Search Criteria and Sort Clauses

Version compatibility
This recipe is compatible with **eZ Publish 5.3** and higher

Introduction

Search Criteria and Sort Clauses are `value` object classes used for building Search Query, to define filter criteria and ordering of the result set. eZ Platform (and eZ Publish Platform 5.x) provides a number of standard Criteria and Sort Clauses that you can use out of the box and that should cover the majority of use cases.

Example of standard ContentId criterion

```
<?php

namespace eZ\Publish\API\Repository\Values\Content\Query\Criterion;

use eZ\Publish\API\Repository\Values\Content\Query\Criterion;
use eZ\Publish\API\Repository\Values\Content\Query\Criterion\Operator\Specifications;
use eZ\Publish\API\Repository\Values\Content\Query\CriterionInterface;

/**
 * A criterion that matches content based on its id
 *
 * Supported operators:
 * - IN: will match from a list of ContentId
 * - EQ: will match against one ContentId
 */
class ContentId extends Criterion implements CriterionInterface
{
    /**
     * Creates a new ContentId criterion
     *
     * @param int|int[] $value One or more content Id that must be matched.
     *
     * @throws \InvalidArgumentException if a non numeric id is given
     * @throws \InvalidArgumentException if the value type doesn't match the operator
     */
    public function __construct( $value )
    {
        parent::__construct( null, null, $value );
    }

    public function getSpecifications()
    {
        $types = Specifications::TYPE_INTEGER | Specifications::TYPE_STRING;
        return array(
            new Specifications( Operator::IN, Specifications::FORMAT_ARRAY, $types ),
            new Specifications( Operator::EQ, Specifications::FORMAT_SINGLE, $types ),
        );
    }

    public static function createFromQueryBuilder( $target, $operator, $value )
    {
        return new self( $value );
    }
}
```

Search Engine Handling of Criteria and Sort Clauses

As Criterions and Sort Clauses are `value` objects which are used to define the Query from API perspective, they are are common for all storage engines. Each storage engine needs to implement its own `handler` for corresponding Criterion and Sort Clause `value` object, which will be used to translate the value object into storage specific search query.

Example of ContentId criterion handler in Legacy Storage Engine

```
<?php

namespace
eZ\Publish\Core\Persistence\Legacy\Content\Search\Common\Gateway\CriterionHandler;

use eZ\Publish\Core\Persistence\Legacy\Content\Search\Common\Gateway\CriterionHandler;
use
eZ\Publish\Core\Persistence\Legacy\Content\Search\Common\Gateway\CriteriaConverter;
use eZ\Publish\API\Repository\Values\Content\Query\Criterion;
use eZ\Publish\Core\Persistence\Database\SelectQuery;

/**
 * Content ID criterion handler
 */
class ContentId extends CriterionHandler
{
    /**
     * Check if this criterion handler accepts to handle the given criterion.
     *
     * @param \eZ\Publish\API\Repository\Values\Content\Query\Criterion $criterion
     *
     * @return boolean
     */
    public function accept( Criterion $criterion )
    {
        return $criterion instanceof Criterion\ContentId;
    }


    /**
     * Generate query expression for a Criterion this handler accepts
     *
     * accept() must be called before calling this method.
     *
     * @param
\eZ\Publish\Core\Persistence\Legacy\Content\Search\Common\Gateway\CriteriaConverter
$converter
     * @param \eZ\Publish\Core\Persistence\Database\SelectQuery $query
     * @param \eZ\Publish\API\Repository\Values\Content\Query\Criterion $criterion
     *
     * @return \eZ\Publish\Core\Persistence\Database\Expression
     */
    public function handle( CriteriaConverter $converter, SelectQuery $query,
Criterion $criterion )
    {
        return $query->expr->in(
            $this->dbHandler->quoteColumn( "id", "ezcontentobject" ),
            $criterion->value
        );
    }
}
```

Example of ContentId criterion handler in Solr Storage engine

```
<?php

namespace eZ\Publish\Core\Persistence\Solr\Content\Search\CriterionVisitor;

use eZ\Publish\Core\Persistence\Solr\Content\Search\CriterionVisitor;
use eZ\Publish\API\Repository\Values\Content\Query\Criterion;
use eZ\Publish\API\Repository\Values\Content\Query\Criterion\Operator;

/**
 * Visits the ContentId criterion
 */
class ContentIdIn extends CriterionVisitor
{
    /**
     * CHeck if visitor is applicable to current criterion
     *
     * @param Criterion $criterion
     *
     * @return boolean
     */
    public function canVisit( Criterion $criterion )
    {
        return
            $criterion instanceof Criterion\ContentId &&
            ( ( $criterion->operator ?: Operator::IN ) === Operator::IN ||
              $criterion->operator === Operator::EQ );
    }


    /**
     * Map field value to a proper Solr representation
     *
     * @param Criterion $criterion
     * @param CriterionVisitor $subVisitor
     *
     * @return string
     */
    public function visit( Criterion $criterion, CriterionVisitor $subVisitor = null )
    {
        return '(' .
            implode(
                ' OR ',
                array_map(
                    function ( $value )
                    {
                        return 'id:"' . $value . '"';
                    },
                    $criterion->value
                )
            ) .
            ')';
    }
}
```

Custom Criteria and Sort Clauses

Sometimes you will find that standard Criteria and Sort Clauses provided with eZ Publish are not sufficient for you needs. Most often this will be the case if you have developed a custom FieldType using external storage, which therefore can not be searched using standard Field Criterion.

> **On use of Field Criterion/SortClause with large databases**
> Field Criterion/SortClause does not perform well by design when using SQL database, so if you have a large database and want to use them you either need to wait and use Solr/ElasticSearch support when official later in 2015, or develop your own Custom Criterion / Sort Clause to avoid use of attributes (Fields) database table, and instead uses a custom simplified table which can handle the amount of data you have.

In this case you can implement a custom Criterion or Sort Clause, together with the corresponding handlers for the storage engine you are using.

Difference between Content and Location Search

These are two basic types of searches, you can either search for Locations or for Content. Each has dedicated methods in Search Service:

Type of search	Method in Search Service
Content	`findContent()`
Content	`findSingle()`
Location	`findLocations()`

All Criterions and Sort Clauses will be accepted with Location Search, but not all of them can be used with Content Search. Reason for this is that while one Location always has exactly one Content item, one Content item can have multiple Locations. In this context some Criterions and Sort Clauses would produce ambiguous queries and such will therefore not be accepted by Content Search.

Content Search will explicitly refuse to accept Criterions and Sort Clauses implementing these abstract classes:

- `eZ\Publish\API\Repository\Values\Content\Query\Criterion\Location`
- `eZ\Publish\API\Repository\Values\Content\SortClause\Criterion\Location`

How to configure your own Criterion and Sort Clause Handlers

After you have implemented your Criterion / Sort Clause and its handler, you will need to configure the handler for the service container using dedicated service tags for each type of search. Doing so will automatically register it and handle your Criterion / Search Clause when it is given as a parameter to one of the Search Service methods.

> You will find all the native handlers and the tags for the Legacy Storage Engine available in the eZ/Publish/Core/settings/storage_engines/legacy/**search_query_handlers.yml** file.

Tags

>=5.4.2

Available tags for Criterion handlers in Legacy Storage Engine are:

- `ezpublish.search.legacy.gateway.criterion_handler.content`
- `ezpublish.search.legacy.gateway.criterion_handler.location`

Available tags for Sort Clause handlers in Legacy Storage Engine are:

- `ezpublish.search.legacy.gateway.sort_clause_handler.content`
- `ezpublish.search.legacy.gateway.sort_clause_handler.location`

<=5.3

Available tags for Criterion handlers in Legacy Storage Engine are:

- `ezpublish.persistence.legacy.search.gateway.criterion_handler.content`

- `ezpublish.persistence.legacy.search.gateway.criterion_handler.location`

Available tags for Sort Clause handlers in Legacy Storage Engine are:

- `ezpublish.persistence.legacy.search.gateway.sort_clause_handler.content`
- `ezpublish.persistence.legacy.search.gateway.sort_clause_handler.location`

Example of registering ContentId Criterion handler in 5.4.x, common for both Content and Location Search

Registering Criterion handler

```
services:
    ezpublish.search.legacy.gateway.criterion_handler.common.content_id:
        class:
eZ\Publish\Core\Search\Legacy\Content\Common\Gateway\CriterionHandler\ContentId
        arguments: [@ezpublish.api.storage_engine.legacy.dbhandler]
        tags:
          - {name: ezpublish.search.legacy.gateway.criterion_handler.content}
          - {name: ezpublish.search.legacy.gateway.criterion_handler.location}
```

Example of registering Depth Sort Clause handler in 5.3.x for Location Search

Registering Sort Clause handler

```
ezpublish.persistence.legacy.search.gateway.sort_clause_handler.location.depth:
    class:
eZ\Publish\Core\Persistence\Legacy\Content\Search\Location\Gateway\SortClauseHandler\L
ocation\Depth
    arguments: [@ezpublish.api.storage_engine.legacy.dbhandler]
    tags:
        - {name:
ezpublish.persistence.legacy.search.gateway.sort_clause_handler.location}
```

See also
See also Symfony documentation about Service Container for passing parameters