Configuration file changes from 2.x to 3.x (up to 3.1.1)

Last major update: March 2nd, 2022

Last update making note of **coming** 3.1.2. changes: June 7, 2024

# GSRS configuration has significantly changed

GSRS 3 is written in an entirely different framework than the previous versions of GSRS so the way to specify the configuration has also had to change. While GSRS 3 REST API tries to be backwards compatible with the GSRS 2 REST API so that API consumers don’t have to change, it was felt a major version update can change how to install and configure a GSRS instance is done especially since GSRS installations only make minor configuration changes.

Like other Spring Boot projects, configuration files should be put in the src/main/resources directory. Please note that this is different than the GSRS 2 directory structure which was modules/ginas/conf.

# GSRS 3 still uses HOCON format

Like previous versions of GSRS, the GSRS 3 configuration uses HOCON (Human-Optimized Config Object Notation) format. It is possible to use Spring properties or yml encoded files instead, but it is not recommended. HOCON format allows inheriting configuration settings so by using the GSRS HOCON configuration, the default settings for GSRS are automatically included. **GSRS 3 installations only need to set custom configurations to override settings and provide database configuration information**

Future: GSRS 3.1.2 will make big changes versus GSRS 3.0-3.1.1  
  
The scope of this document is to compare 2x to 3x. The main points discussed in this document remain pertinent even after 3.1.2. In this section, we just make of note of **projected** for 3.1.2 changes including:

* Extension of HOCON to additional microservices like the Gateway and Frontend
* Standardized 'application.conf' files in each microservice with placeholder defaults and optional overrides via environment variables.
* A move away from “lists” and toward “maps” in the configuration of extensions. Maps will make it easier to override configuration values by referencing a key.

When 3.1.2 is released, a document to understand 3.1.2 changes **will be** located here: <https://github.com/ncats/gsrs-ci/tree/gsrs3-main-deployment/docs/how-configuration-works.md>

# How to turn on HOCON format

Since Spring Boot doesn’t come built in with HOCON support, GSRS 3 relies on a 3rd party open source library to add HOCON support <https://github.com/zeldigas/spring-hocon-property-source> . HOCON support is on by default in the GSRS source repositories. However, in a new fork or supporting library it may not be configured to allow HOCON by default. If that’s the case, please follow the documentation of that library for how to tell Spring to read HOCON format. As of this writing, this usually involves adding a line to a file in your project (and/or creating the file).

File: META-INF/spring.factores

org.springframework.boot.env.PropertySourceLoader=com.github.zeldigas.spring.env.HoconPropertySourceLoader

This will make Spring Boot look for a file named `application.conf` as the configuration file. GSRS 3 has several built in conf files that are already on the classpath for common settings.

For Substances, the file substances-core.conf will contain the default substance configuration settings. To use this configuration file as the base configuration file please create a file called application.conf and as its first line put include “substances-core.conf”. This is done as an example in the gsrs3-main-deployment git repository.

# Potential problems migrating a HOCON configuration file from GSRS 2 to GSRS 3.

By using HOCON format in a Spring Boot application we get all the benefits of leveraging HOCON such as:

* less noisy / less pedantic syntax
* ability to refer to another part of the configuration (set a value to another value)
* import/include another configuration file into the current file
* a mapping to a flat properties list such as Java's system properties
* ability to get values from environment variables
* ability to write comments

We also leverage the benefits of Spring Boot including injecting property values into Beans and setting their values using Spring Expression Language <https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#expressions>

However, Spring Boot property configuration is handled internally a bit differently than the property configuration used in older versions of GSRS which used the Play! Framework. For example, if you had previously attempted to instantiate a custom Java List or Java Set by using a JSON array in the config file, this will no longer typically work. Instead, most of these properties will be interpreted not as Java List or Java Sets at time of deserialization into your object, but instead as LinkedHashMap<Integer,?>. You must therefore either convert all internal initialization / property setting mechanisms to use this data type instead of Lists/Sets OR you must add custom deserialization steps to convert from LinkedHashMap<Integer,?> to your underlying list/set. This is particularly important for cases where customized extensions are used for GSRS. This backend parsing change has been addressed for specific cases where old features of GSRS 2.X are ported to 3.0, but for custom java code, particularly supporting initialization parameters from the config file, care should be taken to ensure that deserialization is complete and works as expected in 3.0.

# Specific changes to Config File Properties

The GSRS 3.0 config file mostly works as it did in 2.X, however, if you are instantiating a model for a customized extension (e.g. IndexValueMaker, ValidationRule, RegisteredFunction, EntityProcessor, ScheduledTask or Exporters), and are using the config file to instantiate your properties in the extension object there are some significant changes. Some of these changes to look out for are exemplified below:

|  |  |  |
| --- | --- | --- |
| **Configuration Feature** | **Purpose** | **Change Notes** |
| Entity Processors (java triggers) | Setting a list of backend triggers to run based on data operations. | 2.X Path: ix.core.entityprocessors  3.X Path: gsrs.entityProcessors  Other Notes: Property called "class" is now called "entityClassName" |
| Validation Rules | Setting a list of validation rules to run on new entries and updates from the REST API. | 2.X Path: substance.validators  3.X Path: gsrs.validators.substances  Other Notes: Replace "substances" with whatever entity is intended. It will use the resource name based on the controller. |
| Index Value Makers | Setting a list of custom fields to be generated and indexed for text search, facets, sort order, etc. | 2.X Path: ix.core.indexValueMakers  3.X Path: gsrs.indexers.list  Other Notes: Property "class" to specify what entity this applies to is not typically necessary anymore. The IVM interface itself should specify which object it applies to. |
| Export Format Support | Setting a list of exporter factories for making certain kinds of data exports (e.g. custom excel, CSV, json, etc.) | 2.X Path: ix.ginas.exportFactories  3.X Path: ix.ginas.export.factories.substances  Other Notes: Replace "substances" with whatever entity is intended. It will use the resource name based on the controller. |
| Hierarchy Recipes | Setting a list of substance-specific hierarchy creation recipes to show which substance is effectively a "more specific" form of another substance. | 2.X Path: substance.hierarchyFinders  3.X Path: substance.hierarchyFinders.recipes |
| Deep Field Indexing | Turns on deep field support to allow text indexing and searching to suggest "did you mean" search suggestions for modifying a query. | 2.X Path: ix.index.deepfields  3.X Path: ix.index.deepfieldsraw  Other Notes: Instead of a JSON list this is now a single text string with ";" delimitating each fully qualified entity class intended to have deep field support |
| Export Directory | Directory that the export files will be stored on the server for convenient download for users. | 2.X Path: export.path.root  3.X Path: ix.ginas.export.path |
| Database Configuration settings | Connect to database for ORM. | 2.X Path: db.default  3.X Path: spring.datasource  Other Notes: The actual syntax needed to establish a database connection is very different and is explained in another document. |

The above list is not exhaustive, but it shows the main areas that may need to be adjusted for common config file migrations. In addition, there are some configuration features that were supported in 2.X which are no longer supported. The default values found in the gsrs-core.conf and substances-core.conf config file in the starter libraries provide a simple overview of some of these configuration parameters and how they can be changed.

# A note on using “.” within property names

While GSRS 3.0 supports HOCON, one specific way HOCON differs in the 3.0 codebase is the support for the period (“.”) character as part of a property name in a configuration object.

For example, the following configuration element in GSRS 2.X was sometimes used:

|  |
| --- |
| ix.core.initializers+={  "class":"ix.ginas.initializers.SQLReportScheduledTaskInitializer",  "output.path":"reports/nameReport-%DATE%.txt"  } |

However, this same syntax in GSRS 3.0 would be treated as equivalent to:

|  |
| --- |
| ix.core.initializers+={  "class":"ix.ginas.initializers.SQLReportScheduledTaskInitializer",  "output":{  "path":"reports/nameReport-%DATE%.txt"  }  } |

This change in parsing behavior changes some deserialization procedures. As a result, cases like “output.path” which may have been used in 2.X configuration settings are more commonly expected as “ouputPath” in the 3.0 codebase.