portal\Base\Portal\Contract\PortalStorageInterface::set` no longer throws exceptions on failure but returns a boolean instead.

Removed

- `\Heptacom\HeptaConnect\Portal\Base\Portal\Contract\PortalStorageInterface::unset` has been replaced by `\Heptacom\HeptaConnect\Portal\Base\Portal\Contract\PortalStorageInterface::delete`.

Fixed

- `\Heptacom\HeptaConnect\Portal\Base\Builder\FlowComponent::reset` now cleans up status reporter building instructions that got previously registered with `\Heptacom\HeptaConnect\Portal\Base\Builder\FlowComponent::statusReporter`
- `\Heptacom\HeptaConnect\Portal\Base\Support\Contract\DeepObjectIteratorContract::iterate` drops usage of `\spl_object_hash` to not break on garbage collection

7.1.76 [0.9.0.2] - 2022-06-01

Fixed

- Fixed address splitting on customer creation in `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Receiver\CustomerReceiver::getAddress` to satisfy the shopware standard address representation regarding street and house number.

7.1.77 [0.9.0.1] - 2022-04-23

Fixed

- Fix portal node service container extension to work with both portal FQCNs `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\LocalShopwarePlatformPortal` and the deprecated `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Portal`

7.1.78 [0.9.0.0] - 2022-04-05

Added

- Interpret `\Heptacom\HeptaConnect\Dataset\Ecommerce\Product\Product::getMedias` to receive medias on products

Changed

- Use `\Heptacom\HeptaConnect\Dataset\Ecommerce\Media\Media::getFile` for transferring instead of `\Heptacom\HeptaConnect\Dataset\Ecommerce\Media\Media::getNormalizedStream`

Deprecated

- Deprecate `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Portal` as renamed to `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\LocalShopwarePlatformPortal`

Removed

- Remove support for `shopware/core: >=6.2.1 <6.4`

Fixed

- Product medias unpacked by `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Unpacker\ProductUnpacker` have position by appearance in the product entity

st changes

7.1.79 [0.8.2] - 2022-02-09

Fixed

- Fix function call on null in `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Emitter\OrderEmitter`.
- Fix compatibility with `shopware/core:^6.4` by supporting `\Shopware\Core\System\Currency\CurrencyEntity::setItemRounding` if it exists.

7.1.80 [0.8.1] - 2021-12-07

Fixed

- Fetch VAT-ID from customer instead of address in `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Packer\CustomerPacker::pack`

7.1.81 [0.8.0] - 2021-11-22

Added

- Add `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Support\LocaleMatcher` to centralize translation handling of incoming locale matching
- Add `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Unpacker\TranslatableUnpacker` to centralize translations payload generation of any `\Heptacom\HeptaConnect\Dataset\Base\Translatable\Contract\TranslatableInterface`
- Add log message code `1637342440` in `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Support\LocaleMatcher::match` when a locale code is tested against a Shopware language
- Add log message code `1637342441` in `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Support\LocaleMatcher::match` when a locale code could not be matched
- Add log message code `1637342442` in `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Support\LocaleMatcher::match` when a locale code could be matched to a unique Shopware language
- Add log message code `1637342443` in `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Support\LocaleMatcher::match` when a locale code could be matched against multiple other Shopware languages
- Add log message code `1637344184` in `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Unpacker\TranslatableUnpacker::unpack` when a translated value is tried to be applied but the language code could not be mapped to a Shopware language
- Add `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Unpacker\CategoryUnpacker` to unpack category data into Shopware API payload
- Add `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Unpacker\CustomerGroupUnpacker` to unpack customer group data into Shopware API payload
- Use `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Unpacker\TranslatableUnpacker` in `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Unpacker\MediaUnpacker` to support translations
- Use `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Unpacker\TranslatableUnpacker` in `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Unpacker\ManufacturerUnpacker` to support translations
- Use `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Unpacker\TranslatableUnpacker` in `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Unpacker\PropertyValueUnpacker` to support translations
- Use `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Unpacker\TranslatableUnpacker` in `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Unpacker\PropertyGroupUnpacker` to support translations
- Add `\Shopware\Core\System\Language\LanguageLoaderInterface` to portal node container
- Add compatibility in code for `ramsey/uuid: ^4` and therefore changed composer requirement to `ramsey/uuid: ^3.5 || ^4`

Changed

- Move `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Receiver\CategoryReceiver` into short notation
- Move `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Receiver\CustomerGroupReceiver` into short notation
- Change default value for configuration for `dal_indexing_mode` from `none` to `queue`

- Use `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Unpacker\TranslatableUnpacker` in `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Unpacker\UnitUnpacker` instead of `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Support\Translator`
- Use `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Unpacker\TranslatableUnpacker` in `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Unpacker\ProductUnpacker` instead of `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Support\TranslationLocaleCache` directly
- Use `\Shopware\Core\System\Language\LanguageLoaderInterface` in `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Support\TranslationLocaleCache` instead of `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Support\DalAccess`
- Remove `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Support\ExistingIdentifierCache` dependency from `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Unpacker\ManufacturerUnpacker`

Fixed

- Use fallback translations values for default language for the keys name and description in `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Unpacker\ProductUnpacker::unpackTranslations`

Removed

- Remove `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Support\Translator` in favour of `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Unpacker\TranslatableUnpacker` and `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Support\TranslationLocaleCache`

7.1.82 [0.7.0] - 2021-09-25

Added

- Add optional operation key `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Support\DalSyncer::upsert`, `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Support\DalSyncer::delete` for easier task recognition
- Add optional context parameter to `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Support\DalSyncer::flush` allowing an override of the used modified context
- Extract locale code caching from `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Support\Translator` into new service `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Support\TranslationLocaleCache`
- New protected method `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Unpacker\ProductUnpacker::unpackTranslations` adds support for translated product content. By default `name` and `description` is supported
- Add product property assignments in return value of `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Unpacker\ProductUnpacker::unpack`
- Add cleanup of previously imported product properties in `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Receiver\ProductReceiver`

Changed

- Improve memory usage and first call wall time of `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Support\ExistingIdentifierCache::getProductMediaId` by dropping id cache warmup
- Throw `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Support\Exception\DuplicateSyncOperationKeyPreventionException` with code `1632595313` when `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Support\DalSyncer::upsert`, `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Support\DalSyncer::delete` and `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Support\DalSyncer::push` have a duplicate mismatch
- Change return value of `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Support\DalSyncer::flush` from `self` to `\Shopware\Core\Framework\Api\Sync\SyncResult` to access pure sync api result
- `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Unpacker\ProductPriceUnpacker::unpack` now expects a price collection and a product identifier to generate product price rules more efficiently

Removed

- Remove unused `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Support\Translator::getIngredientTranslation`

- In favour of translations `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Unpacker\ProductUnpacker::unpack` no longer adds `name` and `description` in the payload root
- Remove `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Unpacker\ProductUnpacker::unpackPrices` due to complete extraction into `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Unpacker\ProductPriceUnpacker::unpack`

Fixed

- Change payload key that references property groups in `\Heptacom\HeptaConnect\Portal\LocalShopwarePlatform\Unpacker\PropertyValueUnpacker::unpack` to fix reception of `\Heptacom\HeptaConnect\Dataset\Ecommerce\Property\PropertyValue`

7.1.83 [1.1.0] - 2023-09-02

Added

- Add interface `\Heptacom\HeptaConnect\Package\Http\Components\HttpRequestCycleProfiling\Contract\HttpRequestCycleModifierInterface` to build components, that will be used to modify recorded HTTP request cycles
- Add methods `\Heptacom\HeptaConnect\Package\Http\Components\HttpRequestCycleProfiling\Contract\HttpRequestCycleCollector::withAddedModifier`, `\Heptacom\HeptaConnect\Package\Http\Components\HttpRequestCycleProfiling\Contract\HttpRequestCycleCollector::withoutModifiers`, `\Heptacom\HeptaConnect\Package\Http\Components\HttpRequestCycleProfiling\Contract\HttpRequestCycleCollector::getModifiers` to manage `\Heptacom\HeptaConnect\Package\Http\Components\HttpRequestCycleProfiling\Contract\HttpRequestCycleModifierInterface` that will be applied when collecting request cycles
- Add request cycle modifier class `\Heptacom\HeptaConnect\Package\Http\Components\HttpRequestCycleProfiling\Modifier\HeaderValueReplacingModifier` to replace header values using RegEx patterns
- Add request cycle modifier class `\Heptacom\HeptaConnect\Package\Http\Components\HttpRequestCycleProfiling\Modifier\RequestUrlModifier` to replace request URL parts using RegEx patterns
- Add base class `\Heptacom\HeptaConnect\Package\Http\Components\HttpRequestCycleProfiling\Modifier\AbstractBodyModifier` to build modifiers depending on the body's mimetype
- Add request cycle modifier class `\Heptacom\HeptaConnect\Package\Http\Components\HttpRequestCycleProfiling\Modifier\JsonBodyFormattingModifier` to format request and response bodies that are of mimetype `application/json`
- Add request cycle modifier class `\Heptacom\HeptaConnect\Package\Http\Components\HttpRequestCycleProfiling\Modifier\XmlBodyFormattingModifier` to format request and response bodies that are of mimetype `application/xml`
- Add composer suggestion to allow `\Heptacom\HeptaConnect\Package\Http\Components\HttpRequestCycleProfiling\Modifier\XmlBodyFormattingModifier` to work

Changed

- Prevent request URL modification in `\Heptacom\HeptaConnect\Package\Http\Components\HttpRequestCycleProfiling\HttpRequestCycleProfiler`, when collector has modifiers assigned

Deprecated

- Deprecate expected request URL modification in `\Heptacom\HeptaConnect\Package\Http\Components\HttpRequestCycleProfiling\HttpRequestCycleProfiler`. Expect to always add `\Heptacom\HeptaConnect\Package\Http\Components\HttpRequestCycleProfiling\Modifier\RequestUrlModifier` to collectors

7.1.84 [1.0.0] - 2023-06-10

Added

- Add composer dependency `heptacom/heptaconnect-portal-base: ^0.9.5` as `\Heptacom\HeptaConnect\Package\Http\HttpPackage` is a package and HTTP middleware `\Heptacom\HeptaConnect\Portal\Base\Web\Http\Contract\HttpClientMiddlewareInterface` is used
- Add composer dependencies `psr/http-client: ^1.0`, `psr/http-factory: ^1.0` and `psr/http-message: ^1.0 || ^2.0` as PSR-7 based HTTP messages are used
- Add composer dependencies `psr/event-dispatcher: ^1.0`, `symfony/event-dispatcher: ^5.0 || ^6.0` and `symfony/event-dispatcher-contracts: ^2.0 || ^3.0` as events are dispatched

- Add composer dependency `symfony/dependency-injection: ^5.0 || ^6.0` as compiler passes are used
- Add composer dependency `symfony/options-resolver: ^5.1 || ^6.0` as unstructured data is validated using the Symfony options resolver
- Introduce outbound HTTP cache feature using `\Heptacom\HeptaConnect\Package\Http\Components\HttpCache\HttpCache` based on `\Heptacom\HeptaConnect\Package\Http\Components\HttpCache\Contract\HttpCacheInterface`
- Add class `\Heptacom\HeptaConnect\Package\Http\Components\HttpCache\Psr7MessageSerializer` described by `\Heptacom\HeptaConnect\Package\Http\Components\HttpCache\Contract\Psr7MessageSerializerInterface` to serialize and deserialize PSR-7 messages for storing them in a cache
- Add event `\Heptacom\HeptaConnect\Package\Http\Components\HttpCache\Contract\Event\HttpCacheActiveEvent` to influence, whether a request cycle is cached
- Add event `\Heptacom\HeptaConnect\Package\Http\Components\HttpCache\Contract\Event\HttpCacheKeyEvent` to influence under which cache key a request is cached
- Add service tag `heptaconnect.http.client.middleware` to `Heptacom\HeptaConnect\Package\Http\Components\HttpCache\Contract\HttpCacheInterface` to ensure it is chained in outgoing HTTP communication
- Introduce HTTP request cycle profiling using `\Heptacom\HeptaConnect\Package\Http\Components\HttpRequestCycleProfiling\HttpRequestCycleProfiler` based on `\Heptacom\HeptaConnect\Package\Http\Components\HttpRequestCycleProfiling\Contract\HttpRequestCycleProfilerInterface` to control, which outgoing request cycles shall be measured and how the measurements will be processed
- Add struct `\Heptacom\HeptaConnect\Package\Http\Components\HttpRequestCycleProfiling\Contract\HttpRequestCycleCollector` to store `\Heptacom\HeptaConnect\Package\Http\Components\HttpRequestCycleProfiling\Contract\HttpRequestCycle` for each request in a request cycle measurement
- Add `\Heptacom\HeptaConnect\Package\Http\DependencyInjection\EventSubscriberTagCompilerPass` as class and into the container building to register any `\Symfony\Component\EventDispatcher\EventSubscriberInterface` as active subscriber
- Add service `Symfony\Component\EventDispatcher\EventDispatcherInterface` for class `\Symfony\Component\EventDispatcher\EventDispatcher` with aliases `event_dispatcher`, `Symfony\Contracts\EventDispatcher\EventDispatcherInterface` and `Psr\EventDispatcher\EventDispatcherInterface`

st changes

7.1.85 [1.0.2] - 2024-01-05

Fixed

- Fix path generation for cookies in `\Heptacom\HeptaConnect\Package\WebFrontend\Components\Session\SessionManager::alterResponse`

7.1.86 [1.0.1] - 2023-11-22

Fixed

- Fix login when no session was started yet

7.1.87 [1.0.0] - 2023-07-10

Added

- Require `php: >=8.0`
- Add composer dependency `ext-filter: *` to validate user input in PHP ini settings
- Add composer dependency `symfony/dependency-injection: ^5.0 || ^6.0` and `symfony/config: ^5.0 || ^6.0` as compiler passes, services.xml files and extensions are used
- Add composer dependency `heptacom/heptaconnect-portal-base: ^0.9.6` as `\Heptacom\HeptaConnect\Package\WebFrontend\WebFrontendPackage` is a package and different flow components are provided
- Add HEPTAconnect package class `\Heptacom\HeptaConnect\Package\WebFrontend\WebFrontendPackage`
- Add composer dependency `symfony/error-handler: ^5.0 || ^6.0` to provide human readable error pages
- Add composer dependencies `psr/http-factory: ^1.0`, `psr/http-message: ^1.0 || ^2.0`, `psr/http-server-handler: ^1.0` and `psr/http-server-middleware: ^1.0` as PSR-7 server requests are processed and responded
- Add HTTP middleware service `Heptacom\HeptaConnect\Package\WebFrontend\Components\ErrorHandler\HttpErrorHandlerMiddleware` with positive priority to catch exception as early as possible and render them as HTML
- Add composer dependency `heptacom/heptaconnect-dataset-base: ^0.9` to use collections and attachable structures
- Add collection service `Heptacom\HeptaConnect\Package\WebFrontend\Components\Notification\NotificationBag` holding `\Heptacom\HeptaConnect\Package\WebFrontend\Components\Notification\Notification` for rendering use
- Add composer dependency `ext-mbstring: *` to work with multibyte strings
- Add composer dependencies `twig/twig: ^3.0` and `twig/string-extra: ^3.0` to make use of the Twig templating engine
- Add service `Heptacom\HeptaConnect\Package\WebFrontend\Components\Template\Hierarchy\Contract\TemplateFinderInterface` implemented by `\Heptacom\HeptaConnect\Package\WebFrontend\Components\Template\Hierarchy\TemplateFinder` to find the next matching template to render in the next step
- Add class `\Heptacom\HeptaConnect\Package\WebFrontend\Components\Template\Hierarchy\TokenParserDecorator` to reuse existing token parser under a different name
- Add class `\Heptacom\HeptaConnect\Package\WebFrontend\Components\Template\Hierarchy\ExtendsTokenParser` as theme-aware implementation for Twig tag `extends`
- Add class `\Heptacom\HeptaConnect\Package\WebFrontend\Components\Template\Hierarchy\IncludeTokenParser` and `\Heptacom\HeptaConnect\Package\WebFrontend\Components\Template\Hierarchy\InheritedInclude` as theme-aware implementation for Twig tag `include`
- Add class `\Heptacom\HeptaConnect\Package\WebFrontend\Components\Template\Hierarchy\NodeExtension` as Twig extension to provide theme-awareness to Twig
- Add interface `\Heptacom\HeptaConnect\Package\WebFrontend\Components\Template\Contract\ThemeInterface` to identify themes and collect them in collection service `Heptacom\HeptaConnect\Package\WebFrontend\Components\Template\Contract\ThemeCollection`
- Add trait `\Heptacom\HeptaConnect\Package\WebFrontend\Components\Template\Utility\ThemePackageTrait` to implement `\Heptacom\HeptaConnect\Package\WebFrontend\Components\Template\Contract\ThemeInterface` for any `\Heptacom\HeptaConnect\Portal\Base\Portal\Contract\PackageContract` without any further code

- Add compiler pass `\Heptacom\HeptaConnect\Package\WebFrontend\DependencyInjection\TemplateTagCompilerPass` to collect themes and bring them in order
- Add compiler pass `\Heptacom\HeptaConnect\Package\WebFrontend\DependencyInjection\TwigExtensionTagCompilerPass` to collect all Twig extensions
- Add compiler pass `\Heptacom\HeptaConnect\Package\WebFrontend\DependencyInjection\RegisterSuggestedTwigExtensionsCompilerPass` to use the Twig Intl extension, when installed
- Add HTTP middleware service `Heptacom\HeptaConnect\Package\WebFrontend\Components\Template\AssetMiddleware` to serve any given path to an asset optimized for web browser caching
- Add Twig test `instanceof` with `\Heptacom\HeptaConnect\Package\WebFrontend\Components\Template\Extension\InstanceOfExtension` to allow for variable checks to be a certain type
- Add Twig test `numeric` with `\Heptacom\HeptaConnect\Package\WebFrontend\Components\Template\Extension\IsNumericExtension`
- Add Twig filter `urldecode` with `\Heptacom\HeptaConnect\Package\WebFrontend\Components\Template\Extension\UrlDecodeExtension` as counterpart to `urlencode`
- Add composer dependency `bentools/iterable-functions: >=1.4 <2` to simplify working with iterables
- Add factory service `Heptacom\HeptaConnect\Package\WebFrontend\Components\Template\Contract\TwigEnvironmentFactoryInterface` implemented by `\Heptacom\HeptaConnect\Package\WebFrontend\Components\Template\TwigEnvironmentFactory` to build common Twig environment instances
- Add base class `\Heptacom\HeptaConnect\Package\WebFrontend\DependencyInjection\AbstractFeature` for Symfony extensions, that are used to group code into features
- Add compiler pass `\Heptacom\HeptaConnect\Package\WebFrontend\DependencyInjection\ProvideContainerParameterForTwigEnvironmentCompilerPass` to pass feature configurations into the Twig template
- Add service `Heptacom\HeptaConnect\Package\WebFrontend\Components\Template\Feature\Debug\DebugTwigEnvironmentFactory` decorating `Heptacom\HeptaConnect\Package\WebFrontend\Components\Template\Contract\TwigEnvironmentFactoryInterface` to enable debugging features
- Add flow component `\Heptacom\HeptaConnect\Package\WebFrontend\Components\Template\Feature\Debug\DebugThemeStatusReporter` to debug theme functionalities
- Add feature class `\Heptacom\HeptaConnect\Package\WebFrontend\Components\Template\Feature\DebugFeature` to control template debugging
- Add Symfony extension `web_frontend_template_debug` configuration `enabled` to enable template debugging
- Add Symfony extension `web_frontend_template_debug` configuration `html_error_renderer` to fully render exceptions
- Add Twig cache implementation `\Heptacom\HeptaConnect\Package\WebFrontend\Components\Template\Feature\Cache\TwigCache`, that works different with temporary files
- Add service `Heptacom\HeptaConnect\Package\WebFrontend\Components\Template\Feature\Cache\CachePath` to handle Twig cache access
- Add flow component `\Heptacom\HeptaConnect\Package\WebFrontend\Components\Template\Feature\Cache\CacheClearCommand` to clear Twig cache
- Add service `Heptacom\HeptaConnect\Package\WebFrontend\Components\Template\Feature\Cache\CachedTwigEnvironmentFactory` decorating `Heptacom\HeptaConnect\Package\WebFrontend\Components\Template\Contract\TwigEnvironmentFactoryInterface` to enable caching features
- Add feature class `\Heptacom\HeptaConnect\Package\WebFrontend\Components\Template\Feature\CacheFeature` to control template caching
- Add Symfony extension `web_frontend_template_cache` configuration `enabled` to enable template caching
- Add theme `WebFrontendPackage` by class `\Heptacom\HeptaConnect\Package\WebFrontend\Components\BootstrapTheme\BootstrapTheme`
- Add editor theme component in `@WebFrontendPackage/ui/_base/component/editor.html.twig`, `@WebFrontendPackage/ui/_base/js/editor.js` and `@WebFrontendPackage/ui/_base/css/editor.css` to have simplified code editor
- Add notification theme component in `@WebFrontendPackage/ui/_base/component/notifications.html.twig` and `@WebFrontendPackage/ui/_base/js/notifications.js` to display notifications with Bootstrap toasts

- Add sidebar theme component in `@WebFrontendPackage/ui/_base/component/sidebar.html.twig`, `@WebFrontendPackage/ui/_base/js/sidebar.js` and `@WebFrontendPackage/ui/_base/css/sidebar.css` divided into `@WebFrontendPackage/ui/component/sidebar/header.html.twig` and `@WebFrontendPackage/ui/component/sidebar/scrollable-content.html.twig` of `@WebFrontendPackage/ui/component/sidebar/item.html.twig` for sidebar menu items
- Add dark mode appearance in `@WebFrontendPackage/ui/_base/js/appearance.js`
- Add left-sidebar page layout in `@WebFrontendPackage/ui/_base/layout.html.twig`
- Add HEPTAconnect icon asset in `src/Components/BootstrapTheme/Resources/public/icon/heptaconnect-logo.png`
- Add feature class `\Heptacom\HeptaConnect\Package\WebFrontend\Components\BootstrapThemeFeature` to control the Bootstrap 5 theme
- Add Symfony extension `web_frontend_bootstrap_theme` configuration `enabled` to enable the Bootstrap 5 theme
- Add composer dependency `psr/simple-cache": "^1.0` to use cache storages for sessions
- Add session storage class `\Heptacom\HeptaConnect\Package\WebFrontend\Components\Session\Session` described by `\Heptacom\HeptaConnect\Package\WebFrontend\Components\Session\Contract\SessionInterface`
- Add class `\Heptacom\HeptaConnect\Package\WebFrontend\Components\Session\SessionManager` described by `\Heptacom\HeptaConnect\Package\WebFrontend\Components\Session\Contract\SessionManagerInterface` to store sessions and access them from requests
- Add HTTP middleware service `Heptacom\HeptaConnect\Package\WebFrontend\Components\Session\SessionMiddleware` with a lower priority than `Heptacom\HeptaConnect\Package\WebFrontend\Components\Template\AssetMiddleware` to ensure assets are not slowed by attaching and storing sessions for every request
- Add feature class `\Heptacom\HeptaConnect\Package\WebFrontend\Components\SessionFeature` to control the session handling
- Add Symfony extension `web_frontend_session` configuration `enabled` to enable cookie-driven session management
- Add Symfony extension `web_frontend_session` configuration `session_lifetime` to defines for how long a session should be stored
- Add Symfony extension `web_frontend_session` configuration `cookie_name` to set the name of the cookie used for storing the session in a request and response
- Add Symfony extension `web_frontend_session` configuration `cache_key_prefix` to set the prefix of the cache storage used for the sessions
- Add base class `\Heptacom\HeptaConnect\Package\WebFrontend\Components\Page\Contract\AbstractPage` to identify page structure classes
- Add compiler pass `\Heptacom\HeptaConnect\Package\WebFrontend\DependencyInjection\RemovePagesCompilerPass` to remove any services, that might accidentally be picked up as service, but are a page structure object
- Add service `Heptacom\HeptaConnect\Package\WebFrontend\Components\Page\Contract\WebPageTwigEnvironmentFactoryInterface` implemented by `\Heptacom\HeptaConnect\Package\WebFrontend\Components\Page\WebPageTwigEnvironmentFactory` to generate Twig environments to render HTML pages
- Add service `Heptacom\HeptaConnect\Package\WebFrontend\Components\Page\Contract\WebPageRendererInterface` implemented by `\Heptacom\HeptaConnect\Package\WebFrontend\Components\Page\WebPageRenderer` to render any `\Heptacom\HeptaConnect\Package\WebFrontend\Components\Page\Contract\AbstractPage` in a request
- Add base class `\Heptacom\HeptaConnect\Package\WebFrontend\Components\Page\Contract\UiHandlerContract` for HTTP handlers, that work with `\Heptacom\HeptaConnect\Package\WebFrontend\Components\Page\Contract\AbstractPage`
- Add compiler pass `\Heptacom\HeptaConnect\Package\WebFrontend\DependencyInjection\ControllerPreparationCompilerPass` to automatically tag services of `\Heptacom\HeptaConnect\Package\WebFrontend\Components\Page\Contract\UiHandlerContract`
- Add fallback page `\Heptacom\HeptaConnect\Package\WebFrontend\Components\Page\DefaultPage\DefaultPage` with template `@WebFrontendPackage/ui/page/index/index.html.twig` handled by `\Heptacom\HeptaConnect\Package\WebFrontend\Components\Page\DefaultPage\DefaultUiHandler` to always have page to show
- Add feature class `\Heptacom\HeptaConnect\Package\WebFrontend\Components\PageFeature` to control page handling
- Add Symfony extension `web_frontend_page` configuration `enabled` to enable page rendering service
- Add Symfony extension `web_frontend_page` configuration `default_page_enabled` to enables the fallback page
- Add Symfony extension `web_frontend_page` configuration `default_page_path` to set the fallback page path

- Add HTTP handler `\Heptacom\HeptaConnect\Package\WebFrontend\Components\AccessProtection\LoginHandler` to render and and perform a login
- Add HTTP handler `\Heptacom\HeptaConnect\Package\WebFrontend\Components\AccessProtection\LogoutHandler` to perform a logout
- Add status reporter `\Heptacom\HeptaConnect\Package\WebFrontend\Components\AccessProtection\AccessLoginCommand` to create root access login links
- Add service `Heptacom\HeptaConnect\Package\WebFrontend\Components\AccessProtection\Contract\AccessProtectionServiceInterface` implemented by `\Heptacom\HeptaConnect\Package\WebFrontend\Components\AccessProtection\AccessProtectionService` to generate root login links
- Add service `Heptacom\HeptaConnect\Package\WebFrontend\Components\AccessProtection\Contract\AuthorizationBackendInterface` implemented by `\Heptacom\HeptaConnect\Package\WebFrontend\Components\AccessProtection\AuthorizationBackend` to manage `htpasswd`-alike file as user directory
- Add HTTP middleware service `Heptacom\HeptaConnect\Package\WebFrontend\Components\AccessProtection\AccessProtectionMiddleware` with a lower priority than `Heptacom\HeptaConnect\Package\WebFrontend\Components\Session\SessionMiddleware` to ensure sessions to access data are available to verify and assign login data
- Add lockscreen page `\Heptacom\HeptaConnect\Package\WebFrontend\Components\AccessProtection\LockscreenPage` with template `@WebFrontendPackage/ui/page/lockscreen/index.html.twig` with custom style in `@WebFrontendPackage/ui/page/lockscreen/css/lockscreen.css` handled by `\Heptacom\HeptaConnect\Package\WebFrontend\Components\AccessProtection\LockscreenUiHandler`
- Add feature class `\Heptacom\HeptaConnect\Package\WebFrontend\Components\AccessProtectionFeature` to control page access protection
- Add Symfony extension `web_frontend_access_protection` configuration `after_login_page_path` to set the path to the page, that will be redirected to after a login
- Add Symfony extension `web_frontend_access_protection` configuration `login_page_path` to set the path to the login form page
- Add Symfony extension `web_frontend_access_protection` configuration `login_path` to set the path to the login action
- Add Symfony extension `web_frontend_access_protection` configuration `logout_path` to set the path to the logout action

st changes

st changes

7.1.88 [0.9.7.0] - 2024-02-10

Fixed

- Fix deprecation notice `Creation of dynamic property` by explicitly declaring property `\Heptacom\HeptaConnect\Storage\Base\Bridge\Support\AbstractSingletonStorageFacade::$portalNodeStorageSetAction`

7.1.89 [0.9.5.0] - 2023-05-27

Fixed

- Fix default sort assignment in `\Heptacom\HeptaConnect\Storage\Base\Action\Identity\Overview\IdentityOverviewCriteria`, `\Heptacom\HeptaConnect\Storage\Base\Action\IdentityRedirect\Overview\IdentityRedirectOverviewCriteria`, `\Heptacom\HeptaConnect\Storage\Base\Action\PortalNode\Overview\PortalNodeOverviewCriteria`, `\Heptacom\HeptaConnect\Storage\Base\Action\PortalNodeAlias\Overview\PortalNodeAliasOverviewCriteria`, `\Heptacom\HeptaConnect\Storage\Base\Action\Route\Overview\RouteOverviewCriteria` and `\Heptacom\HeptaConnect\Storage\Base\Action\RouteCapability\Overview\RouteCapabilityOverviewCriteria` by making `\Heptacom\HeptaConnect\Storage\Base\Action\Identity\Overview\IdentityOverviewCriteria::$sort` protected again
- Fix syntax error in `\Heptacom\HeptaConnect\Storage\Base\Action\IdentityRedirect\Create\IdentityRedirectCreatePayload` affecting `php: ^7.4`

7.1.90 [0.9.4.0] - 2023-03-04

Added

- Add storage key interface `\Heptacom\HeptaConnect\Portal\Base\StorageKey\Contract\IdentityRedirectKeyInterface` with `\Heptacom\HeptaConnect\Storage\Base\IdentityRedirectKeyCollection`
- Add identity redirect create action `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\IdentityRedirect\IdentityRedirectCreateActionInterface` with `\Heptacom\HeptaConnect\Storage\Base\Action\IdentityRedirect\Create\IdentityRedirectCreatePayload`, `\Heptacom\HeptaConnect\Storage\Base\Action\IdentityRedirect\Create\IdentityRedirectCreatePayloadCollection`, `\Heptacom\HeptaConnect\Storage\Base\Action\IdentityRedirect\Create\IdentityRedirectCreateResult` and `\Heptacom\HeptaConnect\Storage\Base\Action\IdentityRedirect\Create\IdentityRedirectCreateResultCollection`
- Add methods `\Heptacom\HeptaConnect\Storage\Base\Bridge\Contract\StorageFacadeInterface::getIdentityRedirectCreateAction`, `\Heptacom\HeptaConnect\Storage\Base\Bridge\Support\AbstractSingletonStorageFacade::getIdentityRedirectCreateAction`, `\Heptacom\HeptaConnect\Storage\Base\Bridge\Support\AbstractSingletonStorageFacade::createIdentityRedirectCreateAction` and `\Heptacom\HeptaConnect\Storage\Base\Bridge\Support\Psr11StorageFacade::createIdentityRedirectCreateAction` to access the storage implementation for `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\IdentityRedirect\IdentityRedirectCreateActionInterface`
- Add identity redirect delete action `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\IdentityRedirect\IdentityRedirectDeleteActionInterface` with `\Heptacom\HeptaConnect\Storage\Base\Action\IdentityRedirect\Delete\IdentityRedirectDeleteCriteria`
- Add methods `\Heptacom\HeptaConnect\Storage\Base\Bridge\Contract\StorageFacadeInterface::getIdentityRedirectDeleteAction`, `\Heptacom\HeptaConnect\Storage\Base\Bridge\Support\AbstractSingletonStorageFacade::getIdentityRedirectDeleteAction`, `\Heptacom\HeptaConnect\Storage\Base\Bridge\Support\AbstractSingletonStorageFacade::createIdentityRedirectDeleteActionInterface` and `\Heptacom\HeptaConnect\Storage\Base\Bridge\Support\Psr11StorageFacade::createIdentityRedirectDeleteActionInterface` to access the storage implementation for `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\IdentityRedirect\IdentityRedirectDeleteActionInterface`
- Add identity redirect overview action `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\IdentityRedirect\IdentityRedirectOverviewActionInterface` with `\Heptacom\HeptaConnect\Storage\Base\Action\IdentityRedirect\Overview\IdentityRedirectOverviewCriteria` and `\Heptacom\HeptaConnect\Storage\Base\Action\IdentityRedirect\Overview\IdentityRedirectOverviewResult`
- Add methods `\Heptacom\HeptaConnect\Storage\Base\Bridge\Contract\StorageFacadeInterface::getIdentityRedirectOverviewAction`, `\Heptacom\HeptaConnect\Storage\Base\Bridge\Support\AbstractSingletonStorageFacade::getIdentityRedirectOverviewAction`,

`\Heptacom\HeptaConnect\Storage\Base\Bridge\Support\AbstractSingletonStorageFacade::createIdentityRedirectOverviewActionInterface` and `\Heptacom\HeptaConnect\Storage\Base\Bridge\Support\Psr11StorageFacade::createIdentityRedirectOverviewActionInterface` to access the storage implementation for `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\IdentityRedirect\IdentityRedirectOverviewActionInterface`

7.1.91 [0.9.0.0] - 2022-04-02

Added

- Add job state transition to schedule jobs after they failed with `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Job\JobScheduleActionInterface`, `\Heptacom\HeptaConnect\Storage\Base\Action\Job\Schedule\JobSchedulePayload` and `\Heptacom\HeptaConnect\Storage\Base\Action\Job\Schedule\JobScheduleResult`
- Add job state transition to fail jobs after they run with `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Job\JobFailActionInterface`, `\Heptacom\HeptaConnect\Storage\Base\Action\Job\Fail\JobFailPayload` and `\Heptacom\HeptaConnect\Storage\Base\Action\Job\Fail\JobFailResult`
- Add job state transition message to `\Heptacom\HeptaConnect\Storage\Base\Action\Job\Schedule\JobSchedulePayload`, `\Heptacom\HeptaConnect\Storage\Base\Action\Job\Fail\JobFailPayload`, `\Heptacom\HeptaConnect\Storage\Base\Action\Job\Finish\JobFinishPayload` and `\Heptacom\HeptaConnect\Storage\Base\Action\Job\Start\JobStartPayload`
- Add portal node extension activation action `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalExtension\PortalExtensionActivateActionInterface` with `\Heptacom\HeptaConnect\Storage\Base\Action\PortalExtension\Activate\PortalExtensionActivatePayload` and `\Heptacom\HeptaConnect\Storage\Base\Action\PortalExtension\Activate\PortalExtensionActivateResult`
- Add portal node extension deactivation action `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalExtension\PortalExtensionDeactivateActionInterface` with `\Heptacom\HeptaConnect\Storage\Base\Action\PortalExtension\Deactivate\PortalExtensionDeactivatePayload` and `\Heptacom\HeptaConnect\Storage\Base\Action\PortalExtension\Deactivate\PortalExtensionDeactivateResult`
- Add portal node extension activity find action `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalExtension\PortalExtensionFindActionInterface` with `\Heptacom\HeptaConnect\Storage\Base\Action\PortalExtension\Find\PortalExtensionFindResult`
- Add interface `\Heptacom\HeptaConnect\Storage\Base\Bridge\Contract\StorageFacadeInterface` for bridges and new composer package `heptacom/heptaconnect-test-suite-storage` to have central point to access storage
- Add supporting base class `\Heptacom\HeptaConnect\Storage\Base\Bridge\Support\Psr11StorageFacade` to implement `\Heptacom\HeptaConnect\Storage\Base\Bridge\Contract\StorageFacadeInterface` based upon a service container
- Add supporting base class `\Heptacom\HeptaConnect\Storage\Base\Bridge\Support\AbstractSingletonStorageFacade` to implement `\Heptacom\HeptaConnect\Storage\Base\Bridge\Contract\StorageFacadeInterface` that ensures in the implementation that services are only factorized once
- Add exception `\Heptacom\HeptaConnect\Storage\Base\Exception\ReadException` for storage actions to express issues on reading
- Add route delete action `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\RouteDeleteActionInterface` with `\Heptacom\HeptaConnect\Storage\Base\Action\Route\Delete\RouteDeleteCriteria`
- Add class `\Heptacom\HeptaConnect\Storage\Base\Action\FileReference\RequestGet\FileReferenceGetRequestCriteria` as input for `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\FileReference\FileReferenceGetRequestActionInterface::getRequest`
- Add class `\Heptacom\HeptaConnect\Storage\Base\Action\FileReference\RequestGet\FileReferenceGetRequestResult` as output for `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\FileReference\FileReferenceGetRequestActionInterface::getRequest`
- Add class `\Heptacom\HeptaConnect\Storage\Base\Action\FileReference\RequestPersist\FileReferencePersistRequestPayload` as input for `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\FileReference\FileReferencePersistRequestActionInterface::persistRequest`
- Add class `\Heptacom\HeptaConnect\Storage\Base\Action\FileReference\RequestPersist\FileReferencePersistRequestResult` as output for `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\FileReference\FileReferencePersistRequestActionInterface::persistRequest`

- Add class `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\FileReference\FileReferenceGetRequestActionInterface` to read serialized requests from storage
- Add class `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\FileReference\FileReferencePersistRequestActionInterface` to write serialized requests to storage
- Add class `\Heptacom\HeptaConnect\Storage\Base\Contract\FileReferenceRequestKeyInterface` as storage key for stored request objects
- Add class `\Heptacom\HeptaConnect\Storage\Base\FileReferenceRequestKeyCollection` as collection for `\Heptacom\HeptaConnect\Storage\Base\Contract\FileReferenceRequestKeyInterface`
- Add `\Heptacom\HeptaConnect\Storage\Base\AliasAwarePortalNodeStorageKey` as implementation to identify a portal node key that must be displayed as alias whenever possible
- Add storage action `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNodeAlias\PortalNodeAliasFindActionInterface`, `\Heptacom\HeptaConnect\Storage\Base\Action\PortalNodeAlias\Find\PortalNodeAliasFindCriteria` and `\Heptacom\HeptaConnect\Storage\Base\Action\PortalNodeAlias\Find\PortalNodeAliasFindResult` to find portal node keys by alias
- Add storage action `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNodeAlias\PortalNodeAliasGetActionInterface`, `\Heptacom\HeptaConnect\Storage\Base\Action\PortalNodeAlias\Get\PortalNodeAliasGetCriteria` and `\Heptacom\HeptaConnect\Storage\Base\Action\PortalNodeAlias\Get\PortalNodeAliasGetResult` to get aliases by portal node keys
- Add storage action `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNodeAlias\PortalNodeAliasOverviewActionInterface`, `\Heptacom\HeptaConnect\Storage\Base\Action\PortalNodeAlias\Overview\PortalNodeAliasOverviewCriteria` and `\Heptacom\HeptaConnect\Storage\Base\Action\PortalNodeAlias\Overview\PortalNodeAliasOverviewResult` to overview all defined portal node aliases
- Add storage action `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNodeAlias\PortalNodeAliasSetActionInterface`, `\Heptacom\HeptaConnect\Storage\Base\Action\PortalNodeAlias\Set\PortalNodeAliasSetPayload` and `\Heptacom\HeptaConnect\Storage\Base\Action\PortalNodeAlias\Set\PortalNodeAliasSetPayloads` to set and unset portal node aliases
- Add exception `\Heptacom\HeptaConnect\Storage\Base\Exception\UpdateException` to identify errors on updates in the storage

Changed

- Replace `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\JobRepositoryContract::add`, `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\JobPayloadRepositoryContract::add` and `\Heptacom\HeptaConnect\Storage\Base\Repository\JobAdd` with storage action `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Job\JobCreateActionInterface`, `\Heptacom\HeptaConnect\Storage\Base\Action\Job\Create\JobCreatePayloads`, `\Heptacom\HeptaConnect\Storage\Base\Action\Job\Create\JobCreatePayload`, `\Heptacom\HeptaConnect\Storage\Base\Action\Job\Create\JobCreateResults` and `\Heptacom\HeptaConnect\Storage\Base\Action\Job\Create\JobCreateResult` to allow batch writing and collected writing of jobs and their payload
- Replace `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\JobRepositoryContract::get` and `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\JobPayloadRepositoryContract::get` with storage action `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Job\JobGetActionInterface`, `\Heptacom\HeptaConnect\Storage\Base\Action\Job\Get\JobGetCriteria` and `\Heptacom\HeptaConnect\Storage\Base\Action\Job\Get\JobGetResult` to allow batch reading and collected reading of jobs and their payload
- Replace `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\JobRepositoryContract::start` with storage action `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Job\JobStartActionInterface`, `\Heptacom\HeptaConnect\Storage\Base\Action\Job\Start\JobStartPayload` and `\Heptacom\HeptaConnect\Storage\Base\Action\Job\Start\JobStartResult` to allow batch state change
- Replace `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\JobRepositoryContract::finish` with storage action `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Job\JobFinishActionInterface`, `\Heptacom\HeptaConnect\Storage\Base\Action\Job\Finish\JobFinishPayload` and `\Heptacom\HeptaConnect\Storage\Base\Action\Job\Finish\JobFinishResult` to allow batch state change

- Replace `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\JobRepositoryContract::remove` and `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\JobPayloadRepositoryContract::remove` with storage action `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Job\JobCreateActionInterface`, `\Heptacom\HeptaConnect\Storage\Base\Action\Job\Create\JobCreatePayloads`, `\Heptacom\HeptaConnect\Storage\Base\Action\Job\Create\JobCreatePayload`, `\Heptacom\HeptaConnect\Storage\Base\Action\Job\Create\JobCreateResults` and `\Heptacom\HeptaConnect\Storage\Base\Action\Job\Create\JobCreateResult` to allow batch deletion and collected deletion of jobs and their payloads
- Split up `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\JobRepositoryContract::cleanup` and `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\JobPayloadRepositoryContract::cleanup` into storage actions `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Job\JobListFinishedActionInterface` and `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Job\JobDeleteActionInterface` to separate searching and finding deletable jobs
- With storage restructure explained in this ADR we add `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNode\PortalNodeCreateActionInterface` to create portal nodes by the given `\Heptacom\HeptaConnect\Storage\Base\Action\PortalNode\Create\PortalNodeCreatePayloads` and `\Heptacom\HeptaConnect\Storage\Base\Action\PortalNode\Create\PortalNodeCreatePayload` to return a `\Heptacom\HeptaConnect\Storage\Base\Action\PortalNode\Create\PortalNodeCreateResults` and `\Heptacom\HeptaConnect\Storage\Base\Action\PortalNode\Create\PortalNodeCreateResult`
- With storage restructure explained in this ADR we add `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNode\PortalNodeDeleteActionInterface` to delete portal nodes by the given `\Heptacom\HeptaConnect\Storage\Base\Action\PortalNode\Delete\PortalNodeDeleteCriteria`
- With storage restructure explained in this ADR we add `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNode\PortalNodeGetActionInterface` to get portal nodes by the given `\Heptacom\HeptaConnect\Storage\Base\Action\PortalNode\Get\PortalNodeGetCriteria` to return a `\Heptacom\HeptaConnect\Storage\Base\Action\PortalNode\Get\PortalNodeGetResult`
- With storage restructure explained in this ADR we add `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNode\PortalNodeListActionInterface` to get all portal nodes and return a `\Heptacom\HeptaConnect\Storage\Base\Action\PortalNode\Listing\PortalNodeListResult`
- With storage restructure explained in this ADR we add `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNode\PortalNodeOverviewActionInterface` to an overview of portal nodes by the given `\Heptacom\HeptaConnect\Storage\Base\Action\PortalNode\Overview\PortalNodeOverviewCriteria` to return a `\Heptacom\HeptaConnect\Storage\Base\Action\PortalNode\Overview\PortalNodeOverviewResult`
- Move class `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Listing\ReceptionRouteListCriteria` to a new namespace `\Heptacom\HeptaConnect\Storage\Base\Action\Route\Listing\ReceptionRouteListCriteria`
- Move class `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Listing\ReceptionRouteListResult` to a new namespace `\Heptacom\HeptaConnect\Storage\Base\Action\Route\Listing\ReceptionRouteListResult`
- Move class `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Overview\RouteOverviewCriteria` to a new namespace `\Heptacom\HeptaConnect\Storage\Base\Action\Route\Overview\RouteOverviewCriteria`
- Move class `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Overview\RouteOverviewResult` to a new namespace `\Heptacom\HeptaConnect\Storage\Base\Action\Route\Overview\RouteOverviewResult`
- Move class `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Find\RouteFindCriteria` to a new namespace `\Heptacom\HeptaConnect\Storage\Base\Action\Route\Find\RouteFindCriteria`
- Move class `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Find\RouteFindResult` to a new namespace `\Heptacom\HeptaConnect\Storage\Base\Action\Route\Find\RouteFindResult`
- Move class `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Get\RouteGetCriteria` to a new namespace `\Heptacom\HeptaConnect\Storage\Base\Action\Route\Get\RouteGetCriteria`
- Move class `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Get\RouteGetResult` to a new namespace `\Heptacom\HeptaConnect\Storage\Base\Action\Route\Get\RouteGetResult`
- Move class `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Create\RouteCreatePayloads` to a new namespace `\Heptacom\HeptaConnect\Storage\Base\Action\Route\Create\RouteCreatePayloads`

- Move class `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Create\RouteCreatePayload` to a new namespace `\Heptacom\HeptaConnect\Storage\Base\Action\Route\Create\RouteCreatePayload`
- Move class `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Create\RouteCreateResults` to a new namespace `\Heptacom\HeptaConnect\Storage\Base\Action\Route\Create\RouteCreateResults`
- Move class `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Create\RouteCreateResult` to a new namespace `\Heptacom\HeptaConnect\Storage\Base\Action\Route\Create\RouteCreateResult`
- Move class `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\RouteCapability\Overview\RouteCapabilityOverviewCriteria` to a new namespace `\Heptacom\HeptaConnect\Storage\Base\Action\RouteCapability\Overview\RouteCapabilityOverviewCriteria`
- Move class `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\RouteCapability\Overview\RouteCapabilityOverviewResult` to a new namespace `\Heptacom\HeptaConnect\Storage\Base\Action\RouteCapability\Overview\RouteCapabilityOverviewResult`
- Move class `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\WebHttpHandlerConfiguration\Find\WebHttpHandlerConfigurationFindCriteria` to a new namespace `\Heptacom\HeptaConnect\Storage\Base\Action\WebHttpHandlerConfiguration\Find\WebHttpHandlerConfigurationFindCriteria`
- Move class `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\WebHttpHandlerConfiguration\Find\WebHttpHandlerConfigurationFindResult` to a new namespace `\Heptacom\HeptaConnect\Storage\Base\Action\WebHttpHandlerConfiguration\Find\WebHttpHandlerConfigurationFindResult`
- Move class `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\WebHttpHandlerConfiguration\Set\WebHttpHandlerConfigurationSetPayloads` to a new namespace `\Heptacom\HeptaConnect\Storage\Base\Action\WebHttpHandlerConfiguration\Set\WebHttpHandlerConfigurationSetPayloads`
- Move class `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\WebHttpHandlerConfiguration\Set\WebHttpHandlerConfigurationSetPayload` to a new namespace `\Heptacom\HeptaConnect\Storage\Base\Action\WebHttpHandlerConfiguration\Set\WebHttpHandlerConfigurationSetPayload`
- Move class `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Listing\ListReceptionRouteListActionInterface` to a new namespace `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\ReceptionRouteListActionInterface`
- Move class `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Overview\RouteOverviewActionInterface` to a new namespace `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\RouteOverviewActionInterface`
- Move class `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Find\RouteFindActionInterface` to a new namespace `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\RouteFindActionInterface`
- Move class `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Get\RouteGetActionInterface` to a new namespace `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\RouteGetActionInterface`
- Move class `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Croute\RouteCreateActionInterface` to a new namespace `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\RouteCreateActionInterface`
- Move class `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\RouteCapability\Overview\RouteCapabilityOverviewActionInterface` to a new namespace `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\RouteCapability\RouteCapabilityOverviewActionInterface`
- Move class `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\WebHttpHandlerConfiguration\Find\WebHttpHandlerConfigurationFindActionInterface` to a new namespace `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\WebHttpHandlerConfiguration\WebHttpHandlerConfigurationFindActionInterface`
- Move class `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\WebHttpHandlerConfiguration\Set\WebHttpHandlerConfigurationSetActionInterface` to a new namespace `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\WebHttpHandlerConfiguration\WebHttpHandlerConfigurationSetActionInterface`
- Replace `\Heptacom\HeptaConnect\Storage\Base\Contract\ConfigurationStorageContract::getConfiguration` with storage action `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNodeConfiguration\PortalNodeConfigurationGetActionInterface`, `\Heptacom\HeptaConnect\Storage\Base\Action\PortalNodeConfiguration\Get\PortalNodeConfigurationGetCriteria` and

`\Heptacom\HeptaConnect\Storage\Base\Action\PortalNodeConfiguration\Get\PortalNodeConfigurationGetResult` that allows for optimizations for different use-cases

- Replace `\Heptacom\HeptaConnect\Storage\Base\Contract\ConfigurationStorageContract::setConfiguration` with storage action `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNodeConfiguration\PortalNodeConfigurationSetActionInterface`, `\Heptacom\HeptaConnect\Storage\Base\Action\PortalNodeConfiguration\Set\PortalNodeConfigurationSetPayload` and `\Heptacom\HeptaConnect\Storage\Base\Action\PortalNodeConfiguration\Set\PortalNodeConfigurationSetPayloads` that allows for optimizations for different use-cases
- Replace `\Heptacom\HeptaConnect\Storage\Base\Contract\EntityMapperContract::mapEntities` with storage action `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Identity\IdentityMapActionInterface::map`, `\Heptacom\HeptaConnect\Storage\Base\Action\Identity\Mapping`, `\Heptacom\HeptaConnect\Storage\Base\Action\Identity\Map\IdentityMapPayload` and `\Heptacom\HeptaConnect\Storage\Base\Action\Identity\Map\IdentityMapResult` that allows for optimizations for different use-cases
- Replace `\Heptacom\HeptaConnect\Storage\Base\MappingPersister\Contract\MappingPersisterContract::persist`, `\Heptacom\HeptaConnect\Storage\Base\MappingPersistPayload` and `\Heptacom\HeptaConnect\Storage\Base\MappingPersister\Exception\MappingConflictException` with storage action `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Identity\IdentityPersistActionInterface::persist`, `\Heptacom\HeptaConnect\Storage\Base\Action\Identity\Persist\IdentityPersistPayload`, `\Heptacom\HeptaConnect\Storage\Base\Action\Identity\Persist\IdentityPersistPayloadCollection`, `\Heptacom\HeptaConnect\Storage\Base\Action\Identity\Persist\IdentityPersistPayloadContract`, `\Heptacom\HeptaConnect\Storage\Base\Action\Identity\Persist\IdentityPersistCreatePayload`, `\Heptacom\HeptaConnect\Storage\Base\Action\Identity\Persist\IdentityPersistDeletePayload`, `\Heptacom\HeptaConnect\Storage\Base\Action\Identity\Persist\IdentityPersistUpdatePayload` and `\Heptacom\HeptaConnect\Storage\Base\Action\Identity\Exception\IdentityConflictException` that allows for optimizations for different use-cases
- Replace `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\MappingRepositoryContract::listByMappingNode`, `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\MappingRepositoryContract::listByPortalNodeAndType`, `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\MappingRepositoryContract::read`, `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\MappingNodeRepositoryContract::listByTypeAndPortalNodeAndExternalIds` and `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\MappingNodeRepositoryContract::read` with storage action `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Identity\IdentityOverviewActionInterface`, `\Heptacom\HeptaConnect\Storage\Base\Action\Identity\Overview\IdentityOverviewCriteria` and `\Heptacom\HeptaConnect\Storage\Base\Action\Identity\Overview\IdentityOverviewResult` that allows for optimizations for different use-cases
- Replace `\Heptacom\HeptaConnect\Storage\Base\Contract\EntityReflectorContract::reflectEntities` with storage action `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Identity\IdentityReflectActionInterface::reflect` and `\Heptacom\HeptaConnect\Storage\Base\Action\Identity\Reflect\IdentityReflectPayload` that allows for optimizations for different use-cases
- Add final modifier to `\Heptacom\HeptaConnect\Storage\Base\Action\Route\Create\RouteCreatePayload` and `\Heptacom\HeptaConnect\Storage\Base\Action\Route\Create\RouteCreatePayloads` to ensure correct usage of implementation. To still add custom data the `\Heptacom\HeptaConnect\Dataset\Base\AttachmentAwareInterface` is implemented by `\Heptacom\HeptaConnect\Storage\Base\Action\Route\Create\RouteCreatePayloads`
- Add final modifier to `\Heptacom\HeptaConnect\Storage\Base\Action\Route\Create\RouteCreateResult` and `\Heptacom\HeptaConnect\Storage\Base\Action\Route\Create\RouteCreateResults` to ensure correct usage of implementation. To still add custom data the `\Heptacom\HeptaConnect\Dataset\Base\AttachmentAwareInterface` is implemented by `\Heptacom\HeptaConnect\Storage\Base\Action\Route\Create\RouteCreateResults`
- Add final modifier to `\Heptacom\HeptaConnect\Storage\Base\Action\Route\Find\RouteFindCriteria` to ensure correct usage of implementation. To still add custom data the `\Heptacom\HeptaConnect\Dataset\Base\AttachmentAwareInterface` is implemented by `\Heptacom\HeptaConnect\Storage\Base\Action\Route\Find\RouteFindCriteria`
- Add final modifier to `\Heptacom\HeptaConnect\Storage\Base\Action\Route\Find\RouteFindResult` to ensure correct usage of implementation. To still add custom data the `\Heptacom\HeptaConnect\Dataset\Base\AttachmentAwareInterface` is implemented by `\Heptacom\HeptaConnect\Storage\Base\Action\Route\Find\RouteFindResult`

- Add final modifier to `\Heptacom\HeptaConnect\Storage\Base\Action\Route\Get\RouteGetCriteria` to ensure correct usage of implementation. To still add custom data the `\Heptacom\HeptaConnect\Dataset\Base\AttachmentAwareInterface` is implemented by `\Heptacom\HeptaConnect\Storage\Base\Action\Route\Get\RouteGetCriteria`
- Add final modifier to `\Heptacom\HeptaConnect\Storage\Base\Action\Route\Get\RouteGetResult` to ensure correct usage of implementation. To still add custom data the `\Heptacom\HeptaConnect\Dataset\Base\AttachmentAwareInterface` is implemented by `\Heptacom\HeptaConnect\Storage\Base\Action\Route\Get\RouteGetResult`
- Add final modifier to `\Heptacom\HeptaConnect\Storage\Base\Action\Route\Listing\ReceptionRouteListCriteria` to ensure correct usage of implementation. To still add custom data the `\Heptacom\HeptaConnect\Dataset\Base\AttachmentAwareInterface` is implemented by `\Heptacom\HeptaConnect\Storage\Base\Action\Route\Listing\ReceptionRouteListCriteria`
- Add final modifier to `\Heptacom\HeptaConnect\Storage\Base\Action\Route\Listing\ReceptionRouteListResult` to ensure correct usage of implementation. To still add custom data the `\Heptacom\HeptaConnect\Dataset\Base\AttachmentAwareInterface` is implemented by `\Heptacom\HeptaConnect\Storage\Base\Action\Route\Listing\ReceptionRouteListResult`
- Add final modifier to `\Heptacom\HeptaConnect\Storage\Base\Action\Route\Overview\RouteOverviewCriteria` to ensure correct usage of implementation. To still add custom data the `\Heptacom\HeptaConnect\Dataset\Base\AttachmentAwareInterface` is implemented by `\Heptacom\HeptaConnect\Storage\Base\Action\Route\Overview\RouteOverviewCriteria`
- Add final modifier to `\Heptacom\HeptaConnect\Storage\Base\Action\Route\Overview\RouteOverviewResult` to ensure correct usage of implementation. To still add custom data the `\Heptacom\HeptaConnect\Dataset\Base\AttachmentAwareInterface` is implemented by `\Heptacom\HeptaConnect\Storage\Base\Action\Route\Overview\RouteOverviewResult`
- Add final modifier to `\Heptacom\HeptaConnect\Storage\Base\Action\RouteCapability\Overview\RouteCapabilityOverviewCriteria` to ensure correct usage of implementation. To still add custom data the `\Heptacom\HeptaConnect\Dataset\Base\AttachmentAwareInterface` is implemented by `\Heptacom\HeptaConnect\Storage\Base\Action\RouteCapability\Overview\RouteCapabilityOverviewCriteria`
- Add final modifier to `\Heptacom\HeptaConnect\Storage\Base\Action\RouteCapability\Overview\RouteCapabilityOverviewResult` to ensure correct usage of implementation. To still add custom data the `\Heptacom\HeptaConnect\Dataset\Base\AttachmentAwareInterface` is implemented by `\Heptacom\HeptaConnect\Storage\Base\Action\RouteCapability\Overview\RouteCapabilityOverviewResult`
- Add final modifier to `\Heptacom\HeptaConnect\Storage\Base\Action\WebHttpHandlerConfiguration\Find\WebHttpHandlerConfigurationFindCriteria` to ensure correct usage of implementation. To still add custom data the `\Heptacom\HeptaConnect\Dataset\Base\AttachmentAwareInterface` is implemented by `\Heptacom\HeptaConnect\Storage\Base\Action\WebHttpHandlerConfiguration\Find\WebHttpHandlerConfigurationFindCriteria`
- Add final modifier to `\Heptacom\HeptaConnect\Storage\Base\Action\WebHttpHandlerConfiguration\Find\WebHttpHandlerConfigurationFindResult` to ensure correct usage of implementation. To still add custom data the `\Heptacom\HeptaConnect\Dataset\Base\AttachmentAwareInterface` is implemented by `\Heptacom\HeptaConnect\Storage\Base\Action\WebHttpHandlerConfiguration\Find\WebHttpHandlerConfigurationFindResult`
- Add final modifier to `\Heptacom\HeptaConnect\Storage\Base\Action\WebHttpHandlerConfiguration\Set\WebHttpHandlerConfigurationSetPayload` and `\Heptacom\HeptaConnect\Storage\Base\Action\WebHttpHandlerConfiguration\Set\WebHttpHandlerConfigurationSetPayloads` to ensure correct usage of implementation. To still add custom data the `\Heptacom\HeptaConnect\Dataset\Base\AttachmentAwareInterface` is implemented by `\Heptacom\HeptaConnect\Storage\Base\Action\WebHttpHandlerConfiguration\Set\WebHttpHandlerConfigurationSetPayloads`
- Add final modifier to `\Heptacom\HeptaConnect\Storage\Base\PreviewPortalNodeKey` and `\Heptacom\HeptaConnect\Storage\Base\PrimaryKeySharingMappingStruct` to ensure correct usage of implementation. Decoration by their interfaces or base classes is still possible
- Replace `\Heptacom\HeptaConnect\Storage\Base\Contract\PortalStorageContract::clear` with storage action `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNodeStorage\PortalNodeStorageClearActionInterface::clear` and `\Heptacom\HeptaConnect\Storage\Base\Action\PortalNodeStorage\Clear\PortalNodeStorageClearCriteria` that allows for optimizations for different use-cases
- Replace `\Heptacom\HeptaConnect\Storage\Base\Contract\PortalStorageContract::deleteMultiple` and `\Heptacom\HeptaConnect\Storage\Base\Contract\PortalStorageContract::unset` with storage action `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNodeStorage\PortalNodeStorageDeleteActionInterface::delete` and

`\Heptacom\HeptaConnect\Storage\Base\Action\PortalNodeStorage\Delete\PortalNodeStorageDeleteCriteria` that allows for optimizations for different use-cases

- Replace `\Heptacom\HeptaConnect\Storage\Base\Contract\PortalStorageContract::getMultiple`, `\Heptacom\HeptaConnect\Storage\Base\Contract\PortalStorageContract::getValue`, `\Heptacom\HeptaConnect\Storage\Base\Contract\PortalStorageContract::getType` and `\Heptacom\HeptaConnect\Storage\Base\Contract\PortalStorageContract::has` with storage action `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNodeStorage\PortalNodeStorageGetActionInterface::get` and `\Heptacom\HeptaConnect\Storage\Base\Action\PortalNodeStorage\Get\PortalNodeStorageGetCriteria`, `\Heptacom\HeptaConnect\Storage\Base\Action\PortalNodeStorage\Get\PortalNodeStorageGetResult`, `\Heptacom\HeptaConnect\Storage\Base\Action\PortalNodeStorage\PortalNodeStorageItemContract` that allows for optimizations for different use-cases
- Replace `\Heptacom\HeptaConnect\Storage\Base\Contract\PortalStorageContract::setMultiple` and `\Heptacom\HeptaConnect\Storage\Base\Contract\PortalStorageContract::set` with storage action `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNodeStorage\PortalNodeStorageSetActionInterface::set` and `\Heptacom\HeptaConnect\Storage\Base\Action\PortalNodeStorage\Set\PortalNodeStorageSetItem`, `\Heptacom\HeptaConnect\Storage\Base\Action\PortalNodeStorage\Set\PortalNodeStorageSetItems` and `\Heptacom\HeptaConnect\Storage\Base\Action\PortalNodeStorage\Set\PortalNodeStorageSetPayload` that allows for optimizations for different use-cases
- Replace `\Heptacom\HeptaConnect\Storage\Base\Contract\PortalStorageContract::list` with storage action `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNodeStorage\PortalNodeStorageListActionInterface::list` and `\Heptacom\HeptaConnect\Storage\Base\Action\PortalNodeStorage\Listing\PortalNodeStorageListCriteria` and `\Heptacom\HeptaConnect\Storage\Base\Action\PortalNodeStorage\Listing\PortalNodeStorageListResult` that allows for optimizations for different use-cases
- Rename `\Heptacom\HeptaConnect\Portal\Base\StorageKey\Contract\MappingExceptionKeyInterface` to `\Heptacom\HeptaConnect\Portal\Base\StorageKey\Contract\IdentityErrorKeyInterface`
- Replace `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\MappingExceptionRepositoryContract::create` with storage action `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\IdentityError\IdentityErrorCreateActionInterface`, `\Heptacom\HeptaConnect\Storage\Base\Action\IdentityError\Create\IdentityErrorCreateResults`, `\Heptacom\HeptaConnect\Storage\Base\Action\IdentityError\Create\IdentityErrorCreateResult`, `\Heptacom\HeptaConnect\Storage\Base\Action\IdentityError\Create\IdentityErrorCreatePayloads` and `\Heptacom\HeptaConnect\Storage\Base\Action\IdentityError\Create\IdentityErrorCreatePayload` to allow batch writing of identity errors
- Move interface `\Heptacom\HeptaConnect\Portal\Base\StorageKey\Contract\RouteKeyInterface` to `\Heptacom\HeptaConnect\Storage\Base\Contract\RouteKeyInterface`
- Move class `\Heptacom\HeptaConnect\Portal\Base\StorageKey\RouteKeyCollection` to `\Heptacom\HeptaConnect\Storage\Base\RouteKeyCollection`

Removed

- Remove class `\Heptacom\HeptaConnect\Storage\Base\Contract\JobInterface`
- Remove class `\Heptacom\HeptaConnect\Storage\Base\Contract\JobPayloadKeyInterface`
- Remove class `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\JobPayloadRepositoryContract`
- Remove class `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\JobRepositoryContract`
- Remove class `\Heptacom\HeptaConnect\Storage\Base\Repository\JobAdd`
- With storage restructure explained in this ADR we remove implementation `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\PortalNodeRepositoryContract::read` in favour of `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNode\PortalNodeGetActionInterface::get` that allows for optimizations for different use-cases
- With storage restructure explained in this ADR we remove implementation `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\PortalNodeRepositoryContract::listAll` in favour of `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNode\PortalNodeListActionInterface::list` that allows for optimizations for different use-cases

- With storage restructure explained in this ADR we remove implementation `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\PortalNodeRepositoryContract::listByClass` in favour of `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNode\PortalNodeOverviewActionInterface::overview` that allows for optimizations for different use-cases
- With storage restructure explained in this ADR we remove implementation `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\PortalNodeRepositoryContract::create` in favour of `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNode\PortalNodeCreateActionInterface::create` that allows for optimizations for different use-cases
- With storage restructure explained in this ADR we remove implementation `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\PortalNodeRepositoryContract::create` in favour of `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNode\PortalNodeDeleteActionInterface::delete` that allows for optimizations for different use-cases
- Remove contracts and exceptions `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\CronjobRepositoryContract` and `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\CronjobRunRepositoryContract` as the feature of cronjobs in its current implementation is removed
- Remove unused contract method `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\MappingRepositoryContract::listByNodes`
- Remove unused contract method `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\MappingRepositoryContract::listUnsavedExternalIds`
- Remove unused contract method `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\MappingRepositoryContract::updateExternalId`
- Remove unused contract method `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\MappingNodeRepositoryContract::listByTypeAndPortalNodeAndExternalId`
- Remove unused contract method `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\MappingNodeRepositoryContract::create`
- Move contract `\Heptacom\HeptaConnect\Storage\Base\Contract\ResourceLockStorageContract` to `\Heptacom\HeptaConnect\Core\Parallelization\Contract\ResourceLockStorageContract` as it will be provided by integration and not storage
- Remove unused contract `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\MappingRepositoryContract`
- Remove unused `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\MappingExceptionRepositoryContract::listByMapping`, `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\MappingExceptionRepositoryContract::listByMappingAndType` and `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\MappingExceptionRepositoryContract::delete`
- Remove unused `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\MappingNodeRepositoryContract`
- Remove deprecated `\Heptacom\HeptaConnect\Storage\Base\Contract\StorageKeyGeneratorContract::generateKey`

st changes

7.1.92 [0.8.5] - 2021-12-28

Fixed

- Change composer dependency `bentools/iterable-functions: >=1 <2` to `bentools/iterable-functions: >=1.4 <2` to ensure availability of `iterable_map`

7.1.93 [0.8.4] - 2021-12-16

Removed

- Remove the code for unit tests, configuration for style checks as well as the Makefile

7.1.94 [0.8.0] - 2021-11-22

Added

- Add `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\JobRepositoryContract::start` for tracking the start of job processing
- Add `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\JobRepositoryContract::finish` for tracking the stop of job processing
- Add `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\JobRepositoryContract::cleanup` and `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\JobPayloadRepositoryContract::cleanup` for cleaning up executed jobs and their payloads
- Add `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Listing\ReceptionRouteListActionInterface` for listing reception routes by the given `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Listing\ReceptionRouteListCriteria` to return a `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Listing\ReceptionRouteListResult`
- Add base class `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Overview\OverviewCriteriaContract` for overview criterias
- With storage restructure explained in this ADR we add `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Overview\RouteOverviewActionInterface` for listing all routes by the given `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Overview\RouteOverviewCriteria` to return a `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Overview\RouteOverviewResult`
- With storage restructure explained in this ADR we add `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Find\RouteFindActionInterface` for checking the existence of a route by its components by the given `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Find\RouteFindCriteria` to return a `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Find\RouteFindResult`
- With storage restructure explained in this ADR we add `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Get\RouteGetActionInterface` for reading metadata of routes by the given `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Get\RouteGetCriteria` to return a `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Get\RouteGetResult`
- With storage restructure explained in this ADR we add `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Create\RouteCreateActionInterface` for creating routes by the given `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Create\RouteCreatePayloads` and `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Create\RouteCreatePayload` to return a `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Create\RouteCreateResults` of `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Create\RouteCreateResult`
- With storage restructure explained in this ADR we add `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\RouteCapability\Overview\RouteCapabilityOverviewActionInterface` for listing available route capabilities by the given `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\RouteCapability\Overview\RouteCapabilityOverviewCriteria` to return a `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\RouteCapability\Overview\RouteCapabilityOverviewResult`
- Add `\Heptacom\HeptaConnect\Storage\Base\Enum\RouteCapability` to hold constant values for route capabilities
- Add interface `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Create\CreatePayloadInterface` to reference to create payloads more easily in exceptions
- Add exception `\Heptacom\HeptaConnect\Storage\Base\Exception\CreateException` for all cases when creation failed

- Add exception `\Heptacom\HeptaConnect\Storage\Base\Exception\InvalidCreatePayloadException` for all cases when creation failed due to invalid payload
- Add exception `\Heptacom\HeptaConnect\Storage\Base\Exception\InvalidOverviewCriteriaException` for cases when overview criteria are malformed
- Add `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\WebHttpHandlerConfiguration\Find\WebHttpHandlerConfigurationFindActionInterface` to get a configuration key for an HTTP handler by `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\WebHttpHandlerConfiguration\Find\WebHttpHandlerConfigurationFindCriteria` into `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\WebHttpHandlerConfiguration\Find\WebHttpHandlerConfigurationFindResult`
- Add `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\WebHttpHandlerConfiguration\Set\WebHttpHandlerConfigurationSetActionInterface` to set configuration keys for HTTP handlers by `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\WebHttpHandlerConfiguration\Set\WebHttpHandlerConfigurationSetPayloads` and its `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\WebHttpHandlerConfiguration\Set\WebHttpHandlerConfigurationSetPayload`

Changed

- Change parameter name in `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\MappingNodeRepositoryContract::listByTypeAndPortalNodeAndExternalId` from `$datasetEntityClassName` to `$entityType`
- Change parameter name in `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\MappingNodeRepositoryContract::listByTypeAndPortalNodeAndExternalIds` from `$datasetEntityClassName` to `$entityType`
- Change parameter name in `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\MappingNodeRepositoryContract::create` from `$datasetEntityClassName` to `$entityType`
- Change parameter name in `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\MappingNodeRepositoryContract::createList` from `$datasetEntityClassName` to `$entityType`
- Change parameter name in `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\MappingRepositoryContract::listUnsavedExternalIds` from `$datasetEntityClassName` to `$entityType`
- Change parameter name in `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\MappingRepositoryContract::listByPortalNodeAndType` from `$datasetEntityType` to `$entityType`
- Change parameter name of `\Heptacom\HeptaConnect\Storage\Base\Exception\UnsharableOwnerException::__construct` from `$expectedDatasetEntityClassName` to `$expectedEntityType`
- Change parameter name of `\Heptacom\HeptaConnect\Storage\Base\PrimaryKeySharingMappingStruct::__construct` from `$datasetEntityClassName` to `$entityType`
- Change method name from `\Heptacom\HeptaConnect\Storage\Base\Contract\MappingNodeStructInterface::getDatasetEntityClassName` to `\Heptacom\HeptaConnect\Storage\Base\Contract\MappingNodeStructInterface::getEntityType`
- Change method name from `\Heptacom\HeptaConnect\Storage\Base\Exception\UnsharableOwnerException::getExpectedDatasetEntityClassName` to `\Heptacom\HeptaConnect\Storage\Base\Exception\UnsharableOwnerException::getExpectedEntityType`
- Change method name from `\Heptacom\HeptaConnect\Storage\Base\PrimaryKeySharingMappingStructgetDatasetEntityClassName` to `\Heptacom\HeptaConnect\Storage\Base\PrimaryKeySharingMappingStruct::getEntityType`
- Change method name from `\Heptacom\HeptaConnect\Storage\Base\PrimaryKeySharingMappingStruct::getForeignDatasetEntityClassName` to `\Heptacom\HeptaConnect\Storage\Base\PrimaryKeySharingMappingStruct::getForeignEntityType`

Removed

- With storage restructure explained in this ADR we remove implementation `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\RouteRepositoryContract::listBySourceAndEntityType` in favour of `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Listing\ReceptionRouteListActionInterface::list`, `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Overview\RouteOverviewActionInterface::overview` and `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Find\RouteFindActionInterface::find` that allows for optimizations for different use-cases

- With storage restructure explained in this ADR we remove implementation `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\RouteRepositoryContract::read` in favour of `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Get\RouteGetActionInterface::get` that allows for optimizations in the storage implementation
- With storage restructure explained in this ADR we remove implementation `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\RouteRepositoryContract::create` in favour of `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Create\RouteCreateActionInterface::create` that allows for optimizations in the storage implementation
- Remove `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\WebhookRepositoryContract`

7.1.95 [0.7.0] - 2021-09-25

Added

- Add methods in `\Heptacom\HeptaConnect\Storage\Base\Contract\PortalStorageContract` (`\Heptacom\HeptaConnect\Storage\Base\Contract\PortalStorageContract::clear`, `\Heptacom\HeptaConnect\Storage\Base\Contract\PortalStorageContract::getMultiple` and `\Heptacom\HeptaConnect\Storage\Base\Contract\PortalStorageContract::deleteMultiple`) to allow PSR simple cache compatibility
- Add contract `\Heptacom\HeptaConnect\Storage\Base\MappingPersister\Contract\MappingPersisterContract`. It must be used with `\Heptacom\HeptaConnect\Storage\Base\MappingPersistPayload`. It can throw `\Heptacom\HeptaConnect\Storage\Base\MappingPersister\Exception\MappingConflictException`.

Changed

- Change parameter in `\Heptacom\HeptaConnect\Storage\Base\TypedMappingCollection::__construct` to allow iterables to be consumed like its parent class

Fixed

- Require previously soft-required `bentools/iterable-functions: >=1 <2`

7.1.96 [0.9.1.1] - 2023-03-07

Added

- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Identity\IdentityReflect::LOOKUP_IDENTITY_REDIRECTS_QUERY` as `315e9e8f-b1b7-4e39-a42b-4dbdf3d8b14c` to identify a query used for looking up identity redirects, that evaluate identities before mapping nodes are evaluated

Fixed

- Add lookup for identity redirects to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Identity\IdentityReflect`

7.1.97 [0.9.1.0] - 2023-03-04

Added

- Add class `\Heptacom\HeptaConnect\Storage\ShopwareDal\StorageKey\IdentityRedirectStorageKey` implementing `\Heptacom\HeptaConnect\Storage\Base\Contract\IdentityRedirectKeyInterface` as storage key for identity redirects
- Add support for `\Heptacom\HeptaConnect\Storage\ShopwareDal\StorageKey\IdentityRedirectStorageKey` into `\Heptacom\HeptaConnect\Storage\ShopwareDal\StorageKeyGenerator`
- Add migration `\Heptacom\HeptaConnect\Storage\ShopwareDal\Migration\Migration1673717600AddIdentityRedirectTable` to add storage for identity redirects
- Add class `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\IdentityRedirect\IdentityRedirectCreate` implementing `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\IdentityRedirect\IdentityRedirectCreateInterface`
- Implement `\Heptacom\HeptaConnect\Storage\Base\Bridge\Contract\StorageFacadeInterface::getIdentityRedirectCreateAction` in `\Heptacom\HeptaConnect\Storage\ShopwareDal\Bridge\StorageFacade::createIdentityRedirectCreateActionInterface` to return `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\IdentityRedirect\IdentityRedirectCreate`
- Add exception code `1673722278` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\IdentityRedirect\IdentityRedirectCreate::create` when the payload refers to a source portal node with an invalid portal node
- Add exception code `1673722279` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\IdentityRedirect\IdentityRedirectCreate::create` when the payload refers to a target portal node with an invalid portal node
- Add exception code `1673722280` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\IdentityRedirect\IdentityRedirectCreate::create` when the payload refers to an unknown entity type
- Add exception code `1673722281` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\IdentityRedirect\IdentityRedirectCreate::create` when the key generator cannot generate a valid identity redirect key
- Add exception code `1673722282` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\IdentityRedirect\IdentityRedirectCreate::create` when writing to the database fails
- Add class `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\IdentityRedirect\IdentityRedirectDelete` implementing `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\IdentityRedirect\IdentityRedirectDeleteInterface`
- Implement `\Heptacom\HeptaConnect\Storage\Base\Bridge\Contract\StorageFacadeInterface::getIdentityRedirectDeleteAction` in `\Heptacom\HeptaConnect\Storage\ShopwareDal\Bridge\StorageFacade::createIdentityRedirectDeleteActionInterface` to return `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\IdentityRedirect\IdentityRedirectDelete`
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\IdentityRedirect\IdentityRedirectDelete::LOOKUP_QUERY` as `26f18fa9-9246-45cf-b7f7-2fc80f61151d` to identify a query used for deleting identity redirects
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\IdentityRedirect\IdentityRedirectDelete::DELETE_QUERY` as `ca54ecac-3b6b-4f54-882e-fea1f19336ba` to identify a query used for looking up identity redirects that can be deleted

- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\IdentityRedirect\IdentityRedirectOverview::OVERVIEW_QUERY` as `832dbfc9-4939-4301-ade4-aa73d961454f` to identify a query used for loading an overview page for identity redirects
- Implement `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\IdentityRedirect\IdentityRedirectOverviewActionInterface` in `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\IdentityRedirect\IdentityRedirectOverview` to list identity redirects
- Add exception code `1673729808` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\IdentityRedirect\IdentityRedirectOverview::overview` when the payload refers to a identity redirect with an invalid identity redirect key
- Add exception code `1673729809` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\IdentityRedirect\IdentityRedirectOverview::overview` when the payload refers to a source portal node with an invalid portal node key
- Add exception code `1673729810` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\IdentityRedirect\IdentityRedirectOverview::overview` when the payload refers to a target portal node with an invalid portal node key
- Add exception code `1673729811` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\IdentityRedirect\IdentityRedirectOverview::overview` when the criteria has an invalid sorting option

Changed

- Add migration `\Heptacom\HeptaConnect\Storage\ShopwareDal\Migration\Migration1677428200AddKeyIndexToPortalNodeStorageTable` to add index to `key` to table `heptaconnect_portal_node_storage` for improved portal node storage reads
- Add migration `\Heptacom\HeptaConnect\Storage\ShopwareDal\Migration\Migration1677950300AddExternalIdIndexToJobTable` to add index to `external_id` to table `heptaconnect_job` for better database usage outside of business logic
- Raise composer dependency constraint for `heptacom/heptaconnect-dataset-base`, `heptacom/heptaconnect-portal-base` and `heptacom/heptaconnect-storage-base` from `^0.9.3` to `^0.9.4`

Fixed

- Ensure query `900bdcb4-3a2a-4092-9eed-f5902e97b02f` in `\Heptacom\HeptaConnect\Storage\ShopwareDal\WebHttpHandlerAccessor` uses an ordering to ensure iteration on big data sets is ordered correctly and passes runtime tests
- Ensure query `f683453e-336f-4913-8bb9-aa0e34745f97` in `\Heptacom\HeptaConnect\Storage\ShopwareDal\WebHttpHandlerPathAccessor` uses an ordering to ensure iteration on big data sets is ordered correctly and passes runtime tests
- Ensure query `f6c5db7b-004d-40c8-b9cc-53707aab658b` in `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\WebHttpHandlerConfiguration\WebHttpHandlerConfigurationFind` uses an ordering to ensure iteration on big data sets is ordered correctly and passes runtime tests
- Fix incorrect SQL statement when deleting entries in `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\WebHttpHandlerConfiguration\WebHttpHandlerConfigurationSet`

7.1.98 [0.9.0.6] - 2023-02-14

Changed

- Add migration `\Heptacom\HeptaConnect\Storage\ShopwareDal\Migration\Migration1674420000AddJobTransactionIdIndex` to add index to `transaction_id` to table `heptaconnect_job` for improved job state changes

Fixed

- Prevent duplication of entries in the portal-storage when updating keys that are already expired.

7.1.99 [0.9.0.5] - 2022-11-19

Fixed

- Fix error when creating mappings via `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Identity\IdentityReflect`. Insertion payload was not binary as expected.

7.1.100 [0.9.0.4] - 2022-10-03

Fixed

- Pagination in `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Identity\IdentityOverview`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNode\PortalNodeOverview`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNodeAlias\PortalNodeAliasOverview`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Route\RouteOverview`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\RouteCapability\RouteCapabilityOverview` was one page in advance and therefore made page 1 only accessable when listing without pagination in criteria

7.1.101 [0.9.0.3] - 2022-09-20

Fixed

- Fix error when creating mappings via `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Identity\IdentityReflect`. The insert-query now uses the correct table-name.

7.1.102 [0.9.0.2] - 2022-07-12

Fixed

- Fix error when deleting many jobs at once by chunking job deletion to 1000 jobs at a time in `Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Job\JobDelete::delete`
- Fix issue in validation before mapping-node merging involving deleted mappings in `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Identity\IdentityPersist::validateMappingNodesCanBeMerged`

7.1.103 [0.9.0.1] - 2022-04-19

Fixed

- Fix error related to foreign key checks in migration `\Heptacom\HeptaConnect\Storage\ShopwareDal\Migration\Migration1639860447UpdateExistingJobData`

7.1.104 [0.9.0.0] - 2022-04-02

Added

- Add class `\Heptacom\HeptaConnect\Storage\ShopwareDal\JobTypeAccessor`
- Add state in `\Heptacom\HeptaConnect\Storage\ShopwareDal\Support\Query\QueryBuilder` to make selects for update to trigger row locks
- Add constants for job states in `\Heptacom\HeptaConnect\Storage\ShopwareDal\Support\Enum\JobStateEnum`
- Add migration `\Heptacom\HeptaConnect\Storage\ShopwareDal\Migration\Migration1639246133CreateStateHistoryForJobs` to add job state history
- Add migration `\Heptacom\HeptaConnect\Storage\ShopwareDal\Migration\Migration1639270114InsertJobStates` to add job states
- Add migration `\Heptacom\HeptaConnect\Storage\ShopwareDal\Migration\Migration1639860447UpdateExistingJobData` to migrate state date into job history
- Implement `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Job\JobCreateActionInterface` in `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Job\JobCreate`

- Implement `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Job\JobDeleteActionInterface` in `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Job\JobDelete`
- Implement `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Job\JobFailActionInterface` in `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Job\JobFail`
- Implement `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Job\JobFinishActionInterface` in `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Job\JobFinish`
- Implement `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Job\JobGetActionInterface` in `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Job\JobGet`
- Implement `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Job\JobListFinishedActionInterface` in `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Job\JobFinishedList`
- Implement `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Job\JobScheduleActionInterface` in `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Job\JobSchedule`
- Implement `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Job\JobStartActionInterface` in `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Job\JobStart`
- Add implementation for `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNode\PortalNodeOverviewActionInterface` in `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNode\PortalNodeOverview`
- Add implementation for `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNode\PortalNodeListActionInterface` in `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNode\PortalNodeList`
- Add implementation for `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNode\PortalNodeGetActionInterface` in `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNode\PortalNodeGet`
- Add implementation for `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNode\PortalNodeDeleteActionInterface` in `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNode\PortalNodeDelete`
- Add implementation for `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNode\PortalNodeCreateActionInterface` in `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNode\PortalNodeCreate`
- Add exception code `1640048751` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNode\PortalNodeCreate::create` when the key generator cannot generate a valid portal node key
- Add exception code `1648345724` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNode\PortalNodeCreate::create` when the portal node alias is empty
- Add exception code `1648345725` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNode\PortalNodeCreate::create` when the portal node alias is already used
- Add exception code `1640405544` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNode\PortalNodeOverview::overview` when the criteria has an invalid sorting option
- Add migration `\Heptacom\HeptaConnect\Storage\ShopwareDal\Migration\Migration1640360050CreatePortalExtensionConfigurationTable` to add table for portal extension activity state
- Add base class `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalExtension\PortalExtensionSwitchActive` to simplify implementations of `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalExtension\PortalExtensionActivateActionInterface` and `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalExtension\PortalExtensionDeactivateActionInterface`
- Implement `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalExtension\PortalExtensionActivateActionInterface` in `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalExtension\PortalExtensionActivate`
- Implement `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalExtension\PortalExtensionDeactivateActionInterface` in `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalExtension\PortalExtensionDeactivate`
- Implement `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalExtension\PortalExtensionFindActionInterface` in `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalExtension\PortalExtensionFind`
- Implement `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNodeConfiguration\PortalNodeConfigurationGetActionInterface` in `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNodeConfiguration\PortalNodeConfigurationGet`
- Implement `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNodeConfiguration\PortalNodeConfigurationSetActionInterface` in `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNodeConfiguration\PortalNodeConfigurationSet`

- Add exception code `1642863637` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNodeConfiguration\PortalNodeConfigurationSet::set` when the payload has an invalid portal node key
- Add exception code `1642863638` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNodeConfiguration\PortalNodeConfigurationSet::set` when the payload value is not JSON serializable
- Add exception code `1642863639` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNodeConfiguration\PortalNodeConfigurationSet::set` when writing to the database fails
- Add exception code `1642863472` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNodeConfiguration\PortalNodeConfigurationGet::get` when the configuration value is not a valid JSON
- Add exception code `1642863473` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNodeConfiguration\PortalNodeConfigurationGet::get` when the configuration value is not a JSON array or JSON object
- Add migration `\Heptacom\HeptaConnect\Storage\ShopwareDal\Migration\Migration1642624782CreatePortalNodeConfigurationTable` to add table for portal node configuration and migrate from the previous storage
- Add exception code `1642937283` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Migration\Migration1642624782CreatePortalNodeConfigurationTable::migrate` when the JSON value from the old storage cannot be parsed
- Add exception code `1642937284` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Migration\Migration1642624782CreatePortalNodeConfigurationTable::migrate` when the JSON value from the old storage has an unexpected form
- Add exception code `1642937285` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Migration\Migration1642624782CreatePortalNodeConfigurationTable::migrate` when the read JSON from the old storage cannot be transformed into JSON for the new storage
- Add exception code `1642940744` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\EntityTypeAccessor::getIdsForTypes` when writing to the database fails
- Add exception code `1642951892` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Identity\IdentityMap::map` when writing to the database fails
- Implement `\Heptacom\HeptaConnect\Storage\Base\Action\Contract\Route\Delete\RouteDeleteActionInterface` in `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Route\RouteDelete` to delete routes
- Add exception code `1643144707` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Identity\IdentityPersist::persist` when check for same external id and having different mapping nodes fails
- Add exception code `1643144708` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Identity\IdentityPersist::persist` when check for same mapping node and having different external ids fails
- Add exception code `1643144709` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Identity\IdentityPersist::persist` when instructed identity mapping cannot be performed as related identities conflict
- Add exception code `1643149115` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Identity\IdentityPersist::persist` when the create-payload refers to a mapping node with an invalid mapping node key
- Add exception code `1643149116` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Identity\IdentityPersist::persist` when the update-payload refers to a mapping node with an invalid mapping node key
- Add exception code `1643149117` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Identity\IdentityPersist::persist` when the delete-payload refers to a mapping node with an invalid mapping node key
- Add exception code `1643149290` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Identity\IdentityPersist::persist` when the update-payload refers to an entry that is not present in storage
- Add exception code `1643149291` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Identity\IdentityPersist::persist` when the delete-payload refers to an entry that is not present in storage
- Add exception code `1643746495` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Identity\IdentityReflect::reflect` when writing to the database fails

- Implement `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Identity\IdentityOverviewActionInterface` in `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Identity\IdentityOverview` to list identities
- Add exception code `1643877525` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Identity\IdentityOverview::overview` when the payload refers to a mapping node with an invalid mapping node key
- Add exception code `1643877526` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Identity\IdentityOverview::overview` when the payload refers to a portal node with an invalid portal node key
- Add exception code `1643877527` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Identity\IdentityOverview::overview` when the criteria has an invalid sorting option
- Implement `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\IdentityError\IdentityErrorCreateActionInterface` in `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\IdentityError\IdentityErrorCreate` to store identity errors
- Add exception code `1645308762` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\IdentityError\IdentityErrorCreate::create` when the payload refers to a portal node with an invalid portal node key
- Add exception code `1645308763` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\IdentityError\IdentityErrorCreate::create` when the referenced mapping node by components is not known
- Add exception code `1645308764` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\IdentityError\IdentityErrorCreate::create` when writing to the database fails
- Implement `\Heptacom\HeptaConnect\Storage\Base\Bridge\Contract\StorageFacadeInterface` in `\Heptacom\HeptaConnect\Storage\ShopwareDal\Bridge\StorageFacade`
- Add query identifier parameter into `\Heptacom\HeptaConnect\Storage\ShopwareDal\Support\Query\QueryBuilder::__construct` that is added on query execution
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Support\Query\QueryIterator::fetchRow` to fetch a row keyed by column names
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Support\Query\QueryIterator::fetchColumn` to fetch a row and return its' first value
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Support\Query\QueryIterator::fetchSingleRow` to fetch a row keyed by column names and verify it is exactly a single row
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Support\Query\QueryIterator::iterateSafelyPaginated` to always paginate over rows keyed by column names even when no max result is given with the given safe pagination size parameter
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Support\Query\QueryBuilder::fetchSingleRow` to forward itself to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Support\Query\QueryIterator::fetchSingleRow`
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Support\Query\QueryBuilder::fetchSingleValue` to forward itself to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Support\Query\QueryIterator::fetchSingleValue`
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Support\Query\QueryBuilder::iterateRows` to forward itself to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Support\Query\QueryIterator::iterate`
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Support\Query\QueryBuilder::iterateColumn` to forward itself to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Support\Query\QueryIterator::iterateColumn`
- Add exception code `1645901524` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Support\Query\QueryIterator::iterateSafelyPaginated` when an invalid safe fetch size is given
- Add exception code `1645901525` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Support\Query\QueryIterator::iterateSafelyPaginated` when the query will be paginated without order statement
- Add exception code `1645901522` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Support\Query\QueryIterator::fetchSingleRow` when more than 1 row can be fetched from a query that expects only a single row
- Add factory `\Heptacom\HeptaConnect\Storage\ShopwareDal\Support\Query\QueryFactory` with configurable fallback pagination size for every builder
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\EntityTypeAccessor::LOOKUP_QUERY` as `992a88ac-a232-4d99-b1cc-4165da81ba77` to identify a query used for looking up entity types
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\JobTypeAccessor::LOOKUP_QUERY` as `28ef8980-146b-416c-8338-f1e394ac8c5f` to identify a query used for looking up job types

- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\RouteCapabilityAccessor::FETCH_QUERY` as `93fd2b30-ca58-4d60-b29e-d14115b5ea2b` to identify a query used for reading route capability data
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\WebHttpHandlerAccessor::FETCH_QUERY` as `900bdcb4-3a2a-4092-9eed-f5902e97b02f` to identify a query used for reading web HTTP handler data
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\WebHttpHandlerPathAccessor::FETCH_QUERY` as `f683453e-336f-4913-8bb9-aa0e34745f97` to identify a query used for reading web HTTP handler path data
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Identity\IdentityMap::MAPPING_NODE_QUERY` as `0d104088-b0d4-4158-8f95-0bc8a6880cc8` to identify a query used for loading related mapping nodes
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Identity\IdentityMap::MAPPING_QUERY` as `3c3f73e2-a95c-4ff3-89c5-c5f166195c24` to identify a query used for loading related mappings
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Identity\IdentityOverview::OVERVIEW_QUERY` as `510bb5ac-4bcb-4ddf-927c-05971298bc55` to identify a query used for loading an overview page for identities
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Identity\IdentityPersist::TYPE_LOOKUP_QUERY` as `4adbdc58-1ec7-45c0-9a5b-0ac983460505` to identify a query used for looking up related entity types
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Identity\IdentityPersist::BUILD_DELETE_PAYLOAD_QUERY` as `db92d189-494e-4d0b-be0b-492e4ded99c1` to identify a query used for reading identities that have to be deleted
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Identity\IdentityPersist::BUILD_UPDATE_PAYLOAD_QUERY` as `ddad865c-0608-42cd-89f1-148a44ed8f31` to identify a query used for reading identities that have be updated
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Identity\IdentityPersist::VALIDATE_CONFLICTS_QUERY` as `38d26bce-b577-4def-9fe3-d055cb63495d` to identify a query used for identifying possible conflicts
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Identity\IdentityPersist::VALIDATE_MERGE_QUERY` as `d8bb9156-edcc-4b1b-8e7e-fae2e8932434` to identify a query used for identifying possible merges
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\IdentityError\IdentityErrorCreate::LOOKUP_QUERY` as `95f2537a-eda2-4123-824d-72f6c871e8a8` to identify a query used for looking up related mapping nodes
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Job\JobCreate::PAYLOAD_LOOKUP_QUERY` as `b2234327-93a0-4854-ac52-fba75f71da74` to identify a query used for looking up payload entries
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Job\JobDelete::DELETE_QUERY` as `f60b01fc-8f9a-4a37-a009-a00db9a64b11` to identify a query used for deleting jobs
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Job\JobDelete::LOOKUP_QUERY` as `c1c41a80-6aec-4499-a07a-26ee57b07594` to identify a query used for looking up jobs that can be deleted
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Job\JobFinishedList::LIST_QUERY` as `008ced6c-7517-46f8-a8a0-8f3c31b50467` to identify a query used for listing finished jobs
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Job\JobGet::FETCH_QUERY` as `809ecd5e-291f-417c-9c76-003c7ead65e9` to identify a query used for reading job data
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalExtension\PortalExtensionFind::LOOKUP_QUERY` as `82bb12c6-ed9c-4646-901a-4ff7e8e4e88c` to identify a query used for looking up portal extension configurations
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalExtension\PortalExtensionSwitchActive::CLASS_NAME_LOOKUP_QUERY` as `a6bbbe3b-bf42-455d-824e-8c1aac4453b6` to identify a query used for looking up class name references
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalExtension\PortalExtensionSwitchActive::ID_LOOKUP_QUERY` as `2fc478d7-4f03-4a3d-a335-d6daf4244c27` to identify a query used for looking up existing configuration ids
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalExtension\PortalExtensionSwitchActive::SWITCH_QUERY` as `5444ccf3-cf11-4a5b-bf5f-8c268dce9c1a` to identify a query used for switching active states of portal extensions
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNode\PortalNodeDelete::DELETE_QUERY` as `219156bb-0598-49df-8205-6d10e8f92a61` to identify a query used for deleting portal nodes
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNode\PortalNodeDelete::LOOKUP_QUERY` as `aafca974-b95e-46ea-a680-834a93d13140` to identify a query used for looking up portal nodes that can be deleted
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNode\PortalNodeGet::FETCH_QUERY` as `efbd19ba-bc8e-412c-afb2-8a21f35e21f9` to identify a query used for reading portal node data

- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNode\PortalNodeList::LIST_QUERY` as `52e85ba9-3610-403b-be28-b8d138481ace` to identify a query used for listing up all portal nodes
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Route\ReceptionRouteList::LIST_QUERY` as `a2dc9481-5738-448a-9c85-617fec45a00d` to identify a query used for listing up all routes that are configured for reception
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Route\RouteDelete::LOOKUP_QUERY` as `b270142d-c897-4d1d-bddb-7641fbfb95a2` to identify a query used for looking up routes to delete
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Route\RouteDelete::DELETE_QUERY` as `384f50ca-1e0a-464b-80fd-824fc83b87ca` to identify a query used for deleting routes
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Route\RouteFind::LOOKUP_QUERY` as `1f0d7c11-0d1c-4834-8b15-148d826d64e8` to identify a query used for looking up routes
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Route\RouteGet::FETCH_QUERY` as `24ab04cd-03f5-40c8-af25-715856281314` to identify a query used for reading route data
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Route\RouteOverview::OVERVIEW_QUERY` as `6cb18ac6-6f5a-4d31-bed3-44849eb51f6f` to identify a query used for loading an overview page for routes
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\RouteCapability\RouteCapabilityOverview::OVERVIEW_QUERY` as `329b4aa3-e576-4930-b89f-c63dca05c16e` to identify a query used for loading an overview page for route capabilities
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\WebHttpHandlerConfiguration\WebHttpHandlerConfigurationFind::LOOKUP_QUERY` as `6c5db7b-004d-40c8-b9cc-53707aab658b` to identify a query used for looking up HTTP handler configurations
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNode\PortalNodeOverview::OVERVIEW_QUERY` as `478b14da-d0a8-44fd-bd1a-0a60ef948dd7` to identify a query used for loading an overview page for portal nodes
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNodeStorage\PortalNodeStorageDelete::DELETE_QUERY` as `40e42cd4-4ac3-4304-8cfc-9083d37e81cd` to identity query used for deleting portal node storage entries
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNodeStorage\PortalNodeStorageDelete::DELETE_EXPIRED_QUERY` as `1972fcfd-5d64-4bce-a6b5-19cb6a8ad671` to identity query used for deleting expired portal node storage entries
- Add exception code `1646209690` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNodeStorage\PortalNodeStorageDelete::delete` when writing to the database fails
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNodeStorage\PortalNodeStorageClear::CLEAR_QUERY` as `1087e0dc-07fe-48d7-903c-9353167c3e89` to identity query used for deleting all portal node storage entries
- Add exception code `1646209691` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNodeStorage\PortalNodeStorageClear::clear` when writing to the database fails
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNodeStorage\PortalNodeStorageGet::FETCH_QUERY` as `679d6e76-bb9c-410d-ac22-17c64afcb7cc` to identity query used for reading portal node storage entries
- Add exception code `1646341933` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNodeStorage\PortalNodeStorageSet::set` when writing to the database fails
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNodeStorage\PortalNodeStorageSet::UPDATE_PREPARATION_QUERY` as `75fada39-34f0-4e03-b3b5-141da358181d` to identity query used for reading portal node storage entries to prepare update statements
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNodeStorage\PortalNodeStorageList::FETCH_QUERY` as `7e532256-22d2-492e-8e76-ab1649ddc4e0` to identity query used for reading all portal node storage entries
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNodeConfiguration\PortalNodeConfigurationGet::FETCH_QUERY` as `be4a9934-2ab2-4c62-8a86-4600c96bc7be` to identify a query used for loading an overview page for portal nodes
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Identity\IdentityReflect::LOOKUP_EXISTING_MAPPING_QUERY` as `64211df0-e928-4fc9-87c1-09a4c03cf98a` to identify a query used for looking up existing mappings
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Identity\IdentityReflect::LOOKUP_EXISTING_MAPPING_NODE_QUERY` as `f6b0f467-0a73-4e1f-ad75-d669899df133` to identify a query used for looking up existing mapping nodes

- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Support\Id` as central utility for generation and converting UUIDs
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Support\DateTime` as central utility for converting dates from and into storage layer acceptable formats
- Implement `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\FileReference\FileReferenceGetRequestActionInterface` in `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\FileReference\FileReferenceGetRequestAction`
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\FileReference\FileReferenceGetRequestAction::FETCH_QUERY` as `25e53ac0-de53-4039-a790-253fb5803fec` to identity query used for fetching stored requests of file references
- Implement `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\FileReference\FileReferencePersistRequestActionInterface` in `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\FileReference\FileReferencePersistRequestAction`
- Add migration `\Heptacom\HeptaConnect\Storage\ShopwareDal\Migration\Migration1645820922AddFileReferenceRequest` to create a table for stored requests of file references
- Add class `\Heptacom\HeptaConnect\Storage\ShopwareDal\StorageKey\FileReferenceRequestStorageKey` as storage key for stored requests of file references
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Job\JobFail::FIND_QUERY` as `9b00334a-cc0b-4017-a9dc-e2520a872064` to identity query used for reading job ids
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Job\JobFail::UPDATE_QUERY` as `2d59f1a4-4baf-4cda-b762-16fb5beda452` to identity query used for updating job states
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Job\JobFinish::FIND_QUERY` as `84e5495d-4733-4e8a-b775-aafba23daa8c` to identity query used for reading job ids
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Job\JobFinish::UPDATE_QUERY` as `393a0ae1-5f42-4a49-96a3-9a23c26e6bd2` to identity query used for updating job states
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Job\JobSchedule::FIND_QUERY` as `87c10b4f-3dcd-460d-ba04-b38acbad6cbe` to identity query used for reading job ids
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Job\JobSchedule::UPDATE_QUERY` as `72372e2f-6e02-470b-89d5-b65ee88024b5` to identity query used for updating job states
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Job\JobStart::FIND_QUERY` as `1bbfc5fe-756c-4171-b645-ad2a6c10f4e7` to identity query used for reading job ids
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Job\JobStart::UPDATE_QUERY` as `0803daca-3ca7-44c4-a492-42cc51e46854` to identity query used for updating job states
- Add migration `\Heptacom\HeptaConnect\Storage\ShopwareDal\Migration\Migration1643220550CreatePortalNodeAliasColumn` to add aliases to portal node and migrate them from `heptacom/heptaconnect-bridge-shopware-platform` if applicable
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\PortalNodeAliasAccessor` to access portal node aliases in a cache manner
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\PortalNodeAliasAccessor::ID_LOOKUP_QUERY` as `8f493191-2ba8-4c9f-b4ff-641fc1afdc56` to identify query used for looking up portal node ids by aliases
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\PortalNodeAliasAccessor::ALIAS_LOOKUP_QUERY` as `81bd204c-97c0-4259-bf82-8b835f2f0237` to identify query used for looking up portal node aliases by ids
- Implement `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNodeAlias\PortalNodeAliasFindActionInterface` in `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNodeAlias\PortalNodeAliasFind`
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNodeAlias\PortalNodeAliasFind::FIND_QUERY` as `8ffc1022-c03b-4f3f-a2f6-5807710dbb6f` to identify query used for finding portal node ids by aliases
- Implement `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNodeAlias\PortalNodeAliasGetActionInterface` in `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNodeAlias\PortalNodeAliasGet`
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNodeAlias\PortalNodeAliasGet::FETCH_QUERY` as `f3e31372-bc6b-444d-99ee-38b74f9cf9fc` to identify query used for finding portal node aliases by their ids
- Implement `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNodeAlias\PortalNodeAliasOverviewActionInterface` in `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNodeAlias\PortalNodeAliasOverview`
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNodeAlias\PortalNodeAliasOverview::OVERVIEW_QUERY` as `8467ced0-3575-410f-8155-e36e7e8f0e0b` to identify query used for loading an overview page for portal node aliases

- Implement `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\PortalNodeAlias\PortalNodeAliasSetActionInterface` in `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNodeAlias\PortalNodeAliasSet`
- Add exception code `1647941560` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNodeAlias\PortalNodeAliasOverview::overview` when the criteria has an invalid sorting option
- Add exception code `1645446078` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNodeAlias\PortalNodeAliasSet::set` when the payload has an invalid portal node key
- Add exception code `1645446809` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNodeAlias\PortalNodeAliasSet::set` when the payload has an empty alias
- Add exception code `1645446810` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNodeAlias\PortalNodeAliasSet::set` when the payload has an already used alias
- Add exception code `1645448849` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNodeAlias\PortalNodeAliasSet::set` when writing to the database fails

Changed

- Change interface of `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Route\ReceptionRouteList` from `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Listing\ReceptionRouteListActionInterface` to `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\ReceptionRouteListActionInterface`
- Change interface of `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Route\RouteOverview` from `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Overview\RouteOverviewActionInterface` to `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\RouteOverviewActionInterface`
- Change interface of `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Route\RouteFind` from `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Find\RouteFindActionInterface` to `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\RouteFindActionInterface`
- Change interface of `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Route\RouteGet` from `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Get\RouteGetActionInterface` to `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\RouteGetActionInterface`
- Change interface of `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Route\RouteCreate` from `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Create\RouteCreateActionInterface` to `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\RouteCreateActionInterface`
- Change interface of `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\RouteCapability\RouteCapabilityOverview` from `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\RouteCapability\Overview\RouteCapabilityOverviewActionInterface` to `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\RouteCapability\RouteCapabilityOverviewActionInterface`
- Change interface of `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\WebHttpHandlerConfiguration\WebHttpHandlerConfigurationFind` from `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\WebHttpHandlerConfiguration\Find\WebHttpHandlerConfigurationFindActionInterface` to `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\WebHttpHandlerConfiguration\WebHttpHandlerConfigurationFindActionInterface`
- Change interface of `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\WebHttpHandlerConfiguration\WebHttpHandlerConfigurationSet` from `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\WebHttpHandlerConfiguration\Set\WebHttpHandlerConfigurationSetActionInterface` to `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\WebHttpHandlerConfiguration\WebHttpHandlerConfigurationSetActionInterface`
- Rename `\Heptacom\HeptaConnect\Storage\ShopwareDal\EntityMapper` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Identity\IdentityMap` and implement `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Identity\IdentityMapActionInterface`
- Rename `\Heptacom\HeptaConnect\Storage\ShopwareDal\EntityReflector` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Identity\IdentityReflect` and implement `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Identity\IdentityReflectActionInterface`

- Rename `\Heptacom\HeptaConnect\Storage\ShopwareDal\MappingPersister\MappingPersister` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Identity\IdentityPersist` and implement `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Identity\IdentityPersistActionInterface`
- Remove exception code `1637467903` from `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\RouteCapability\RouteCapabilityOverview::overview` expect exception code `1645901521` instead
- Remove exception code `1637467906` from `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Route\RouteFind::find` expect exception code `1645901521` instead
- Move exception code `1637467900` from `\Heptacom\HeptaConnect\Storage\ShopwareDal\Support\Query\QueryIterator::doIterate` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Support\Query\QueryIterator::getExecuteStatement` that is used as central point for this exception to happen
- Remove exception code `1637467905` from `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Route\RouteOverview::overview` expect exception code `1637467900` instead
- Remove exception code `1637542091` from `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\WebHttpHandlerConfiguration\WebHttpHandlerConfigurationFind::find` expect exception code `1645901522` instead
- Remove exception code `1637467901` from `\Heptacom\HeptaConnect\Storage\ShopwareDal\RouteCapabilityAccessor::getIdsForNames` expect exception code `1637467900` instead
- Remove exception code `1637467899` from `\Heptacom\HeptaConnect\Storage\ShopwareDal\WebHttpHandlerAccessor::getIdsForHandlers` expect exception code `1637467900` instead
- Remove exception code `1637467898` from `\Heptacom\HeptaConnect\Storage\ShopwareDal\WebHttpHandlerPathAccessor::getIdsForPaths` expect exception code `1637467900` instead
- Change dependency in `\Heptacom\HeptaConnect\Storage\ShopwareDal\RouteCapabilityAccessor` from `\Doctrine\DBAL\Connection` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Support\Query\QueryFactory`
- Add dependency `\Heptacom\HeptaConnect\Storage\ShopwareDal\Support\Query\QueryFactory` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\WebHttpHandlerAccessor`
- Add dependency `\Heptacom\HeptaConnect\Storage\ShopwareDal\Support\Query\QueryFactory` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\WebHttpHandlerPathAccessor`
- Rename `\Heptacom\HeptaConnect\Storage\ShopwareDal\StorageKey\MappingExceptionStorageKey` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\StorageKey\IdentityErrorStorageKey`
- Add dependency `\Heptacom\HeptaConnect\Storage\ShopwareDal\PortalNodeAliasAccessor` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\StorageKeyGenerator` to support alias storage key serialization and deserialization
- Use `\Heptacom\HeptaConnect\Storage\Base\AliasAwarePortalNodeStorageKey` as alias aware implementation for `\Heptacom\HeptaConnect\Storage\ShopwareDal\StorageKey\PortalNodeStorageKey`

Removed

- Remove class `\Heptacom\HeptaConnect\Storage\ShopwareDal\Job` as base contract has been removed
- Remove class `\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\JobPayloadRepository` as base contract has been removed
- Remove class `\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\JobRepository` as base contract has been removed
- Remove class `\Heptacom\HeptaConnect\Storage\ShopwareDal\StorageKey\JobPayloadStorageKey` as base contract has been removed and its support in `\Heptacom\HeptaConnect\Storage\ShopwareDal\StorageKeyGenerator`
- Remove implementation `\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\PortalNodeRepositoryContract::read` in favour of `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNode\PortalNodeGet::get` that allows for optimizations for different use-cases
- Remove implementation `\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\PortalNodeRepositoryContract::listAll` in favour of `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNode\PortalNodeList::list` that allows for optimizations for different use-cases

- Remove implementation `\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\PortalNodeRepositoryContract::listByClass` in favour of `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNode\PortalNodeOverview::overview` that allows for optimizations for different use-cases
- Remove implementation `\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\PortalNodeRepositoryContract::create` in favour of `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNode\PortalNodeCreate::create` that allows for optimizations for different use-cases
- Remove implementation `\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\PortalNodeRepositoryContract::create` in favour of `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNode\PortalNodeDelete::delete` that allows for optimizations for different use-cases
- Remove implementation `\Heptacom\HeptaConnect\Storage\ShopwareDal\ConfigurationStorage::getConfiguration` in favour of `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNodeConfiguration\PortalNodeConfigurationGet::get` that allows for optimizations for different use-cases
- Remove implementation `\Heptacom\HeptaConnect\Storage\ShopwareDal\ConfigurationStorage::setConfiguration` in favour of `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNodeConfiguration\PortalNodeConfigurationSet::set` that allows for optimizations for different use-cases
- Remove previously deprecated `\Heptacom\HeptaConnect\Storage\ShopwareDal\StorageKey\CronjobStorageKey`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\StorageKey\CronjobRunStorageKey`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\CronjobRepository` and `\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\CronjobRunRepository` as the feature of cronjobs in its current implementation is removed
- Remove previously deprecated `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Cronjob\CronjobCollection`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Cronjob\CronjobDefinition`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Cronjob\CronjobEntity`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Cronjob\CronjobRunCollection`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Cronjob\CronjobRunDefinition`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Cronjob\CronjobRunEntity` as the feature of cronjobs in its current implementation is removed
- Add migration `\Heptacom\HeptaConnect\Storage\ShopwareDal\Migration\Migration1642885343RemoveCronjobAndCronjobRunTable` to remove the tables `heptaconnect_cronjob` and `heptaconnect_cronjob_run` as the feature of cronjobs in its current implementation is removed
- Replace dependencies in `\Heptacom\HeptaConnect\Storage\ShopwareDal\EntityTypeAccessor` from `\Shopware\Core\Framework\DataAbstractionLayer\EntityRepositoryInterface` to `\Doctrine\DBAL\Connection` to drop Shopware DAL usage
- Remove implementation `\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\MappingRepository::listByNodes` from removed contract `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\MappingRepositoryContract::listByNodes`
- Remove implementation `\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\MappingRepository::listUnsavedExternalIds` from removed contract `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\MappingRepositoryContract::listUnsavedExternalIds`
- Remove implementation `\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\MappingRepository::updateExternalId` from removed contract `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\MappingRepositoryContract::updateExternalId`
- Remove implementation `\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\MappingNodeRepository::listByTypeAndPortalNodeAndExternalId` from removed contract `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\MappingNodeRepositoryContract::listByTypeAndPortalNodeAndExternalId`
- Remove implementation `\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\MappingNodeRepository::create` from removed contract `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\MappingNodeRepositoryContract::create`
- Remove deprecated `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Route\RouteDefinition`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Route\RouteEntity` and `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Route\RouteCollection`
- Integrate `\Heptacom\HeptaConnect\Storage\ShopwareDal\ResourceLockStorage` into `heptacom/heptaconnect-core` as `\Heptacom\HeptaConnect\Core\Parallelization\ResourceLockStorage`
- Remove unused composer dependency `symfony/lock: >=4`

- Remove implementation `\Heptacom\HeptaConnect\Storage\ShopwareDal\PortalStorage::unset` and `\Heptacom\HeptaConnect\Storage\ShopwareDal\PortalStorage::deleteMultiple` in favour of `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNodeStorage\PortalNodeStorageDelete::delete` that allows for optimizations for different use-cases
- Remove implementation `\Heptacom\HeptaConnect\Storage\ShopwareDal\PortalStorage::clear` in favour of `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNodeStorage\PortalNodeStorageClear::clear` that allows for optimizations for different use-cases
- Remove implementation `\Heptacom\HeptaConnect\Storage\ShopwareDal\PortalStorage::getValue`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\PortalStorage::getType`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\PortalStorage::has` and `\Heptacom\HeptaConnect\Storage\ShopwareDal\PortalStorage::getMultiple` in favour of `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNodeStorage\PortalNodeStorageGet::get` that allows for optimizations for different use-cases
- Remove implementation `\Heptacom\HeptaConnect\Storage\ShopwareDal\PortalStorage::set` in favour of `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNodeStorage\PortalNodeStorageSet::set` that allows for optimizations for different use-cases
- Remove implementation `\Heptacom\HeptaConnect\Storage\ShopwareDal\PortalStorage::list` in favour of `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\PortalNodeStorage\PortalNodeStorageList::list` that allows for optimizations for different use-cases
- Remove implementation `\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\MappingRepository` as base contract has been removed
- Remove implementation `\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\MappingExceptionRepository` as base contract has been removed
- Remove implementation `\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\MappingNodeRepository` as base contract has been removed
- Remove unused trait `\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\EntityRepositoryChecksTrait` as all using implementations have been removed
- Remove unused `\Heptacom\HeptaConnect\Storage\ShopwareDal\DalAccess`
- Remove deprecated `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\EntityType\EntityTypeCollection`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\EntityType\EntityTypeDefinition`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\EntityType\EntityTypeEntity`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Job\JobCollection`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Job\JobDefinition`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Job\JobEntity`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Job\JobPayloadCollection`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Job\JobPayloadDefinition`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Job\JobPayloadEntity`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Job\JobTypeCollection`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Job\JobTypeDefinition`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Job\JobTypeEntity`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Mapping\MappingCollection`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Mapping\MappingDefinition`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Mapping\MappingEntity`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Mapping\MappingErrorMessageCollection`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Mapping\MappingErrorMessageDefinition`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Mapping\MappingErrorMessageEntity`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Mapping\MappingNodeCollection`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Mapping\MappingNodeDefinition`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Mapping\MappingNodeEntity`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\PortalNode\PortalNodeCollection`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\PortalNode\PortalNodeDefinition`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\PortalNode\PortalNodeEntity`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\PortalNode\PortalNodeStorageCollection`,

`\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\PortalNode\PortalNodeStorageDefinition`,
`\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\PortalNode\PortalNodeStorageEntity`

- Remove unused `\Heptacom\HeptaConnect\Storage\ShopwareDal\ContextFactory`
- Remove implementation `\Heptacom\HeptaConnect\Storage\ShopwareDal\StorageKeyGenerator::generateKeys` as interface `\Heptacom\HeptaConnect\Storage\Base\Contract\StorageKeyGeneratorContract::generateKey` is removed
- Remove implementation `\Heptacom\HeptaConnect\Storage\ShopwareDal\StorageKey\MappingStorageKey` as base contract has been removed
- Remove support for `doctrine/dbal: >=2.1 <2.11`

st changes

7.1.105 [0.8.5] - 2022-02-01

Fixed

- Reset array keys after merging mapping nodes in `\Heptacom\HeptaConnect\Storage\ShopwareDal\MappingPersister\MappingPersister` to avoid InvalidArgumentException `Expected input to be non associative array.` to get thrown by `\Shopware\Core\Framework\DataAbstractionLayer\Write\EntityWriter`.

7.1.106 [0.8.4] - 2022-01-22

Added

- The `\Heptacom\HeptaConnect\Storage\ShopwareDal\MappingPersister\MappingPersister` will now attempt to merge mapping-nodes when there are no conflicts. Now mappings can be integrated into an existing mapping-node during a reception.

7.1.107 [0.8.3] - 2022-01-05

Changed

- Add migration `\Heptacom\HeptaConnect\Storage\ShopwareDal\Migration\Migration1641403938AddChecksumIndexToJobPayloadTable` to add index to `checksum` to table `heptaconnect_job_payload` for improved listings and searches

7.1.108 [0.8.2] - 2021-12-30

Fixed

- Use target portal node key in `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Route\RouteFind` to query the target portal node instead of using the source portal node key

7.1.109 [0.8.1] - 2021-11-22

Fixed

- Replace exception code `1637467902` with `1637542091` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\WebHttpHandlerConfiguration\WebHttpHandlerConfigurationFind::find` when query execution could not return a ResultStatement

7.1.110 [0.8.0] - 2021-11-22

Added

- Add migration `\Heptacom\HeptaConnect\Storage\ShopwareDal\Migration\Migration1632763825RenameDatasetEntityTypeTable` to rename database table `heptaconnect_dataset_entity_type` to `heptaconnect_entity_type`
- Add migration `\Heptacom\HeptaConnect\Storage\ShopwareDal\Migration\Migration1629643769AddJobStartAndFinishFields` to add `started_at` and `finished_at` datetime columns into table `heptaconnect_job` for job processing tracking
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\JobRepository::start` to implement new `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\JobRepositoryContract::start` for tracking the start of job processing
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\JobRepository::finish` to implement new `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\JobRepositoryContract::finish` for tracking the stop of job processing
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\JobRepository::cleanup` and `\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\JobPayloadRepository::cleanup` to implement new `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\JobRepositoryContract::cleanup` and `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\JobPayloadRepositoryContract::cleanup` for cleaning up executed jobs and their payloads
- Add migration `\Heptacom\HeptaConnect\Storage\ShopwareDal\Migration\Migration1635019143EntityTypeIndexHappenedAtColumns` to add descending indices to `created_at` and `updated_at` to table `heptaconnect_entity_type` for improved listings and searches

- Add migration `\Heptacom\HeptaConnect\Storage\ShopwareDal\Migration\Migration1635019144JobIndexHappenedAtColumns` to add descending indices to `created_at` and `updated_at` to table `heptaconnect_job` for improved listings and searches
- Add migration `\Heptacom\HeptaConnect\Storage\ShopwareDal\Migration\Migration1635019145JobIndexNewHappenedAtColumns` to add descending indices to `started_at` and `finished_at` to table `heptaconnect_job` for improved listings and searches
- Add migration `\Heptacom\HeptaConnect\Storage\ShopwareDal\Migration\Migration1635019146JobPayloadIndexHappenedAtColumns` to add descending indices to `created_at` and `updated_at` to table `heptaconnect_job_payload` for improved listings and searches
- Add migration `\Heptacom\HeptaConnect\Storage\ShopwareDal\Migration\Migration1635019147JobTypeIndexHappenedAtColumns` to add descending indices to `created_at` and `updated_at` to table `heptaconnect_job_type` for improved listings and searches
- Add migration `\Heptacom\HeptaConnect\Storage\ShopwareDal\Migration\Migration1635019148MappingIndexHappenedAtColumns` to add descending indices to `created_at`, `updated_at` and `deleted_at` to table `heptaconnect_mapping` for improved listings and searches
- Add migration `\Heptacom\HeptaConnect\Storage\ShopwareDal\Migration\Migration1635019149MappingErrorMessageIndexHappenedAtColumns` to add descending indices to `created_at` and `updated_at` to table `heptaconnect_mapping_error_message` for improved listings and searches
- Add migration `\Heptacom\HeptaConnect\Storage\ShopwareDal\Migration\Migration1635019150MappingNodeIndexHappenedAtColumns` to add descending indices to `created_at`, `updated_at` and `deleted_at` to table `heptaconnect_mapping_node` for improved listings and searches
- Add migration `\Heptacom\HeptaConnect\Storage\ShopwareDal\Migration\Migration1635019151PortalNodeIndexHappenedAtColumns` to add descending indices to `created_at`, `updated_at` and `deleted_at` to table `heptaconnect_portal_node` for improved listings and searches
- Add migration `\Heptacom\HeptaConnect\Storage\ShopwareDal\Migration\Migration1635019152PortalNodeStorageIndexHappenedAtColumns` to add descending indices to `created_at`, `updated_at` and `deleted_at` to table `heptaconnect_portal_node_storage` for improved listings and searches
- Add migration `\Heptacom\HeptaConnect\Storage\ShopwareDal\Migration\Migration1635019153RouteIndexHappenedAtColumns` to add descending indices to `created_at`, `updated_at` and `deleted_at` to table `heptaconnect_route` for improved listings and searches
- Add implementation for `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Listing\ReceptionRouteListActionInterface` in `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Route\ReceptionRouteList`
- Add implementation for `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Overview\RouteOverviewActionInterface` in `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Route\RouteOverview`
- Add implementation for `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Find\RouteFindActionInterface` in `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Route\RouteFind`
- Add implementation for `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Get\RouteGetActionInterface` in `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Route\RouteGet`
- Add implementation for `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\Route\Create\RouteCreateActionInterface` in `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Route\RouteCreate`
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Support\Query\QueryIterator` to simplify DBAL paginated iteration
- Add migration `\Heptacom\HeptaConnect\Storage\ShopwareDal\Migration\Migration1635512814OnDeleteCascadeFromMappingNodeToMapping` to cascade delete from `heptaconnect_mapping_node` to `heptaconnect_mapping`
- Add migration `\Heptacom\HeptaConnect\Storage\ShopwareDal\Migration\Migration1635713039CreateRouteCapabilityTable` to create database table `heptaconnect_route_capability` to store route capability types
- Add migration `\Heptacom\HeptaConnect\Storage\ShopwareDal\Migration\Migration1635713040SeedReceptionRouteCapability` to add the reception capability type
- Add migration `\Heptacom\HeptaConnect\Storage\ShopwareDal\Migration\Migration1635713041CreateRouteToRouteCapabilityTable` to create database table `heptaconnect_route_has_capability` to connect routes to their capabilities
- Add migration `\Heptacom\HeptaConnect\Storage\ShopwareDal\Migration\Migration1635713042SeedReceptionCapabilityToRoute` to add every capability type to every route
- Add implementation for `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\RouteCapability\Overview\RouteCapabilityOverviewActionInterface` in `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\RouteCapability\RouteCapabilityOverview`
- Add custom `\Heptacom\HeptaConnect\Storage\ShopwareDal\Support\Query\QueryBuilder` based upon `\Doctrine\DBAL\Query\QueryBuilder` for parameterized pagination for easier SQL statement caching
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\RouteCapabilityAccessor` to read route capabilities efficiently for other internal operations
- Add exception code `1636505518` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Route\RouteOverview::overview` when the criteria has an invalid sorting option

- Add exception code `1636505519` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\RouteCapability\RouteCapabilityOverview::overview` when the criteria has an invalid sorting option
- Add exception code `1636573803` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Route\RouteCreate::create` when the payload refers to a source portal node with an invalid portal node
- Add exception code `1636573804` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Route\RouteCreate::create` when the payload refers to a target portal node with an invalid portal node
- Add exception code `1636573805` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Route\RouteCreate::create` when the payload refers to an unknown route capability
- Add exception code `1636573806` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Route\RouteCreate::create` when the payload refers to an unknown entity type
- Add exception code `1636573807` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Route\RouteCreate::create` when the key generator cannot generate a valid route key
- Add exception code `1636576240` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Route\RouteCreate::create` when writing to the database fails
- Add migration `\Heptacom\HeptaConnect\Storage\ShopwareDal\Migration\Migration1636817108CreateWebHttpHandlerPathTable` to create table `heptaconnect_web_http_handler_path` to hold indexed HTTP handler paths
- Add migration `\Heptacom\HeptaConnect\Storage\ShopwareDal\Migration\Migration1636817109CreateWebHttpHandlerTable` to create table `heptaconnect_web_http_handler` to hold HTTP handlers based upon their portal nodes and paths
- Add migration `\Heptacom\HeptaConnect\Storage\ShopwareDal\Migration\Migration1636817110CreateWebHttpHandlerConfigurationTable` to create table `heptaconnect_web_http_handler_configuration` to hold HTTP handler configurations
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\WebHttpHandlerAccessor` to read and insert HTTP handler entries efficiently for other internal operations
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\WebHttpHandlerPathAccessor` to read and insert HTTP handler paths entries efficiently for other internal operations
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\WebHttpHandlerPathIdResolver` to centralize path id prediction
- Add implementation for `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\WebHttpHandlerConfiguration\Find\WebHttpHandlerConfigurationFindActionInterface` in `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\WebHttpHandlerConfiguration\WebHttpHandlerConfigurationFind`
- Add implementation for `\Heptacom\HeptaConnect\Storage\Base\Contract\Action\WebHttpHandlerConfiguration\Set\WebHttpHandlerConfigurationSetActionInterface` in `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\WebHttpHandlerConfiguration\WebHttpHandlerConfigurationSet`
- Add exception code `1636827821` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\WebHttpHandlerConfiguration\WebHttpHandlerConfigurationSet::set` when the payload refers to an invalid portal node
- Add exception code `1636827822` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\WebHttpHandlerConfiguration\WebHttpHandlerConfigurationSet::set` when the payload refers to an HTTP handler path that could not be looked up or created
- Add exception code `1636827823` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\WebHttpHandlerConfiguration\WebHttpHandlerConfigurationSet::set` when the payload refers to an HTTP handler by path and portal node that could not be looked up or created
- Add exception code `1636827824` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\WebHttpHandlerConfiguration\WebHttpHandlerConfigurationSet::set` when writing to the database fails
- Add exception code `1637467897` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\WebHttpHandlerPathAccessor::getIdsForPaths` when `\array_combine` returns false
- Add exception code `1636528918` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Route\RouteOverview::overview` when the criteria has an invalid sorting option
- Add exception code `1637467898` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\WebHttpHandlerPathAccessor::getIdsForPaths` when query execution could not return a Statement
- Add exception code `1637467899` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\WebHttpHandlerAccessor::getIdsForHandlers` when query execution could not return a Statement

- Add exception code `1637467900` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Support\Query\QueryIterator::doIterate` when query execution could not return a ResultStatement
- Add exception code `1637467901` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\RouteCapabilityAccessor::getIdsForNames` when query execution could not return a ResultStatement
- Add exception code `1637467902` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\WebHttpHandlerConfiguration\WebHttpHandlerConfigurationFind::find` when query execution could not return a Statement
- Add exception code `1637467903` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\RouteCapability\RouteCapabilityOverview::overview` when query execution could not return a ResultStatement
- Add exception code `1637467905` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Route\RouteOverview::overview` when query execution could not return a ResultStatement
- Add exception code `1637467906` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Route\RouteFind::find` when query execution could not return a ResultStatement

Changed

- Change namespace from `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\DatasetEntityType` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\EntityType` and rename folder appropriately
- Change class name from `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\EntityType\DatasetEntityTypeDefinition` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\EntityType\EntityTypeDefinition`
- Change class name from `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\EntityType\DatasetEntityTypeCollection` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\EntityType\EntityTypeCollection`
- Change class name from `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\EntityType\DatasetEntityTypeEntity` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\EntityType\EntityTypeEntity`
- Change class name from `\Heptacom\HeptaConnect\Storage\ShopwareDal\DatasetEntityTypeAccessor` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\EntityTypeAccessor`
- Change method name from `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Mapping\MappingNodeEntity::getDatasetEntityClassName` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Mapping\MappingNodeEntity::getEntityType`
- Change method name from `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Mapping\MappingEntity::getDatasetEntityClassName` to `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Mapping\MappingEntity::getEntityType`
- Change parameter name of `\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\MappingRepository::listByPortalNodeAndType` from `$datasetEntityType` to `$entityType`
- Change parameter name of `\Heptacom\HeptaConnect\Storage\ShopwareDal\EntityTypeAccessor::getIdsForTypes` from `$datasetEntityClassNames` to `$entityTypes`
- Change parameter name of `\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\MappingNodeRepository::listByTypeAndPortalNodeAndExternalId` from `$datasetEntityClassName` to `$entityType`
- Change parameter name of `\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\MappingNodeRepository::listByTypeAndPortalNodeAndExternalIds` from `$datasetEntityClassName` to `$entityType`
- Change parameter name of `\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\MappingNodeRepository::create` from `$datasetEntityClassName` to `$entityType`
- Change parameter name of `\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\MappingNodeRepository::createList` from `$datasetEntityClassName` to `$entityType`
- Change parameter name of `\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\MappingRepository::listUnsavedExternalIds` from `$datasetEntityClassName` to `$entityType`

Deprecated

- Mark `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Route\RouteDefinition`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Route\RouteEntity` and `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Route\RouteCollection` as deprecated as DAL usage is discouraged

Removed

- Remove implementation `\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\RouteRepository::listBySourceAndEntityType` in favour of `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Route\ReceptionRouteList::list`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Route\RouteOverview::overview` and `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Route\RouteFind::find` that are optimized for different use-cases
- Remove implementation `\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\RouteRepository::read` in favour of `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Route\RouteGet::get` that is optimized for known use-cases
- Remove implementation `\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\RouteRepository::create` in favour of `\Heptacom\HeptaConnect\Storage\ShopwareDal\Action\Route\RouteCreate::create` that is optimized for known use-cases
- Remove `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Webhook\WebhookCollection`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Webhook\WebhookDefinition` and `\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Webhook\WebhookEntity` in favour of a storage independent solution
- Remove `\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\WebhookRepository` and `\Heptacom\HeptaConnect\Storage\ShopwareDal\StorageKey\WebhookStorageKey` in favour of a storage independent solution
- Remove support for `\Heptacom\HeptaConnect\Storage\ShopwareDal\StorageKey\WebhookStorageKey` in `\Heptacom\HeptaConnect\Storage\ShopwareDal\StorageKeyGenerator`
- Add `\Heptacom\HeptaConnect\Storage\ShopwareDal\Migration\Migration1636704625RemoveWebhookTable` to drop the `heptaconnect_webhook` table
- Remove support for `shopware/core: 6.2.*`
- Remove configuration merging from `\Heptacom\HeptaConnect\Storage\ShopwareDal\ConfigurationStorage::setConfiguration` which is already done by the core package

Fixed

- Change `\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\MappingExceptionRepository::create` so it includes a check for the success of `\json_encode`

7.1.111 [0.7.0] - 2021-09-25

Added

- Add support for composer dependency `ramsey/uuid: 4.*`
- Add implementation for `\Heptacom\HeptaConnect\Storage\Base\Contract\PortalStorageContract` in `\Heptacom\HeptaConnect\Storage\ShopwareDal\PortalStorage::clear`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\PortalStorage::getMultiple` and `\Heptacom\HeptaConnect\Storage\ShopwareDal\PortalStorage::deleteMultiple` to allow PSR simple cache compatibility
- New service `\Heptacom\HeptaConnect\Storage\ShopwareDal\MappingPersister` responsible for saving mappings after reception. Could improve usages of `\Heptacom\HeptaConnect\Storage\Base\Contract\Repository\MappingRepositoryContract`.

Changed

- Improve performance of `\Heptacom\HeptaConnect\Storage\ShopwareDal\EntityMapper::mapEntities`
- Improve performance of `\Heptacom\HeptaConnect\Storage\ShopwareDal\EntityReflector::reflectEntities`

Fixed

- Change string comparison on database layer from whitespace-unaware, case-insensitive to binary for jobs, job payloads, mappings, portal nodes, portal node storage, data entity class names so lookups are one-to-one which therefore affects behaviour `\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\JobPayloadRepository::add`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\JobRepository::add`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\MappingNodeRepository::listByTypeAndPortalNodeAndExternalId`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\MappingNodeRepository::listByTypeAndPortalNodeAndExternalIds`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\MappingNodeRepository::create`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\MappingNodeRepository::createList`,

`\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\MappingRepository::listByPortalNodeAndType`,
`\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\MappingRepository::listUnsavedExternalIds`,
`\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\MappingRepository::updateExternalId`,
`\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\PortalNodeRepository::listByClass`,
`\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\PortalNodeRepository::create`,
`\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\RouteRepository::listBySourceAndEntityType`,
`\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\RouteRepository::create`,
`\Heptacom\HeptaConnect\Storage\ShopwareDal\DatasetEntityTypeAccessor::getIdsForTypes`,
`\Heptacom\HeptaConnect\Storage\ShopwareDal\EntityMapper::mapEntities`,
`\Heptacom\HeptaConnect\Storage\ShopwareDal\EntityReflector::reflectEntities`,
`\Heptacom\HeptaConnect\Storage\ShopwareDal\PortalStorage::set`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\PortalStorage::unset`,
`\Heptacom\HeptaConnect\Storage\ShopwareDal\PortalStorage::getValue`,
`\Heptacom\HeptaConnect\Storage\ShopwareDal\PortalStorage::getType`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\PortalStorage::has`

- Disable HTML stripping from string columns in DAL for jobs, mappings and portal node storage so storing data will allow `<>` symbols which therefore affects behaviour `\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\JobRepository::add`,
`\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\MappingNodeRepository::listByTypeAndPortalNodeAndExternalId`,
`\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\MappingNodeRepository::listByTypeAndPortalNodeAndExternalIds`,
`\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\MappingRepository::listByPortalNodeAndType`,
`\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\MappingRepository::listUnsavedExternalIds`,
`\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\MappingRepository::updateExternalId`,
`\Heptacom\HeptaConnect\Storage\ShopwareDal\EntityMapper::mapEntities`,
`\Heptacom\HeptaConnect\Storage\ShopwareDal\EntityReflector::reflectEntities`,
`\Heptacom\HeptaConnect\Storage\ShopwareDal\PortalStorage::set`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\PortalStorage::unset`,
`\Heptacom\HeptaConnect\Storage\ShopwareDal\PortalStorage::getValue`,
`\Heptacom\HeptaConnect\Storage\ShopwareDal\PortalStorage::getType`, `\Heptacom\HeptaConnect\Storage\ShopwareDal\PortalStorage::has`
- `\Heptacom\HeptaConnect\Storage\ShopwareDal\EntityMapper` now respects soft-deletions of mappings and mapping nodes.
- `\Heptacom\HeptaConnect\Storage\ShopwareDal\EntityReflector` now respects soft-deletions of mappings and mapping nodes.

7.1.112 [0.5.0] - 2021-07-11

Deprecated

- Deprecate cronjobs to allow for new implementation at different point in time and with it
`\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Cronjob\CronjobCollection`,
`\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Cronjob\CronjobDefinition`,
`\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Cronjob\CronjobEntity`,
`\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Cronjob\CronjobRunCollection`,
`\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Cronjob\CronjobRunDefinition`,
`\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Cronjob\CronjobRunEntity`,
`\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\CronjobRepository`,
`\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\CronjobRunRepository`,
`\Heptacom\HeptaConnect\Storage\ShopwareDal\StorageKey\CronjobStorageKey`,
`\Heptacom\HeptaConnect\Storage\ShopwareDal\StorageKey\CronjobRunStorageKey`
- Deprecate webhooks to allow for new implementation at different point in time and with it
`\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Webhook\WebhookCollection`,
`\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Webhook\WebhookDefinition`,
`\Heptacom\HeptaConnect\Storage\ShopwareDal\Content\Webhook\WebhookEntity`,
`\Heptacom\HeptaConnect\Storage\ShopwareDal\Repository\WebhookRepository`,
`\Heptacom\HeptaConnect\Storage\ShopwareDal\StorageKey\WebhookStorageKey`

Fixed

- Fix bug and improved performance on entity reflection in
`\Heptacom\HeptaConnect\Storage\ShopwareDal\EntityReflector::reflectEntities` when empty entity collection has been passed in

7.1.113 [0.4.0] - 2021-07-03

Added

- Add support for preview portal node keys `\Heptacom\HeptaConnect\Storage\Base\PreviewPortalNodeKey` in `\Heptacom\HeptaConnect\Storage\ShopwareDal\ConfigurationStorage::getConfiguration`

Changed

- Improve performance of `\Heptacom\HeptaConnect\Storage\ShopwareDal\EntityMapper::mapEntities` by restructuring database queries
- Improve performance of `\Heptacom\HeptaConnect\Storage\ShopwareDal\EntityReflector::reflectEntities` by restructuring database queries