User Interface Configuration

- Overview
- Configuration File Format
  - Migrate environment file to YAML
- Configuration Override
  - In 7.2 or above
  - In 7.1 or 7.0
- Configuration Reference
  - Production Mode
  - UI Core Settings
  - REST API Settings
  - Cache Settings - General
  - Cache Settings - Server Side Rendering (SSR)
  - Authentication Settings
  - Form Settings
  - Notification Settings
  - Submission Settings
  - Language Settings
  - Browse By Settings
  - Community-List Settings
  - Homepage Settings
  - Undo Settings
  - Item Access Labels
  - Item Page Settings
  - Theme Settings
  - Media Viewer Settings
    - Uploading video captioning files
  - Toggle end-user agreement and privacy policy
  - Settings for rendering Markdown, HTML and MathJax in metadata
  - Controlled Vocabularies in Search Filters
  - Universal (Server-side Rendering) Settings
  - Debug Settings

Overview

As the DSpace 7 User Interface is built on Angular.io, it aligns with many of the best practices of that platform & the surrounding community. One example is that our UI uses the TypeScript language. That said, you do NOT need to be deeply familiar with TypeScript to edit themes or other configuration.

In DSpace 7.2 and later, the UI Configuration format changed to support runtime configuration loading

As of DSpace 7.2, the UI configuration format has changed to YAML in order to support runtime configuration. This means that reloading configurations now simply requires restarting the UI (which generally takes a few seconds).

In DSpace 7.1 and below, you had to rebuild the UI anytime you modified a configuration setting. The UI configuration format was Typescript which required recompiling each time a setting changed.

Configuration File Format

In DSpace 7.2 and above, the Configuration format is now YAML and is located at `./config/config.*.yaml`.

In DSpace 7.1 and 7.0, the Configuration format was a Typescript file and was located at `./src/environments/environment.*.ts`. The structure of this file was essentially a JSON like format.

If you are upgrading from 7.0 or 7.1 to 7.2 (or later), you will either need to migrate your old configuration file (from Typescript to YAML) or start fresh. You can migrate your old (7.0 or 7.1) "environment.*.ts" configuration file to the new "config.*.yml" format (see below).

Migrate environment file to YAML

These instructions will allow you to migrate your old (7.0 or 7.1) "environment.*.ts" configuration file to the new "config.*.yml" format.

1. First, you will need to make minor modifications to the old "environment.*.ts" configuration file(s) to ensure it no longer imports any other files:

Replace all imports in environment.*.ts

```
// (1) FIRST, you must comment out or remove all 4 imports at the top of the file. For example:

//import { GlobalConfig } from '../config/global-config.interface';
//import { NotificationAnimationsType } from '../app/shared/notifications/models/notification-animations-type';
//import { BrowseByType } from '../app/browse-by/browse-by-switcher/browse-by-decorator';
//import { RestRequestMethod } from '../app/core/data/rest-request-method';

// (2) SECOND, replace those 4 imported objects with these 4 objects.
// Paste this code at the top of the file where those imports were.

interface GlobalConfig { }

enum NotificationAnimationsType {
    Fade = 'fade',
    FromTop = 'fromTop',
    FromRight = 'fromRight',
    FromBottom = 'fromBottom',
    FromLeft = 'fromLeft',
    Rotate = 'rotate',
    Scale = 'scale'
}

enum BrowseByType {
    Title = 'title',
    Metadata = 'metadata',
    Date = 'date'
}

enum RestRequestMethod {
    GET = 'GET',
    POST = 'POST',
    PUT = 'PUT',
    DELETE = 'DELETE',
    OPTIONS = 'OPTIONS',
    HEAD = 'HEAD',
    PATCH = 'PATCH'
}
```

2. Now, you are ready to run the "yarn env:yaml" command to transform this old configuration into the new format.

Migrate from environment.*.ts to config.*.yml

```
yarn env:yaml [relative-path-to-environment.ts] [optional-relative-path-to-YAML]

# For example, from the 7.2 (or above) root directory, run this:
# yarn env:yaml relative/path/to/old/environment.prod.ts config/config.prod.yml
```

3. Finally, you should replace the old environment.*.ts config file(s) with the stock versions. They continue to provide default configuration values, but customization should be done in the YAML files. If you had created *additional* environment files, those can be deleted.

Configuration Override

In 7.2 or above

Starting in 7.2, if you make a configuration update, you only need to restart the frontend. There is no need to rebuild unless you have made code changes in "/src" directory or similar.

The UI configuration files reside in the `./config/` folder in the Angular UI source code. The default configuration is provided in `config.yml`.

To change the default configuration values, you simply create (one or more) local files that override the parameters you need to modify. You can use `config.example.yml` as a starting point.

- For example, create a new `config.dev.yml` file in `config/` for a `development` environment;

- For example, create a new `config.prod.yml` file in `config/` for a `production` environment;

Configurations can also be overridden via one of the following

- Using Environment variables. All environment variables MUST (1) be prefixed with "DSPACE_", (2) use underscores as separators (no dots allowed), and (3) use all uppercase. Some examples are below:

```
# "ui" settings environment variables
ui.host => DSPACE_UI_HOST # The host name
ui.port => DSPACE_UI_PORT # The port number
ui.nameSpace => DSPACE_UI_NAMESPACE # The namespace
ui.ssl => DSPACE_UI_SSL # Whether the angular application uses SSL [true/false]

# "rest" settings environment variables
rest.host => DSPACE_REST_HOST # The host name of the REST application
rest.port => DSPACE_REST_PORT # The port number of the REST application
rest.nameSpace => DSPACE_REST_NAMESPACE # The namespace of the REST application
rest.ssl => DSPACE_REST_SSL # Whether the angular REST uses SSL [true/false]

# Other examples
defaultLanguage => DSPACE_DEFAULTLANGUAGE
mediaViewer.video => DSPACE_MEDIAVIEWER_VIDEO

# Multi-valued setting examples
# If a setting can have multiple values (e.g. theme names), then use an index number (starting with zero)
# to specify the multiple values.
# The below example is equivalent to:
# themes:
#   - name: 'dspace'
#   - name: 'mytheme'
#     handle: '10673/123'
DSPACE_THEMES_0_NAME = 'dspace'
DSPACE_THEMES_1_NAME = 'mytheme'
DSPACE_THEMES_1_HANDLE = '10673/123'
```

- Or, by creating a `.env` (environment) file in the project root directory and setting the environment variables in that location.

The override priority ordering is as follows (with items listed at the top overriding all other settings)

1. Environment variables
2. The `.env` file
3. The `./config/config.prod.yml`, `./config/config.dev.yml` or `./config/config.test.yml` files (depending on current mode)
4. The `./config/config.yml` file
5. The hardcoded defaults in `./src/config/default-app-config.ts`

In 7.1 or 7.0

The UI configuration files reside in the ./`src/environments/` folder in the Angular UI source code. The default configuration are in `environment.common.ts` in that directory.

To change the default configuration values, you simply create (one or more) local files that override the parameters you need to modify. You can use `environment.template.ts` as a starting point.

- For example, create a new `environment.dev.ts` file in `src/environments/` for a `development` environment;
- For example, create a new `environment.prod.ts` file in `src/environments/` for a `production` environment;

The "ui" and "rest" sections of the configuration may also be overridden separately via one of the following

- By setting any of the following environment variables in your system:

```
# "ui" settings environment variables
DSPACE_HOST # The host name of the angular application
DSPACE_PORT # The port number of the angular application
DSPACE_NAMESPACE # The namespace of the angular application
DSPACE_SSL # Whether the angular application uses SSL [true/false]

# "rest" settings environment variables
DSPACE_REST_HOST # The host name of the REST application
DSPACE_REST_PORT # The port number of the REST application
DSPACE_REST_NAMESPACE # The namespace of the REST application
DSPACE_REST_SSL # Whether the angular REST uses SSL [true/false]
```

- Or, by creating a `.env` (environment) file in the project root directory and setting the environment variables in that location.

The override priority ordering is as follows (with items listed at the top overriding all other settings)

1. Environment variables
2. The ".env" file
3. The "environment.prod.ts", "environment.dev.ts" or "environment.test.ts"
4. The "environment.common.ts"

Configuration Reference

The following configurations are available in `./src/environments/environment.common.ts` These settings may be overridden as described above

Production Mode

Only valid for 7.1 or 7.0

As of 7.2 and above, Angular production mode is automatically enabled whenever you are running the app in Production mode (NODE_ENV=production, or 'yarn start:prod' or 'yarn serve:ssr'). Angular production mode is automatically disabled when you are running the app in Development mode (NODE_ENV=development, or 'yarn start:dev" or 'yarn serve')

When Production mode is enabled, this enables Angular's runtime production mode and compresses the built application. This should always be enabled in Production scenarios.

```
production: true
```

UI Core Settings

The "ui" (user interface) section defines where you want Node.js to run/respond. It may correspond to your primary/public URL, but it also may not (if you are running behind a proxy). In this example, we are setting up our UI to just use localhost, port 4000. This is a common setup for when you want to use Apache or Nginx to handle HTTPS and proxy requests to Node.js running on port 4000.

Format for 7.2 or later (config.*.yml)

```
ui:
  ssl: false
  host: localhost
  port: 4000
  # NOTE: Space is capitalized because 'namespace' is a reserved string in TypeScript
  nameSpace: /
  # The rateLimiter settings limit each IP to a 'max' of 500 requests per 'windowMs' (1 minute).
  rateLimiter:
    windowMs: 60000 # 1 minute
    max: 500 # limit each IP to 500 requests per windowMs
```

Format for 7.1 or 7.0 (environment.*.ts)

```
ui: {
  ssl: false,
  host: 'localhost',
  port: 4000,
  // NOTE: Space is capitalized because 'namespace' is a reserved string in TypeScript
  nameSpace: '/',
  // The rateLimiter settings limit each IP to a "max" of 500 requests per "windowMs" (1 minute).
  rateLimiter: {
    windowMs: 1 * 60 * 1000,   // 1 minute
    max: 500 // limit each IP to 500 requests per windowMs
  }
},
```

The "rateLimiter" sub-section can be used to protect against a DOS (denial of service) attack when the UI is processed on the server side (i.e. server-side rendering). Default settings are usually OK. In Angular, server-side rendering occurs to support better Search Engine Optimization (SEO), as well as to support clients which cannot use Javascript. See also Angular's docs on Server-side rendering.

REST API Settings

The "rest" (REST API) section defines which REST API the UI will use. The REST settings MUST correspond to the primary URL of the backend. Usually, this means they must be kept in sync
with the value of `dspace.server.url` in the backend's `local.cfg`

This example is valid if your Backend is publicly available at https://api.mydspace.edu/server/ . Keep in mind that the "port" must always be specified even if it's a standard port (i.e. port 80 for HTTP and port 443 for HTTPS).

Format for 7.2 or later (config.*.yml)

```
rest:
  ssl: true
  host: api.mydspace.edu
  port: 443
  # NOTE: Space is capitalized because 'namespace' is a reserved string in TypeScript
  nameSpace: /server
```

Format for 7.1 or 7.0 (environment.*.ts)

```
rest: {
  ssl: true,
  host: 'api.mydspace.edu',
  port: 443,
  // NOTE: Space is capitalized because 'namespace' is a reserved string in TypeScript
  nameSpace: '/server'
},
```

Cache Settings - General

The "cache" section controls how long objects/responses will remain in the UI cache. The defaults should be OK for most sites.

Format for 7.2 or later (config.*.yml)

```
cache:
  # NOTE: how long should objects be cached for by default
  msToLive:
    default: 900000 # 15 minutes
  # Default 'Cache-Control' HTTP Header to set for all static content (including compiled *.js files)
  # Defaults to one week. This lets a user's browser know that it can cache these files for one week,
  # after which they will be "stale" and need to be redownloaded.
  control: max-age=604800 # one week
  autoSync:
    defaultTime: 0
    maxBufferSize: 100
    timePerMethod:
      PATCH: 3 # time in seconds
```

Format for 7.1 or 7.0 (environment.*.ts)

```
cache: {
  // NOTE: how long should objects be cached for by default
  msToLive: {
    default: 15 * 60 * 1000, // 15 minutes
  },
  control: 'max-age=60', // revalidate browser
  autoSync: {
    defaultTime: 0,
    maxBufferSize: 100,
    timePerMethod: {[RestRequestMethod.PATCH]: 3} as any // time in seconds
  }
},
```

Cache Settings - Server Side Rendering (SSR)

Available in 7.5 or later

Caching options are also available for the User Interface's "server-side rendering" (which uses Angular Universal). Server-side rendering is used to pre-generate full HTML pages and pass those back to users. This is necessary for Search Engine Optimization (SEO) as some web crawlers cannot use Javascript. It also can be used to immediately show the first HTML page to users while the Javascript app loads in the user's browser.

While server-side-rendering is highly recommended on all sites, it can result in Node.js having to pre-generate many HTML pages at once when a site has a large number of simultaneous users/bots. This may cause Node.js to spend a lot of time processing server-side-rendered content, slowing down the entire site.

Therefore, DSpace provides some basic caching of server-side rendered pages, which allows the same pre-generated HTML to be sent to many users /bots at once & decreases the frequency of server-side rendering.

Two cache options are provide: `botCache` and `anonymousCache`. As the names suggest, the botCache is used for known web crawlers / bots, while the anonymousCache may be used for all anonymous (non-authenticated) users. By default, only the botCache is enabled. But highly active sites may wish to enable the anonymousCache as well, since it can provide users with a more immediate response when they encounter cached pages.

Keep in mind, when the "anonymousCache" is enabled, this means that all non-authenticated users will utilize this cache. This cache can result in massive speed improvements (for initial page load), as the majority of users may be interacting with cached content. However, these users may occasionally encounter cached pages which are outdated or "stale" (based on values of "timeToLive" and "allowStale"). This means that these users will not immediately see new updates or newly added content (Communities, Collections, Items) until the cache has refreshed itself. That said, when "timeToLive" is set to a low value (like 10 seconds), this risk is minimal for highly active pages/content.

config.*.yml

```
cache:
  ...
  serverSide:
    # Set to true to see all cache hits/misses/refreshes in your console logs. Useful for debugging SSR caching
issues.
    debug: false
    # When enabled (i.e. max > 0), known bots will be sent pages from a server side cache specific for bots.
    # (Keep in mind, bot detection cannot be guarranteed. It is possible some bots will bypass this cache.)
    botCache:
      # Maximum number of pages to cache for known bots. Set to zero (0) to disable server side caching for
bots.
      # Default is 1000, which means the 1000 most recently accessed public pages will be cached.
      # As all pages are cached in server memory, increasing this value will increase memory needs.
      # Individual cached pages are usually small (<100KB), so max=1000 should only require ~100MB of memory.
      max: 1000
      # Amount of time after which cached pages are considered stale (in ms). After becoming stale, the cached
      # copy is automatically refreshed on the next request.
      # NOTE: For the bot cache, this setting may impact how quickly search engine bots will index new content
on your site.
      # For example, setting this to one week may mean that search engine bots may not find all new content for
one week.
      timeToLive: 86400000 # 1 day
      # When set to true, after timeToLive expires, the next request will receive the *cached* page & then re-
render the page
      # behind the scenes to update the cache. This ensures users primarily interact with the cache, but may
receive stale pages (older than timeToLive).
      # When set to false, after timeToLive expires, the next request will wait on SSR to complete & receive a
fresh page (which is then saved to cache).
      # This ensures stale pages (older than timeToLive) are never returned from the cache, but some users will
wait on SSR.
      allowStale: true
    # When enabled (i.e. max > 0), all anonymous users will be sent pages from a server side cache.
    # This allows anonymous users to interact more quickly with the site, but also means they may see slightly
    # outdated content (based on timeToLive)
    anonymousCache:
      # Maximum number of pages to cache. Default is zero (0) which means anonymous user cache is disabled.
      # As all pages are cached in server memory, increasing this value will increase memory needs.
      # Individual cached pages are usually small (<100KB), so a value of max=1000 would only require ~100MB of
memory.
      max: 0
      # Amount of time after which cached pages are considered stale (in ms). After becoming stale, the cached
      # copy is automatically refreshed on the next request.
      # NOTE: For the anonymous cache, it is recommended to keep this value low to avoid anonymous users seeing
outdated content.
      timeToLive: 10000 # 10 seconds
      # When set to true, after timeToLive expires, the next request will receive the *cached* page & then re-
render the page
      # behind the scenes to update the cache. This ensures users primarily interact with the cache, but may
receive stale pages (older than timeToLive).
      # When set to false, after timeToLive expires, the next request will wait on SSR to complete & receive a
fresh page (which is then saved to cache).
      # This ensures stale pages (older than timeToLive) are never returned from the cache, but some users will
wait on SSR.
      allowStale: true
```

Authentication Settings

The "auth" section provides some basic authentication-related settings. Currently, it's primarily settings related to when a session timeout warning will be showed to your users, etc.

Format for 7.2 or later (config.*.yml)

```
auth:
  # Authentication UI settings
  ui:
    # the amount of time before the idle warning is shown
    timeUntilIdle: 900000 # 15 minutes
    # the amount of time the user has to react after the idle warning is shown before they are logged out.
    idleGracePeriod: 300000 # 5 minutes
  # Authentication REST settings
  rest:
    # If the rest token expires in less than this amount of time, it will be refreshed automatically.
    # This is independent from the idle warning.
    timeLeftBeforeTokenRefresh: 120000 # 2 minutes
```

Format for 7.1 or 7.0 (environment.*.ts)

```
auth: {
  // Authentication UI settings
  ui: {
    // the amount of time before the idle warning is shown
    timeUntilIdle: 15 * 60 * 1000, // 15 minutes
    // the amount of time the user has to react after the idle warning is shown before they are logged out.
    idleGracePeriod: 5 * 60 * 1000, // 5 minutes
  },
  // Authentication REST settings
  rest: {
    // If the rest token expires in less than this amount of time, it will be refreshed automatically.
    // This is independent from the idle warning.
    timeLeftBeforeTokenRefresh: 2 * 60 * 1000, // 2 minutes
  },
},
```

Form Settings

The "form" section provides basic settings for any forms displayed in the UI. At this time, these settings only include a validatorMap, which is not necessary to modify for most sites

Format for 7.2 or later (config.*.yml)

```
form:
  # (7.5 and above) Whether to enable "spellcheck" attribute of textareas in forms.
  spellCheck: true
  # NOTE: Map server-side validators to comparative Angular form validators
  validatorMap:
    required: required
    regex: pattern
```

Format for 7.1 or 7.0 (environment.*.ts)

```
form: {
  // NOTE: Map server-side validators to comparative Angular form validators
  validatorMap: {
    required: 'required',
    regex: 'pattern'
  }
},
```

Notification Settings

The "notifications" section provides options related to where user notifications will appear in your UI. By default, they appear in the top right corner, and timeout after 5 seconds.

Format for 7.2 or later (config.*.yml)

```
notifications:
  rtl: false
  position:
    - top
    - right
  maxStack: 8
  # NOTE: after how many seconds notification is closed automatically. If set to zero notifications are not
closed automatically
  timeOut: 5000 # 5 second
  clickToClose: true
  # NOTE: 'fade' | 'fromTop' | 'fromRight' | 'fromBottom' | 'fromLeft' | 'rotate' | 'scale'
  animate: scale
```

Format for 7.1 or 7.0 (environment.*.ts)

```
notifications: {
  rtl: false,
  position: ['top', 'right'],
  maxStack: 8,
  // NOTE: after how many seconds notification is closed automatically. If set to zero notifications are not
closed automatically
  timeOut: 5000, // 5 second
  clickToClose: true,
  // NOTE: 'fade' | 'fromTop' | 'fromRight' | 'fromBottom' | 'fromLeft' | 'rotate' | 'scale'
  animate: NotificationAnimationsType.Scale
},
```

The set of valid animations can be found in the NotificationAnimationsType, and are implemented in `./src/shared/animations/`

Submission Settings

The "submission" section provides some basic Submission/Deposit UI options. These allow you to optionally enable an autosave (disabled by default), and custom styles/icons for metadata fields or authority confidence values.

Format for 7.2 or later (config.*.yml)

```
submission:
  autosave:
    # NOTE: which metadata trigger an autosave
    metadata: []
    # NOTE: after how many time (milliseconds) submission is saved automatically
    # eg. timer: 300000 # 5 minutes
    timer: 0
  icons:
    metadata:
      # NOTE: example of configuration
      #   # NOTE: metadata name
      # - name: dc.author
      #   # NOTE: fontawesome (v5.x) icon classes and bootstrap utility classes can be used
      #   style: fas fa-user
      - name: dc.author
        style: fas fa-user
      # default configuration
      - name: default
        style: ''
    authority:
      confidence:
        # NOTE: example of configuration
        #   # NOTE: confidence value
        # - name: dc.author
        #   # NOTE: fontawesome (v5.x) icon classes and bootstrap utility classes can be used
        #   style: fa-user
        - value: 600
          style: text-success
        - value: 500
          style: text-info
        - value: 400
          style: text-warning
        # default configuration
        - value: default
          style: text-muted
```

Format for 7.1 or 7.0 (environment.*.ts)

```
submission: {
    autosave: {
      // NOTE: which metadata trigger an autosave
      metadata: [],
      /**
       * NOTE: after how many time (milliseconds) submission is saved automatically
       * eg. timer: 5 * (1000 * 60); // 5 minutes
       */
      timer: 0
    },
    icons: {
      metadata: [
        /**
         * NOTE: example of configuration
         * {
         *    // NOTE: metadata name
         *    name: 'dc.author',
         *    // NOTE: fontawesome (v5.x) icon classes and bootstrap utility classes can be used
         *    style: 'fa-user'
         * }
         */
        {
          name: 'dc.author',
          style: 'fas fa-user'
        },
        // default configuration
        {
          name: 'default',
          style: ''
        }
      ],
      authority: {
        confidence: [
          /**
           * NOTE: example of configuration
           * {
           *    // NOTE: confidence value
           *    value: 'dc.author',
           *    // NOTE: fontawesome (v4.x) icon classes and bootstrap utility classes can be used
           *    style: 'fa-user'
           * }
           */
          {
            value: 600,
            style: 'text-success'
          },
          {
            value: 500,
            style: 'text-info'
          },
          {
            value: 400,
            style: 'text-warning'
          },
          // default configuration
          {
            value: 'default',
            style: 'text-muted'
          },

        ]
      }
    }
  },
```

Language Settings

The "defaultLanguage" and "languages" sections allow you to customize which languages to support in your User Interface. See also Multilingual Support.

Format for 7.2 or later (config.*.yml)

```
#  Default Language in which the UI will be rendered if the user's browser language is not an active language
defaultLanguage: en

# Languages. DSpace Angular holds a message catalog for each of the following languages.
# When set to active, users will be able to switch to the use of this language in the user interface.
# All out of the box language packs may be found in the ./src/assets/i18n/ directory
languages:
  - code: en
    label: English
    active: true
  - code: cs
    label: eština
    active: true
  - code: de
    label: Deutsch
    active: true
  - ...
```

Format for 7.1 or 7.0 (environment.*.ts)

```
// Default Language in which the UI will be rendered if the user's browser language is not an active language
defaultLanguage: 'en',
// Languages. DSpace Angular holds a message catalog for each of the following languages.
// When set to active, users will be able to switch to the use of this language in the user interface.
languages: [{
  code: 'en',
  label: 'English',
  active: true,
}, {
  code: 'de',
  label: 'Deutsch',
  active: true,
},
...
],
```

The DSpace UI requires that a corresponding language pack file (named with the language code and ending in ".json5") be placed in `./src/assets/i18n/`. See also DSpace 7 Translation - Internationalization (i18n) - Localization (l10n) for information about how to create and contribute these files.

Browse By Settings

In 7.2 or above, the "browseBy" section only provides basic UI configurations for "Browse by" pages (/browse path). The "Browse by" options that appear in the "All of DSpace" header menu *are determined dynamically from the REST API*. This allows the UI to change dynamically based on the configured browse indexes in Discovery.

Format for 7.2 or later (config.*.yml)

```
browseBy:
  # Amount of years to display using jumps of one year (current year - oneYearLimit)
  oneYearLimit: 10
  # Limit for years to display using jumps of five years (current year - fiveYearLimit)
  fiveYearLimit: 30
  # The absolute lowest year to display in the dropdown (only used when no lowest date can be found for all
items)
  defaultLowerLimit: 1900
  # If true, thumbnail images for items will be added to BOTH search and browse result lists. (default: true)
  showThumbnails: true
  # The number of entries in a paginated browse results list.
  # Rounded to the nearest size in the list of selectable sizes on the settings menu.
  pageSize: 20

# NOTE: The "types" section no longer exists, as it is determined dynamically via the REST API
```

NOTE: The "pageSize" configuration will always round to the closest "pageSizeOptions" value listed in "page-component-options.model.ts"

In 7.1 or 7.0, the "browseBy" section allowed you to customize which "Browse by" options appear in the "All of DSpace" header menu at the top of your DSpace site. The "id" MUST correspond to the *name* of a valid Browse index available from your REST API (see documentation on the REST API /api /discover/browses endpoint). It is possible to configure additional indexes on the Backend using Discovery, and any configured index appears in your REST API.

Format for 7.1 or 7.0 (environment.*.ts)

```
browseBy: {
  // Amount of years to display using jumps of one year (current year - oneYearLimit)
  oneYearLimit: 10,
  // Limit for years to display using jumps of five years (current year - fiveYearLimit)
  fiveYearLimit: 30,
  // The absolute lowest year to display in the dropdown (only used when no lowest date can be found for all
items)
  defaultLowerLimit: 1900,
  // List of all the active Browse-By types
  // Adding a type will activate their Browse-By page and add them to the global navigation menu,
  // as well as community and collection pages
  // Allowed fields and their purpose:
  //    id:             The browse id to use for fetching info from the rest api
  //    type:           The type of Browse-By page to display
  //    metadataField:  The metadata-field used to create starts-with options (only necessary when the type is
set to 'date')
  types: [
    {
      id: 'title',
      type: BrowseByType.Title,
    },
    {
      id: 'dateissued',
      type: BrowseByType.Date,
      metadataField: 'dc.date.issued'
    },
    {
      id: 'author',
      type: BrowseByType.Metadata
    },
    {
      id: 'subject',
      type: BrowseByType.Metadata
    }
  ]
},
```

Community-List Settings

Available in 7.4 or later

The "communityList" section allows you to configure the behavior of the "Communities & Collections" page (/community-list path), which is linked in the header.

config.*.yml

```
communityList:
  # Number of communities to list per expansion (i.e. each time you click "show more")
  pageSize: 20
```

NOTE: The "pageSize" configuration will always round to the closest "pageSizeOptions" value listed in "page-component-options.model.ts"

Homepage Settings

Available in 7.4 or later

The "homePage" section allows you to configure the behavior of the DSpace homepage (/ path).

config.*.yml

```
homePage:
  recentSubmissions:
    # The number of item showing in recent submissions list. Set to "0" to hide all recent submissions
    pageSize: 5
    # Date field to use to sort recent submissions
    sortField: 'dc.date.accessioned'
  topLevelCommunityList:
    # Number of communities to list (per page) on the home page
    # This will always round to the nearest number from the list of page sizes. e.g. if you set it to 7 it'll
use 10
    pageSize: 5
```

NOTE: The "pageSize" configuration will always round to the closest "pageSizeOptions" value listed in "page-component-options.model.ts"

Undo Settings

Both the "item" edit and "collection" edit screens allow you to undo changes within a specific time. This is controlled by these settings:

Format for 7.2 or later (config.*.yml)

```
item:
  edit:
    undoTimeout: 10000 # 10 seconds

collection:
  edit:
    undoTimeout: 10000 # 10 seconds
```

Format for 7.1 or 7.0 (environment.*.ts)

```
item: {
  edit: {
    undoTimeout: 10000 // 10 seconds
  }
},
collection: {
  edit: {
    undoTimeout: 10000 // 10 seconds
  }
},
```

Item Access Labels

Available in 7.3 or later

Item access labels allow to display for each item in search results if it is Open Access, under embargo, restricted or metadata only (does not contain any file/bitstream). This feature is disabled by default, but can be enabled in your config.*.yml.

config.*.yml

```
# Item Config
item:
  # Show the item access status label in items lists (default=false)
  showAccessStatuses: true
```

Item Page Settings

Available in 7.5 or later

The "item" section allows you to configure the behavior of the Item pages.

config.*.yml

```
item:
  ...
  bitstream:
    # Number of entries in the bitstream list in the item view page.
    pageSize: 5
```

NOTE: The "pageSize" configuration will always round to the closest "pageSizeOptions" value listed in "page-component-options.model.ts"

Theme Settings

The "themes" section allows you to configure which theme(s) are enabled for your DSpace site (with the default theme being the "dspace" one). You can enable a single theme across all pages, and/or enable specific alternative themes based on a specific Community, Collection or Item (by UUID or Handle), or based on a Regex match of a URL pattern. This allows you fine grained control over how your site looks, including the ability to customize it per Community or Collection or even per specific pages in the site. *See User Interface Customization for details of how to create a new, custom theme.*

Format for 7.2 or later (config.*.yml)

```
themes:
  # Add additional themes here. In the case where multiple themes match a route, the first one
  # in this list will get priority. It is advisable to always have a theme that matches
  # every route as the last one
  #
  # # A theme with a handle property will match the community, collection or item with the given
  # # handle, and all collections and/or items within it
  # - name: 'custom',
  #   handle: '10673/1233'
  #
  # # A theme with a regex property will match the route using a regular expression. If it
  # # matches the route for a community or collection it will also apply to all collections
  # # and/or items within it
  # - name: 'custom',
  #   regex: 'collections\/e8043bc2.*'
  #
  # # A theme with a uuid property will match the community, collection or item with the given
  # # ID, and all collections and/or items within it
  # - name: 'custom',
  #   uuid: '0958c910-2037-42a9-81c7-dca80e3892b4'
  #
  # # The extends property specifies an ancestor theme (by name). Whenever a themed component is not found
  # # in the current theme, its ancestor theme(s) will be checked recursively before falling back to default.
  # - name: 'custom-A',
  #   extends: 'custom-B',
  #   # Any of the matching properties above can be used
  #   handle: '10673/34'
  #
  # - name: 'custom-B',
  #   extends: 'custom',
  #   handle: '10673/12'
  #
  # # A theme with only a name will match every route
  # name: 'custom'
  #
  # # This theme will use the default bootstrap styling for DSpace components
  # - name: BASE_THEME_NAME
  #
  - name: dspace
    # Whenever this theme is active, the following tags will be injected into the <head> of the page.
    # Example use case: set the favicon based on the active theme.
    headTags:
    - tagName: link
      attributes:
        rel: icon
        href: assets/dspace/images/favicons/favicon.ico
        sizes: any
    - tagName: link
      attributes:
        rel: icon
        href: assets/dspace/images/favicons/favicon.svg
        type: image/svg+xml
    - tagName: link
      attributes:
        rel: apple-touch-icon
        href: assets/dspace/images/favicons/apple-touch-icon.png
    - tagName: link
      attributes:
        rel: manifest
        href: assets/dspace/images/favicons/manifest.webmanifest
```

Format for 7.1 or 7.0 (environment.*.ts)

```
themes: [
  // Add additional themes here. In the case where multiple themes match a route, the first one
  // in this list will get priority. It is advisable to always have a theme that matches
  // every route as the last one

  // {
  //   // A theme with a handle property will match the community, collection or item with the given
  //   // handle, and all collections and/or items within it
  //   name: 'custom',
  //   handle: '10673/1233'
  // },
  // {
  //   // A theme with a regex property will match the route using a regular expression. If it
  //   // matches the route for a community or collection it will also apply to all collections
  //   // and/or items within it
  //   name: 'custom',
  //   regex: 'collections\/e8043bc2.*'
  // },
  // {
  //   // A theme with a uuid property will match the community, collection or item with the given
  //   // ID, and all collections and/or items within it
  //   name: 'custom',
  //   uuid: '0958c910-2037-42a9-81c7-dca80e3892b4'
  // },
  // {
  //    // Using the "extends" property allows a theme to extend/borrow from an ancestor theme (by name).
  //    // Wherever a theme component is now found in this themes, its ancestor theme(s) will be checked
  //    // recursively before falling back to default.
  //    name: 'custom-A',
  //    extends: 'custom-B',
  //    // Any of the matching properties above can be used
  //    handle: 10673/34,
  // },
  // {
  //    name: 'custom-B',
  //    extends: 'custom',
  //    handle: 10673/12,
  // },
  // {
  //   // A theme with only a name will match every route
  //   name: 'custom'
  // },
  // {
  //   // This theme will use the default bootstrap styling for DSpace components
  //   name: BASE_THEME_NAME
  // },
  {
    // The default dspace theme
    name: 'dspace'
    // Whenever this theme is active, the following tags will be injected into the <head> of the page.
    // Example use case: set the favicon based on the active theme.
    headTags: [
      {
        // Insert <link rel="icon" href="assets/dspace/images/favicons/favicon.ico" sizes="any"/> into the
<head> of the page.
        tagName: 'link',
        attributes: {
          'rel': 'icon',
          'href': 'assets/dspace/images/favicons/favicon.ico',
          'sizes': 'any',
        }
      },
      ...
    ]
  },
],
```

Media Viewer Settings

The DSpace UI comes with a basic, out-of-the-box Media Viewer (disabled by default). This media viewer can support any files which have a MIME Type that *begins with* either "image/*", "video/*", or "audio/*".

Format for 7.2 or later (config.*.yml)

```
# Whether to enable media viewer for image and/or video Bitstreams (i.e. Bitstreams whose MIME type starts with
'image' or 'video').
# When "image: true", this enables a gallery viewer where you can zoom or page through images.
# When "video: true", this enables embedded video streaming.  This embedded video streamer also supports audio
files.
mediaViewer:
  image: false
  video: false
```

Uploading video captioning files

As of 7.5 (or later), the Video viewer also supports WebVTT (or VTT) Captioning. Video captioning requires that a WebVTT Caption file (.vtt) be uploaded into the DSpace Item (DSpace is not able to create or generate these .vtt files). Here's an example of how to setup captioning:

- The Item must already have a Bitstream which is a video file (in a "video/*" format) in the ORIGINAL bundle. In this example, we'll assume it is named "myVideo.mp4"
- Upload a corresponding WebVTT Caption file named "`[video-filename]-[languageCode].vtt`" to the ORIGINAL bundle.
    - For a video named "myVideo.mp4", an English caption file would be named "myVideo.mp4-en.vtt".
    - If an additional Spanish language caption file was uploaded, it should be named "myVideo.mp4-es.vtt".
    - All WebVTT Caption files MUST use two-letter ISO 639-1 Language Codes. A list of all supported Language Codes can be found in "src/app/item-page/media-viewer/media-viewer-video/language-helper.ts"
- Once the Caption file is uploaded, reload the video viewer (on the Item page). You should now see the "Captions" (or CC) option is now available. (Depending on the browser you use, this option may appear in the lower menu of the video, or require you to open an options menu.) Selecting it will enable captioning in your language of choice.

Format for 7.1 or 7.0 (environment.*.ts)

```
// Whether to enable media viewer for image and/or video Bitstreams (i.e. Bitstreams whose MIME type starts
with "image" or "video").
// For images, this enables a gallery viewer where you can zoom or page through images.
// For videos, this enables embedded video streaming
mediaViewer: {
  image: false,
  video: false,
},
```

Toggle end-user agreement and privacy policy

Available in 7.4 or later

The DSpace UI comes with basic end-user agreement and privacy policy functionality. Since release 7.4 these features can be disabled in a configuration file. More information on what disabling on of these features results in is documented in the default app configuration (see code snippet below).

config.*.yml

```
info:
  # Whether the end user agreement is required before users may use the repository.
  # If enabled, the user will be required to accept the agreement before they can use the repository.
  # If disabled, the page will not exist and no agreement is required to use the repository
  enableEndUserAgreement: false
  # Whether the privacy statement should exist or not.
  enablePrivacyStatement: false
```

By default the features are enabled.

Settings for rendering Markdown, HTML and MathJax in metadata

Available in 7.4 or later

The DSpace UI can support Markdown (using https://commonmark.org/) and MathJax (https://www.mathjax.org) in metadata field values. Both Markdown and MathJax are disabled by default.

HTML is a part of markdown, so enabling the markdown option will ensure HTML tags in metadata field values get rendered as well

```
# Whether to enable Markdown (https://commonmark.org/) and MathJax (https://www.mathjax.org/)
# display in supported metadata fields. By default, only dc.description.abstract is supported.
markdown:
  enabled: false
  mathjax: false
```

Mathjax will only be rendered if markdown is enabled, so configuring 'markdown.mathjax = true' with 'markdown.enabled = false' will have no effect.

By default, only the "dc.description.abstract" metadata supports these formats when enabled. To enable markdown for other metadata fields, a custom sub-component of the ItemPageFieldComponent has to be created for that metadata field, with the enableMarkdown field set to true. Refer to the ItemPageAbstractFieldComponent component for an example.

Controlled Vocabularies in Search Filters

Available in 7.5 or later

When using hierarchical controlled vocabularies (e.g. SRSC as described in Authority Control of Metadata Values), it's possible to search using the controlled vocabulary hierarchy via the search filters. To enable this feature, you must specify the filter and vocabulary to enable as follows:

```
# Which vocabularies should be used for which search filters
# and whether to show the filter in the search sidebar
# Take a look at the filter-vocabulary-config.ts file for documentation on how the options are obtained
vocabularies:
  - filter: 'subject'
    vocabulary: 'srsc'
    enabled: true
```

Keep in mind, the "filter" MUST be a valid search filter (e.g. subject, author) as seen on the "/api/discover/facets" REST API endpoint. The "vocabulary" MUST be a valid controlled vocabulary installed in your DSpace backend (under "[dspace]/config/controlled-vocab/" folder based on the documentation at Authority Control of Metadata Values.

When this feature is enabled, you should see a "Browse [filter] tree" link in the search filter on the search results page (and anywhere search filters are shown). This "Browse [filter] tree" link will allow you to select a search filter from within the configured hierarchical vocabulary.

Universal (Server-side Rendering) Settings

Only valid for 7.1 or 7.0

⚠ As of DSpace 7.2, these settings are no longer editable. Universal is automatically enabled at all times to support Search Engine Optimization.

The "universal" section pertains to enabling/disabling Angular Universal for Server-side rendering. DSpace requires server-side rendering to support Search Engine Optimization. When it's turned off, your site may not be able to be indexed in Google, Google Scholar and other search engines.

environment.*.ts

```
// Angular Universal settings
universal: {
  preboot: true,
  async: true,
  time: false
},
```

Debug Settings

The "debug" property allows you to turn on debugging in the Angular UI. When enabled, your environment and all Redux actions/transfers are logged to the console. This is only ever needed if you are debugging a tricky issue.

Format for 7.2 or later (config.*.yml)

```
# NOTE: will log all redux actions and transfers in console
debug: false
```

Format for 7.1 or 7.0 (environment.*.ts)

```
// NOTE: will log all redux actions and transfers in console
debug: false
```