

GSRS 3.1.2 Release Notes

July 2025

Table Contents

[Overview 2](#_Toc1740868549)

[Highlighted New Features and Improvements 3](#_Toc451549136)

[Spring Boot upgrade 3](#_Toc1902912858)

[HOCON config changes 3](#_Toc1076520591)

[Cross entity search 5](#_Toc947101185)

[Structure search improvements 9](#_Toc411719639)

[Impurities Improvement: 9](#_Toc1051239474)

[Other selected Improvements and bug fixes: 17](#_Toc479023627)

[Improvement: A new Exporter: SQL Exporter 17](#_Toc956116043)

[Improvement: ScheduledExportTask 18](#_Toc821679227)

[Improvement: Two InChIKeys for certain chemicals 21](#_Toc1885404405)

[Improvement: Show changes between two adjacent versions a substance 21](#_Toc95142425)

[Improvement: PDF download option 23](#_Toc1876070369)

[Improvement: JmesPathValidator allows you to create new business rules easily 24](#_Toc1010521527)

[Improvement: GsrsApiExporterFactory allows you to transfer data between instances of GSRS in real time 25](#_Toc1265139978)

[Improvement: SQLExporterFactory allows you to write the results of a SQL query to a file 26](#_Toc1164332490)

[Improvement: ScheduledExportTask automatically runs SQL and sends the results to configured destinations. 27](#_Toc2084707665)

[Improvement: JmespathSpreadsheetExporterFactory makes it easy to create a custom spreadsheet exporter 29](#_Toc655925560)

[Bug fix: Pasting certain SMILES into a GSRS structure box resulted in an improbably structure 31](#_Toc153138520)

[Bug fix: User Saved List 33](#_Toc466402474)

[Improvement: Update fields definition in Impurities tables 33](#_Toc292841959)

[33](#_Toc985858365)

[Notes 33](#_Toc353502861)

[JSDraw License and Configuration to Disable JSDraw 33](#_Toc604404920)

# Overview

The **GSRS 3.1.2** software release is a minor release. Our primary focus is on upgrading Spring Boot and updating the HOCON configuration. At the same time, we have introduced **cross-entity search**, which allows users to retrieve related records from other entity services based on the current search results in a given entity service. **Exact Plus** structure search has been added as a salt-stripped version of the existing Exact structure search. A **simplified registration form** for impurities is now supported. Other improvements and bug fixes are also included in this version; selected highlights are listed below.

**Highlighted new features include:**

* Spring Boot upgrade
* HOCON config changes
* Cross-entity search
* Exact plus structure search

**Highlighted improvements and bug fixes include:**

* Impurities improvements
* SQL exporter
* Two InChIKeys for certain chemicals
* Show changes between two adjacent edits

# Highlighted New Features and Improvements

## Spring Boot upgrade

**Purpose and Motivation:**

Older versions of Spring Boot are often linked to numerous security vulnerabilities. Newer releases address these issues by patching known security flaws and updating embedded libraries with additional security fixes. Moreover, using outdated versions can make debugging more challenging, as most online resources and documentation focus on the latest Spring Boot 3.x versions. This is an effort to keep the framework up to date.  
  
**How it works:**

We updated the Spring Boot version from 2.4.5 to 2.7.18 in this release. No action is needed from the user for this upgrade.

**Notes for the future:**   
We will continue to update Spring Boot to version 3.x.

## HOCON config changes

**Purpose and Motivation:**

GSRS provides highly flexible services that different organizations can customize for their own use. Besides the extendable software design and implementation, it is all made possible through configuration. The configuration system prior to version 3.1.2 had its limitations. With the previous setup, it was difficult to modify configuration in deployed systems. In some cases, we had to rebuild JAR files just to apply a configuration change. GSRS has used HOCON (Human-Optimized Config Object Notation) configuration format since version 2.x, In version 3.1.2, we have taken full advantage its features in all GSRS 3.x microservices. With the new configuration strategy and expanded use HOCON, we can override settings using environment variables, and file includes. This makes it possible to update configurations without rebuilding the JARs. It also makes it easier to deploy GSRS with Docker.

The benefit of HOCON is that it:

* provides a more readable format;
* allows for easier environment variable substitutions;
* allows for more flexible overrides; and
* allows for more flexible file includes.

We acknowledge that there is a learning curve when switching to HOCON, but once you get used to it, you'll find it well worth the effort.

**How it works:**

In version 3.1.2, we expanded the use of HOCON in configuration, and we made major changes in how configuration works.

* All microservices now use HOCON. Previously, the gateway and frontend services did not.
* Validators, indexers, processors and other extensions are now configured as map-lists rather than in numerically indexed arrays.
* We dramatically increased support for the passing environment variables that are subsequently interpolated in the application.conf files.
* We added major support for full and partial overrides of configuration values.

These are BIG but manageable changes, much more detail on these changes and steps for migration is available here:   
https://github.com/ncats/gsrs3-main-deployment/blob/main/docs/how-configuration-works-3.1.2.md

**Notes for the Future:**

We will continue working on improving how configuration works to make it more user-friendly and easier to maintain. One area we plan to improve next is non-entity services.

## Cross entity search

**Purpose and Motivation:**

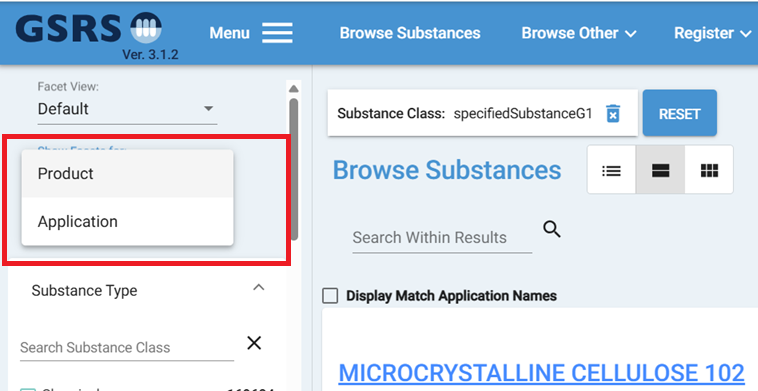
Searching across different entities—such as Substance, Product, and Application—has been a longstanding challenge in GSRS. In GSRS 3.1.2, a new feature called *Cross-Entity Search* has been introduced to address this issue. This feature allows users to search and filter data across multiple entities and view the desired results directly in the browser.

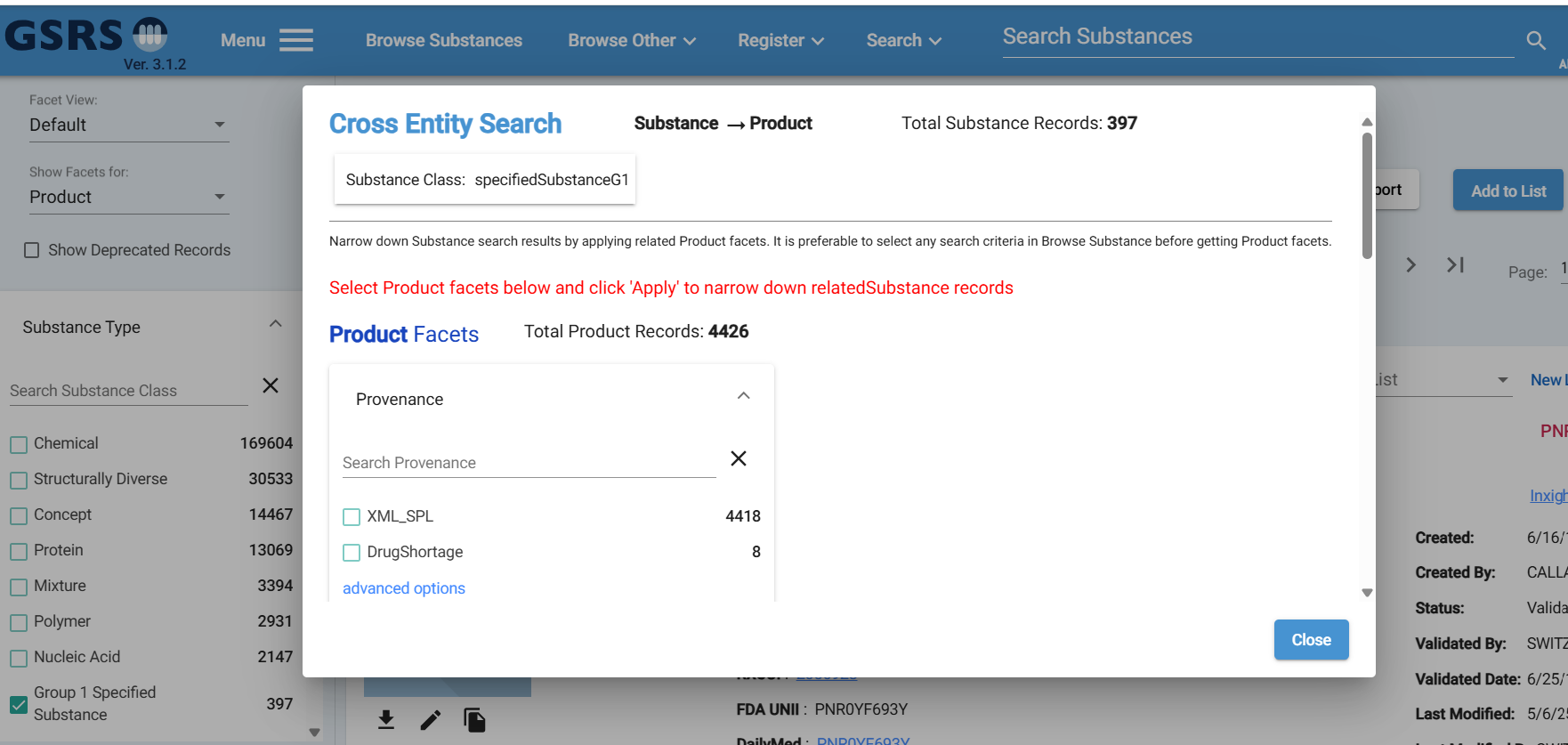
**How it works:**

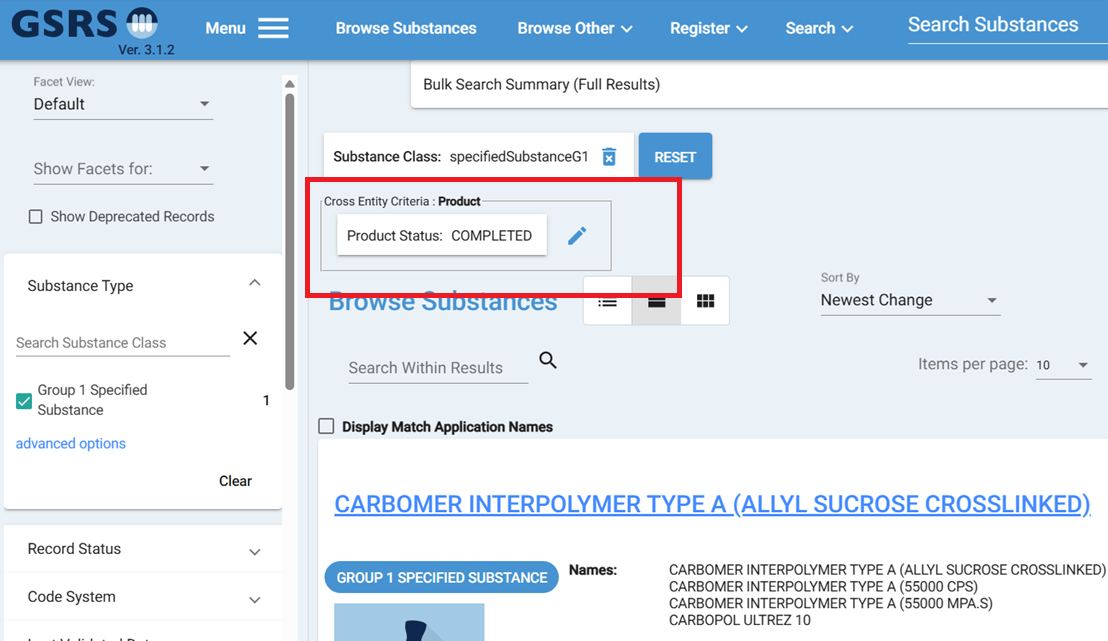
In GSRS 3.1.2, a new feature Cross-Entity Search allows user to take current search results in the browser and select facets of the other selected sub-entity and return results. In Browse Substance page, a new dropdown “Show Facet For”, is implemented where “Product” and “Application” values are displayed. See *Figure 1* below. As well, In Browse Application and Product pages, in dropdown “Show Facet For”, a “Substance” value is displayed.

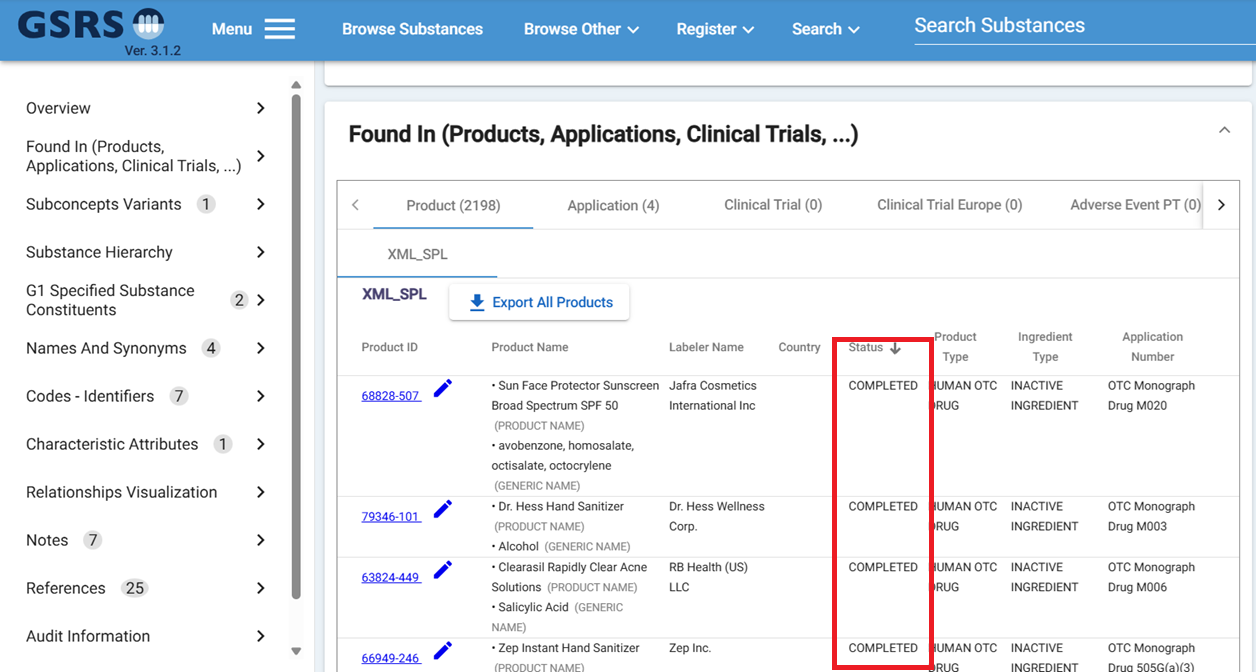
For Example, In Browse Substance page, user selects facet “Substance Class” as value “Specified Substance Group 1” and performs the search. The search results display all the Substances where Substance Class is Specified Substance Group 1. Then user selects “Product” value from dropdown “Show Facets For”. After that a popup will appear where user can display the Product facets for all the Substance UUIDs for Specified Substance Group 1. See *Figure 2* below. The user selects the desired facets such as “Product Status” is “Completed” and click “Apply” button on the facet. In the backend it performs a bulk search and sends back the results to Browse Substance page where Substance Class is Specified Substance Group 1 and Product Status is completed. See *Figure 3* below. To verify, go to Substance Details page, and click on ‘Product’ tab, and see that the Substance has Product records where Status is Completed. See *Figure 4* below.

Same search strategy applies in Browse Application and Product pages. This feature provides powerful search between entities to narrow down the data for research and critical decision making.

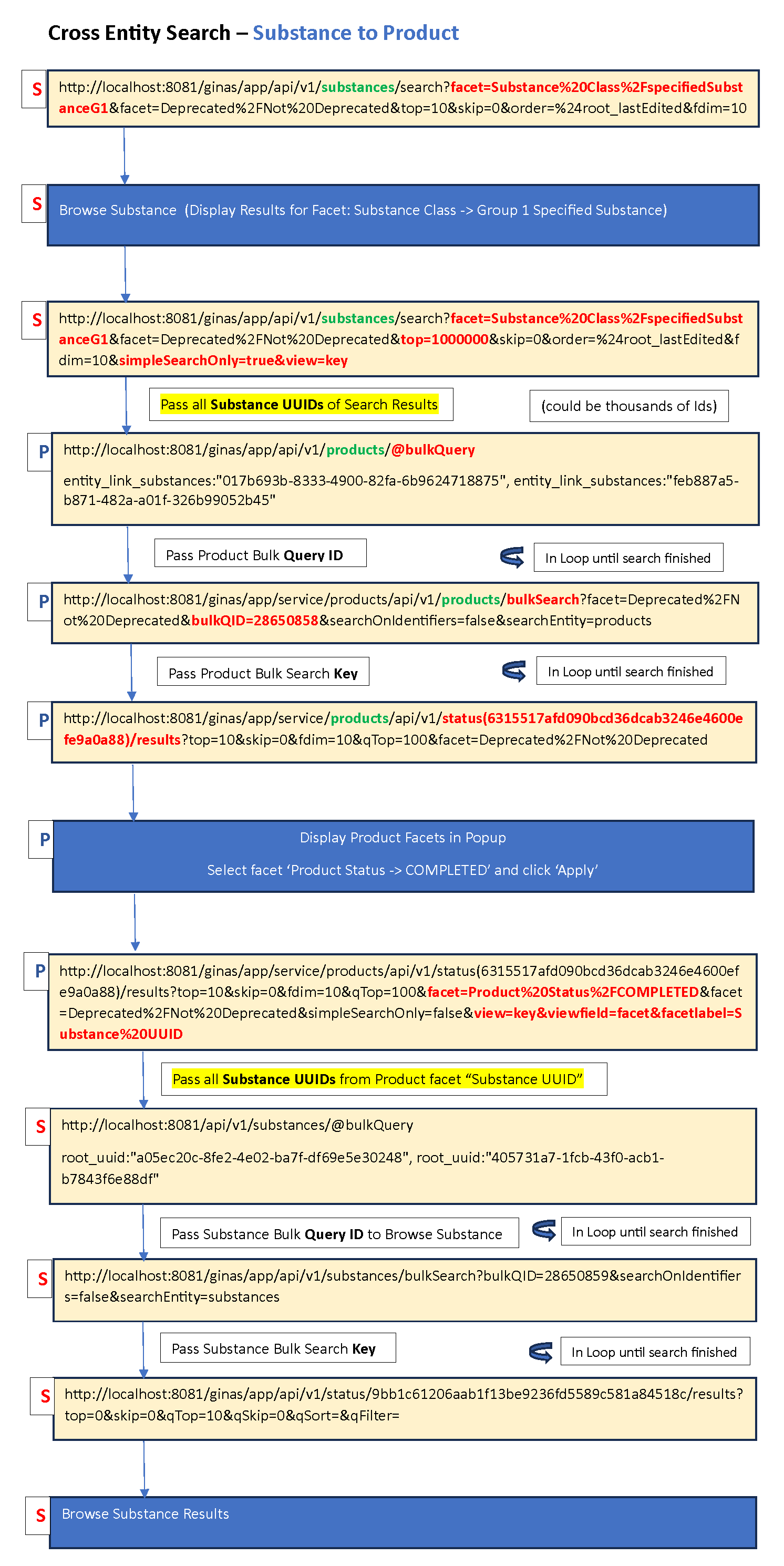
 ***Figure 1 – Show Facets of Other Entities***

 ***Figure 2 – Product Facets***

***Figure 3 – Product Facets Slected***

***Figure 4 – Substance Detail Page***

To achieve Cross-Entity search capability, it requires series of Bulk Query, Bulk Search, and Bulk Search Status Results REST API calls. In the frontend code, it uses Substance UUIDs to pass in the Bulk Query and use the bulk ID to get bulk Search and Bulk Status Results. See Figure 5 below to see the flow chart of the API calls.



***Figure 5 – API Calls***

The dropdown “Show Facet For” is configurable and can easily be hidden or shown on Browser by using the frontend configuration. If set to true, it will show the dropdown on the browser. If set to false, the dropdown will be hidden on the browser.

"showCrossEntitySearchDropdown": true,

**Notes for the future:**

Will continue to improve the Cross-entity search feature and increase the performance since it can be slow when searching large data set.

## Structure search improvements

We have added a new type of search: ‘Exact Plus’.

**Purpose and Motivation:**

This is a salt-stripped version of the existing Exact structure search.

**How it works:**

An Exact Plus search takes the input query molecule and removes any known salt or solvate fragment. (The list of salt and solvate fragments is configurable so you may add your own counterions or solvate structures.)

We then perform an exact search using the salt-stripped query molecule.

Note: if the salt-stripping process results in an empty structure, we revert to the input query structure. (An empty structure results from, for example, sodium tartrate, both of whose fragments are on the default salt list.)

**Notes for the Future:**

We are open to your feedback on the behavior of our structure searching options!

## Impurities Improvement:

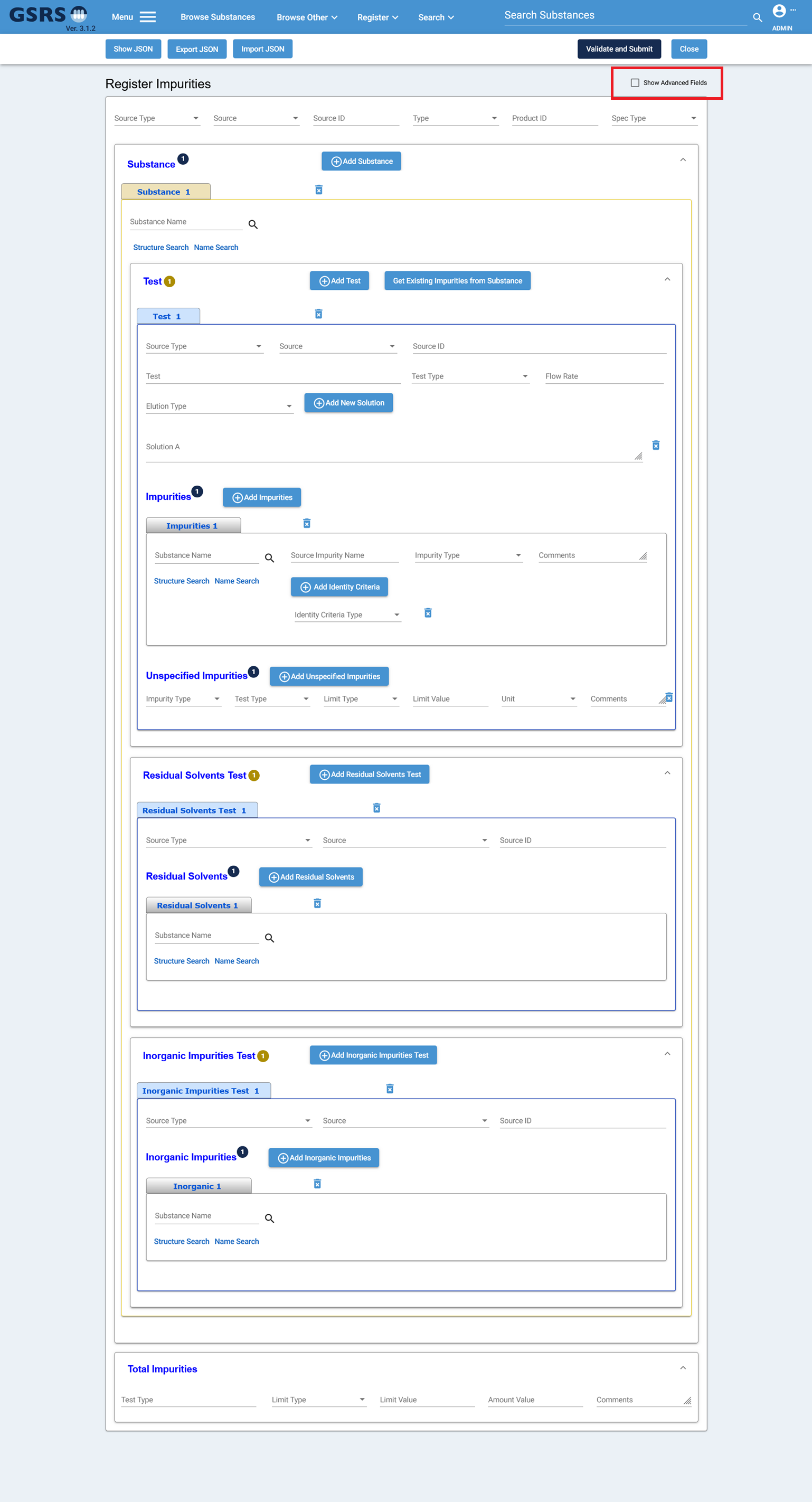
**Purpose and Motivation:**

The Register and Update Impurities form is a useful feature which allows users to enter data about impurities and monograph details. However, this form contains around hundreds of fields, which can be overwhelming to enter data if the user only needs to enter limited data. Therefore, the solution is to simplify the form so that either limited or lots of data can be entered into the form.

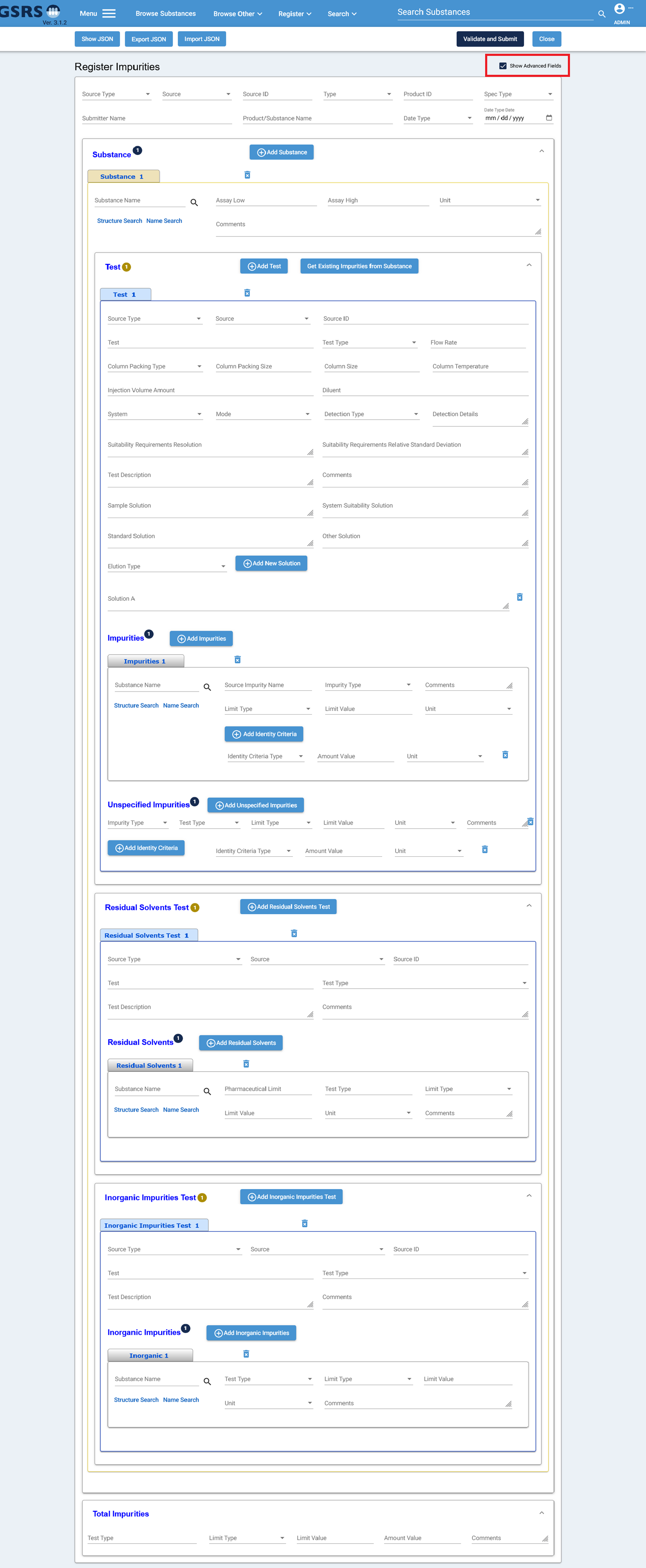
**How it works:**

In GSRS 3.1.2, in Register and Update Impurities pages, we added a checkbox “Show Advanced Fields”, which allow to show and hide fields which can be called either Simple form or Advanced form. When the check box is un-checked it will show simple form, meaning it will show limited fields. See *Figure 1* below. When the checkbox is checked, it will show advanced form, which means all the available fields are shown in the form. See *Figure 2 below.* This way we do not have to create two separate forms, and the solution can be achieved in one form itself. Each field is configurable in the frontend config file and reads the config details in the frontend code. Based on that the fields are displayed on the either Simple or Advanced form.

In the Simple Form, the form seems less crowded and makes it easy for user to enter the basic data. As well, it makes it easier for users to understand and interact with the form efficiently and minimize unnecessary fields and fill the form quickly. The Advanced form is also useful for some users as it can capture additional detailed data about the Impurities, which can be valuable for data analysis.



***Figure 6 – Simple Register Impurities***



***Figure 7 – Advanced Register Impurities***

The Impurities form fields can be configurable in the frontend config file, which can be used to decide which fields should display in Simple or Advanced view. In the configuration below, the value “simple” means that this field should be shown in the simple form. It also means the “Show Advanced Field” is unchecked. When value is “Advanced”, it means that this field should be shown in Advanced form along with fields in the Simple form. This also means that the “Show Advanced Field” checkbox is checked. Also, it is not used in the configuration here, but you can also use value “removed”, which means that field will not be shown at all, neither in Simple nor Advance Form. Having the fields configurable allows to manage the fields visibility based on the user’s need and will not need to change the frontend code and re-deploy the code on server.

"impuritiesForm": {

"settingsDisplay": {

"overview": {

"sourceType": "simple",

"source": "simple",

"sourceId": "simple",

"type": "simple",

"productId": "simple",

"specType": "simple",

"submitterName": "advanced",

"productSubstanceName": "advanced",

"dateType": "advanced",

"dateTypeDate": "advanced"

},

"substance": {

"substanceUuid": "simple",

"low": "advanced",

"high": "advanced",

"unit": "advanced",

"comments": "advanced"

},

"test": {

"sourceType": "simple",

"source": "simple",

"sourceId": "simple",

"test": "simple",

"testType": "simple",

"flowRate": "simple",

"columnPackingType": "advanced",

"columnPackingSize": "advanced",

"columnSize": "advanced",

"columnTemperature": "advanced",

"injectionVolumeAmount": "advanced",

"diluent": "advanced",

"system": "advanced",

"mode": "advanced",

"detectionType": "advanced",

"detectionDetails": "advanced",

"suitabilityReqResolution": "advanced",

"suitabilityReqRelStandardDeviation": "advanced",

"testDescription": "advanced",

"comments": "advanced",

"sampleSolution": "advanced",

"systemSuitabilitySolution": "advanced",

"standardSolution": "advanced",

"otherSolution": "advanced",

"elutionType": "simple",

"solutionDescription": "simple",

"mobilePhase": "simple"

},

"impurities": {

"relatedSubstanceUnii": "simple",

"sourceImpurityName": "simple",

"impurityType": "simple",

"comments": "simple",

"limitType": "advanced",

"limitValue": "advanced",

"unit": "advanced",

"identityCriteriaType": "simple",

"amountValue": "advanced",

"criteriaUnit": "advanced"

},

"unspecifiedImpurities": {

"impurityType": "simple",

"testType": "simple",

"limitType": "simple",

"limitValue": "simple",

"unit": "simple",

"comments": "simple",

"identityCriteriaType": "advanced",

"amountValue": "advanced",

"criteriaUnit": "advanced"

},

"residualSolventsTest": {

"sourceType": "simple",

"source": "simple",

"sourceId": "simple",

"test": "advanced",

"testType": "advanced",

"testDescription": "advanced",

"comments": "advanced"

},

"residualSolvents": {

"relatedSubstanceUuid": "simple",

"pharmaceuticalLimit": "advanced",

"testType": "advanced",

"limitType": "advanced",

"limitValue": "advanced",

"unit": "advanced",

"comments": "advanced"

},

"inorganicImpuritiesTest": {

"sourceType": "simple",

"source": "simple",

"sourceId": "simple",

"test": "advanced",

"testType": "advanced",

"testDescription": "advanced",

"comments": "advanced"

},

"inorganicImpurities": {

"relatedSubstanceUnii": "simple",

"testType": "advanced",

"limitType": "advanced",

"limitValue": "advanced",

"unit": "advanced",

"comments": "advanced"

},

"impuritiesTotal": {

"testType": "simple",

"limitType": "simple",

"limitValue": "simple",

"amountValue": "simple",

"comments": "simple"

}

}

}

**Notes for the Future:**

In the future will improve the Impurities form by updating the fields based on the requirements. We might remove or add fields to maximize the use of the data for research or decision-making purpose.

# Other selected Improvements and bug fixes:

## Improvement: A new Exporter: SQL Exporter

**Purpose and Motivation:**

This exporter provides the ability to use a SQL statement to drive a data export.

You can add multiple instances of this exporter, each configured with a different SQL statement. When the individual instance is selected, the SQL is run to fill a data file that is then available for download.

**How it works:**

Most exporters create a data file based on the current results set (generally used after a search). This exporter runs an SQL query over the *entire* database. It may be confusing to users that a recently-performed search has no bearing on the data extracted by the SQL Exporter!

Example configuration:

ix.ginas.export.exporterfactories.substances.list.SQLExporterFactory1 = {  
 "exporterFactoryClass": "gsrs.module.substance.exporters.SQLExporterFactory",  
 "parameters": {  
 "format": {  
 "extension": "dnames.xlsx",  
 "displayName": "Display Names (xlsx) File"  
 },  
 files: [  
 {  
 "name":"Names",  
 "format":"PostgreSQLCsv",  
 "header":"UNII,NAME",  
 "sql":"SELECT S.APPROVAL\_ID AS UNII, COALESCE(N.FULL\_NAME, N.NAME) AS NAME FROM IX\_GINAS\_SUBSTANCES S LEFT JOIN IX\_GINAS\_NAME N ON S.UUID = N.OWNER\_UUID AND N.DISPLAY\_NAME = '1'"  
 },  
 {  
 "name":"CAS",  
 "format":"PostgreSQLCsv",  
 "header":"UNII,CAS",  
 "sql":"SELECT S.APPROVAL\_ID AS UNII, C.CODE AS CAS FROM IX\_GINAS\_SUBSTANCES S LEFT JOIN IX\_GINAS\_CODE C ON S.UUID = C.OWNER\_UUID AND C.CODE\_SYSTEM = 'CAS' AND C.TYPE = 'PRIMARY'"  
 }  
 ]  
 }  
}

Note: that the PostgresQLCsv format can be used with other RDBMS!

**Notes for the Future:**

The team may look for an alternative way to run SQL Exporter and similar exporters that do not use the current result set.

## Improvement: ScheduledExportTask

**Purpose and Motivation:**

This scheduled task provides a way to run a report on a regular basis. The report uses a configured GSRS API query to find records in the database, write a pre-configured list of fields for these records to a file and then send the data to various destinations:

* A file on disk
* Email recipients
  + Data can be sent as an attachment or in the body of the email message
* Ftp servers

**How it works:**  
You add one or more instances of this task to your configuration (for the substance service) and the task becomes available to admins in the Scheduled Jobs tab of the admin panel.

Add this to the pom.xml file for the substance service:

<dependency>

<groupId>com.sun.mail</groupId>

<artifactId>jakarta.mail</artifactId>

<version>2.0.1</version>

</dependency>

<dependency>  
 <groupId>com.sun.activation</groupId>  
 <artifactId>jakarta.activation</artifactId>  
 <version>2.0.1</version>  
 </dependency>

Sample configuration:

gsrs.scheduled-tasks.list.ScheduledExportTask1 = {

"scheduledTaskClass" : "gsrs.module.substance.tasks.ScheduledExportTask",

"parameters" : {

"cron": "0 10 1 \* \* ?",

"autorun": false,

"description": "Scheduled export of substances edited over the past month",

"extension": "csv",

"query": "root\_codes\_codeSystem:\"^FDA UNII$\" AND root\_lastEdited:[P1M TO 10E50]",

"filenameTemplate": "auto-export-$DATE$",

"publicOnly": false,

"username": "ADMIN",

"parameters": {},

"destinations": [

{

"uri": "file:///home/srs/exports/export.csv"

},

{

"uri":"sftp://target\_server/inbox/last\_month\_edited.csv",

"user":"username",

"password":"PASSWORD",

"userDirIsRoot":"false",

"strictHostKeyChecking":"no",

"sessionTimeoutMillis":"10000"

},

{

"uri":"ftp://target\_server\_2/gsrs\_exports/",

"user":"username",

"password":"PASSWORD",

"userDirIsRoot":"true",

"pasiveMode":"true"

},

{

"uri": "smtp://mail.server.org:25",

"from": "gsrs@server.org",

"to": "some.user@server.org",

"subject": "Substances edited last month",

"body": "<h1>Substances</h1></br><table><tr><th>Approval ID</th><th>Display Name</th></tr>",

"recordTemplate": "<tr><td><a href='https://gsrs.ncats.nih.gov/ginas/app/ui/substances/{0}'>{1}</a></td><td>{2}</td></tr>",

"footer": "</table>",

"charset": "utf-8",

"maxSize": "10M"

},

{

"uri": "smtp://mail.server.org:25/last\_month\_edited.csv",

"from": "gsrs@server.org",

"to": "some.user@server.org",

"subject": "Substances edited last month",

"body": "Substances edited last month",

"maxSize": "10M"

}

]

}

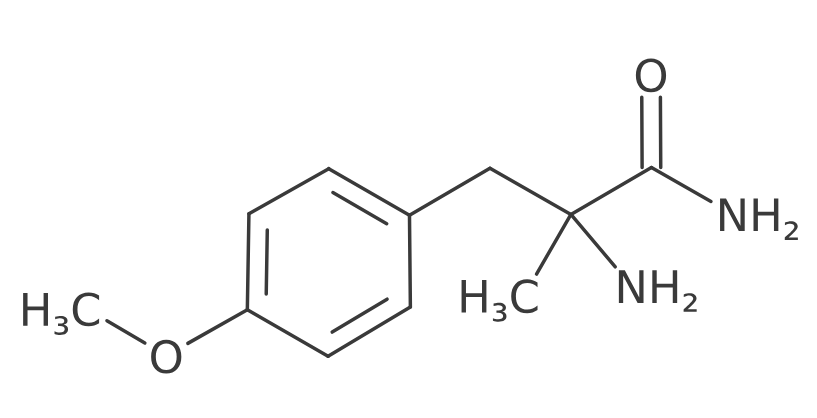
}

## Improvement: Two InChIKeys for certain chemicals

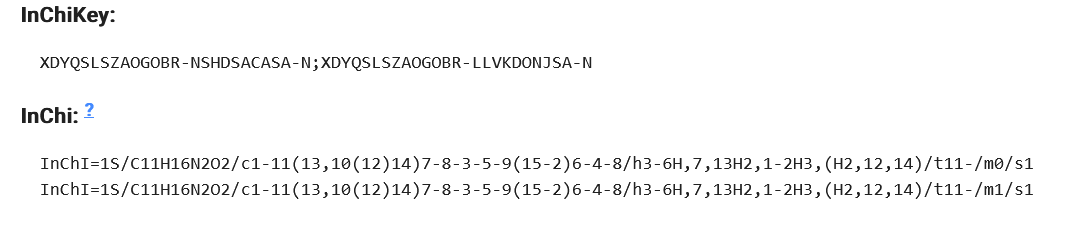
**Purpose and Motivation:**

Some chemical species can we represented by more than one structure and therefore have more than one InChI and InChIKey. in this improvement, we provide a pair of InChIs and InChIKeys for those species with (+/-) optical activity *and* at least one defined stereocenter.

For example, consider 2-Amino-3-(4-methoxyphenyl)-2-methylpropanamide

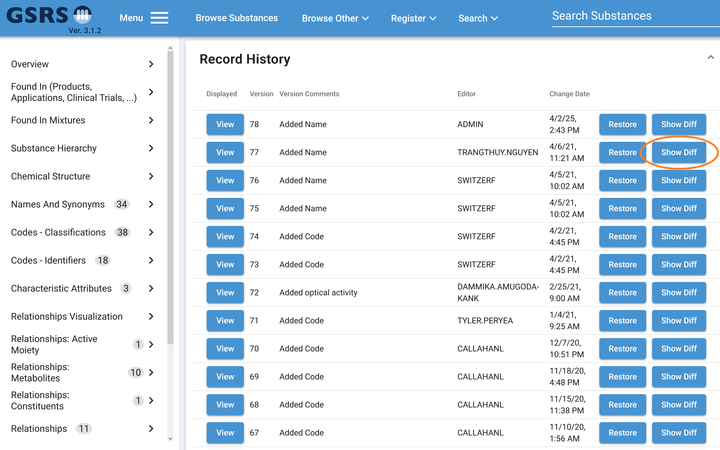
  
***Figure 8 – Example Chemical Structure***

The above structure has no stereochemical markings at the chiral carbon and is marked as a mixture of two enantiomers. This version of GSRS displays 2 InChIs and 2 InChIKeys:

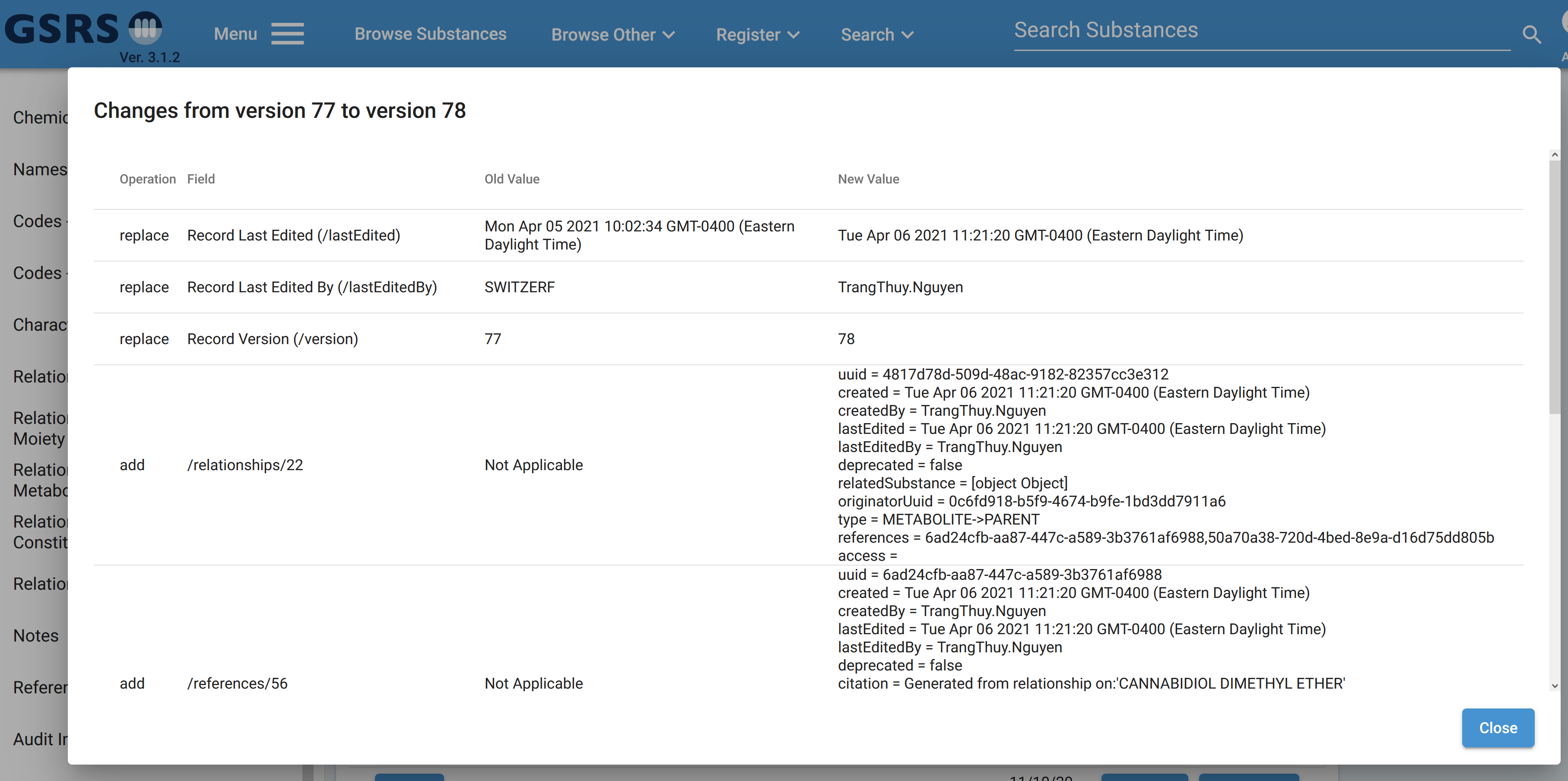
**How it works:**   
Behind the scenes. GSRS generates both enantiomers and calculates the InChI and InChIKey of each.

## Improvement: Show changes between two adjacent versions a substance

**Purpose and Motivation:**   
The record change history displays the time of the change, the editor, and any associated comments. However, this information is sometimes insufficient, as it doesn't indicate exactly what was modified during the edit. The change could be as minor as a typo correction or as significant as updates to multiple fields. To provide more clarity, a **"Show Diff"** button has been added in this release.  
  
**How it works:**  
In the record history page, a **"Show Diff"** button has been added, as shown in Figure 9. When clicked, a popup appears displaying all the changes in detail. As illustrated in Figure 10, some fields have been added, while others have been updated. The popup lists the field names along with their old and updated values.



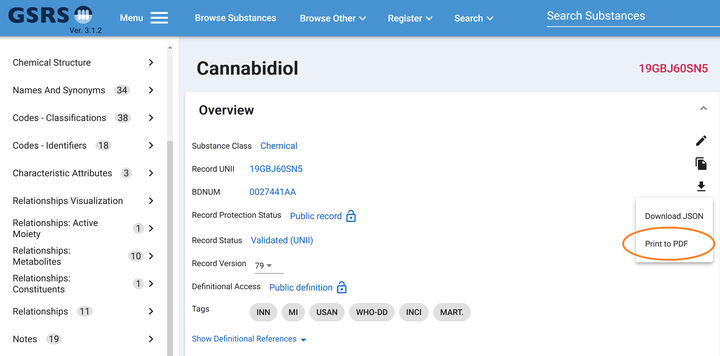
***Figure 9 – "Show Diff” Button***

***Figure 10 – Example Chemical***

## Improvement: PDF download option

**Purpose and Motivation:**   
PDF download option for a single substance is added to this release.

**How it works:**  
On the substance detail page, select the download icon, then choose **Print to PDF** to generate a printable PDF format of the substance.

***Figure 11 – Example Chemical***

## Improvement: JmesPathValidator allows you to create new business rules easily

**Purpose and Motivation:**

JMESPath is a query language for JSON. You can extract and transform elements from a JSON document. (from <https://jmespath.org/tutorial.html>) Since GSRS records are represented in JSON, you can use JMESPath to analyze parts of a GSRS record.

**How it works:**

Add a section to the configuration of the substance service (in file substances.conf, for example), that includes one or more JMESPath expressions

gsrs.validators.substances.list.jmes1 = {

"validatorClass" = "ix.ginas.utils.validation.validators.JmespathValidator",  
 "newObjClass" = "ix.ginas.models.v1.Substance",  
 "configClass" = "SubstanceValidatorConfig",  
 "parameters"= {  
 "expressions" = [  
 {"messageType": "ERROR", "messageTemplate": "The backslash character(s) in name '%s' are not allowed.", "expression": "new.names[?contains(name, '\\\\')].name | [0]"}

]  
 }

The above example shows a way to prevent a user from entering ‘\’ inside a substance name. The possibilities are endless.

## Improvement: GsrsApiExporterFactory allows you to transfer data between instances of GSRS in real time

**Purpose and Motivation:**

This new exporter allows you synchronize data between the GSRS you are logged into and another GSRS system where you also have login credentials.

**How it works:**

Add a section to the configuration of the substance service (in file substances.conf, for example), similar to this

ix.ginas.export.exporterfactories.substances.list.GsrsApiExporterFactory = {

"exporterFactoryClass": "gsrs.module.substance.exporters.GsrsApiExporterFactory",

"order": 1001,

"parameters": {

"format": {

"extension": "gsrsapi",

"displayName": "Send to GSRS on Alternate Laptop"

},

"headers": {

"auth-username": "admin",

"auth-token": "....."

},

"baseUrl": "https://other\_computer.ourcompany.com:8080/api/v1/substances",

"timeout": "120000",

"trustAllCerts": true,

"allowedRole": "Approver",

"newAuditor": "admin",

"changeReason": "API Test (Version {{version}})",

"validate": false

}

}

Now, when you perform a search and wish to export data, one of the options is ‘Send to GSRS on Alternate Laptop.’

## Improvement: SQLExporterFactory allows you to write the results of a SQL query to a file

**Purpose and Motivation:**

When you have a SQL query that you need to run periodically and extract the results to a file, this exporter can work for you.

**How it works:**

Add a section like this to the configuration of the substance service (in file substances.conf, for example):

ix.ginas.export.exporterfactories.substances.list.SQLExporterFactory = {

"exporterFactoryClass": "gsrs.module.substance.exporters.SQLExporterFactory",

"parameters": { # "format": {

"extension": "cas.zip",

"displayName": "CAS Codes Report Combination (zip) File",

"extension": "unii\_data.zip",

"displayName": "UNIIs with names, and with CAS Codes Report (zip) File"

},

files: [

{

"name":"Names.csv",

"encoding":"ISO-8859-1",

"delimiter":",",

"escapeChar":"",

"quoteChar":""",

"header":"UNII;NAME",

"sql":"SELECT S.APPROVAL\_ID AS UNII, COALESCE(N.FULL\_NAME, N.NAME) AS NAME FROM IX\_GINAS\_SUBSTANCE S LEFT JOIN IX\_GINAS\_NAME N ON S.UUID = N.OWNER\_UUID WHERE S.DEPRECATED = '0' AND N.DISPLAY\_NAME = '1' "

}

]

}

}

This exporter is somewhat different from most exporters. Most exporters work off the current list of records (from a search you just performed). The SQLExporter exports the data returned by the configured SQL query, ignoring the current list.

This exporter runs the configured SQL and writes the results to the configured file, optionally compressing the file when finished.

Improvement: ScheduledExportTask automatically runs SQL and sends the results to configured destinations.

**Purpose and Motivation:**

This scheduled task automatically runs a GSRS query, according to a configured schedule, writes the results to a file and then sends the file to various destinations. You can have the results routed to

* An FTP servers
* An email address (with the data as either an attachment or the body of the message).
* Local disk file

**How it works:**

Add a section like this to the configuration of the substance service (in file substances.conf, for example):

gsrs.scheduled-tasks.list.ScheduledExportTask2 = {

"scheduledTaskClass": "gsrs.module.substance.tasks.ScheduledExportTask",

"order": 393101,

"parameters": {

"cron": "0 10 1 \* \* ?",

"autorun": false,

"description": "Report of polymer substances with an RXCUI code",

"extension": "csv",

"query": "root\_substanceClass:"polymer" AND root\_codes\_codeSystem:\"^RXCUI$\"",

"filenameTemplate": "auto-export-$DATE$",

"publicOnly": false,

"username": "ADMIN",

"parameters": {},

"destinations": [

{

"uri": "<file://D:/temp/reports/RXCUI_polymers.csv>"

},

{

"uri": "smtp://50.87.144.106:587",

"from": "[user1@compan](mailto:mitch.miller@thinkscience.us)1.com”,

"to": "[user2@company2.com](mailto:mitch.miller@yahoo.com)",

"subject": "Interesting Substances",

"body": "<h1>Substances</h1></br><table><tr><th>Approval ID</th><th>Display Name</th></tr>",

"recordTemplate": "<tr><td><a href='http://localhost:8081/ginas/app/ui/substances/{0}'>{1}</a></td><td>{2}</td></tr>",

"footer": "</table>",

"charset": "utf-8",

"maxSize": "10M",

"startTls": true,

"user": "user1@company1.com",

"password": ""

},

]

}

}

The query is formulated using the GSRS API.

Note that GSRS scheduled tasks can be configured to run automatically according to a fixed schedule (for example, once per day) or run ad-hoc by a user with the ADMIN role.

## Improvement: JmespathSpreadsheetExporterFactory makes it easy to create a custom spreadsheet exporter

**Purpose and Motivation:**

The new JmespathSpreadsheetExporterFactory provides a way to configure the export of data to a Excel sheet using a series of JMESPath expressions.

JMESPath is a query language for JSON. You can extract and transform elements from a JSON document. (from <https://jmespath.org/tutorial.html>) Since GSRS records are represented in JSON, you can use JMESPath to extract parts of a GSRS record.

**How it works:**

Add a section like this to the configuration of the substance service (in file substances.conf, for example):

ix.ginas.export.exporterfactories.substances.list.JmespathSpreadsheetExporterFactory = {

"exporterFactoryClass": "gsrs.module.substance.exporters.JmespathSpreadsheetExporterFactory",

"parameters": {

"format": {

"extension": "custom-report.xlsx",

"displayName": "Jmespath Custom Report (xlsx) File" },

"columnExpressions": [

{

"name":"UUID", "expression":"uuid"},

{"name":"NAME", "expression":"\_name"}, {"name":"APPROVAL\_ID", "expression":"\_approvalIDDisplay"}, {"name":"SMILES", "expression":"structure.smiles"},

{"name":"FORMULA", "expression":"structure.formula"},

{"name":"SUBSTANCE\_TYPE", "expression":"substanceClass"},

{"name":"STD\_INCHIKEY", "expression":"structure.inchikey"},

{"name":"STD\_INCHIKEY\_FORMATTED", "expression":"structure.inchikeyf"}, {"name":"CAS", "expression":"codes[?codeSystem=='CAS'].code","delimiter":"|"}, {"name":"EC", "expression":"codes[?codeSystem=='ECHA (EC/EINECS)'].code"}, {"name":"ITIS", "expression":"codes[?codeSystem=='ITIS'].code"},

{"name":"NCBI", "expression":"codes[?codeSystem=='NCBI TAXONOMY'].code"},

{"name":"USDA\_PLANTS", "expression":"codes[?codeSystem=='USDA PLANTS'].code"},

{"name":"INN", "expression":"codes[?codeSystem=='INN'].code"},

{"name":"NCI\_THESAURUS", "expression":"codes[?codeSystem=='NCI\_THESAURUS'].code"},

{"name":"RXCUI", "expression":"codes[?codeSystem=='RXCUI'].code"},

{"name":"PUBCHEM", "expression":"codes[?codeSystem=='PUBCHEM'].code"},

{"name":"MPNS", "expression":"codes[?codeSystem=='MPNS'].code"},

{"name":"CompTox", "expression":"codes[?codeSystem=='EPA CompTox'].code"},

{"name":"INGREDIENT\_TYPE", "expression":"relationships[?contains(['IONIC MOIETY', 'MOLECULAR FRAGMENT', 'UNSPECIFIED INGREDIENT', 'SPECIFIED SUBSTANCE'], type)].type || 'INGREDIENT SUBSTANCE'"},

{"name":"PROTEIN\_SEQUENCE", "expression":"protein.subunits[].sequence", "delimiter":"|"},

{"name":"NUCLEIC\_ACID\_SEQUENCE", "expression":"nucleicAcid.subunits[].sequence", "delimiter":"|"},

{"name":"RECORD\_ACCESS\_GROUPS", "expression":"access", "delimiter":"|"}, {"name":"LAST\_EDITED","expression":"lastEdited", "datetime":"yyyy-MM-dd HH:mm:ss"}

]

}

}

Where each item in the columnExpressions collection describes a column in the spreadsheet, with the ‘expression’ in JMESPath. This exporter works on the records from the most recent search.

## 

## Bug fix: Pasting certain SMILES into a GSRS structure box resulted in an improbably structure

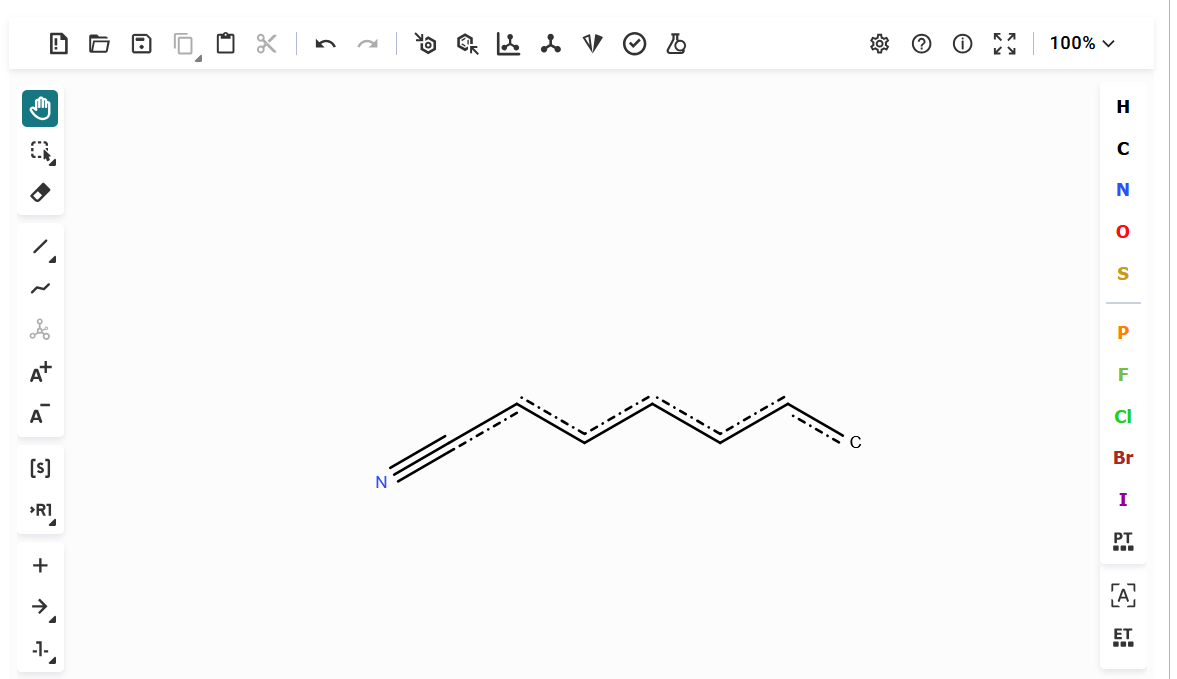
**Purpose and Motivation:**

When SMILES strings containing a pound sign (‘#,’ symbolic of a triple bond) were pasted into a GSRS structure search or registration box, a structure was generated with most bonds given the ‘single or aromatic’ type, reserved for query. This was probably different from the user’s intention.

Example:

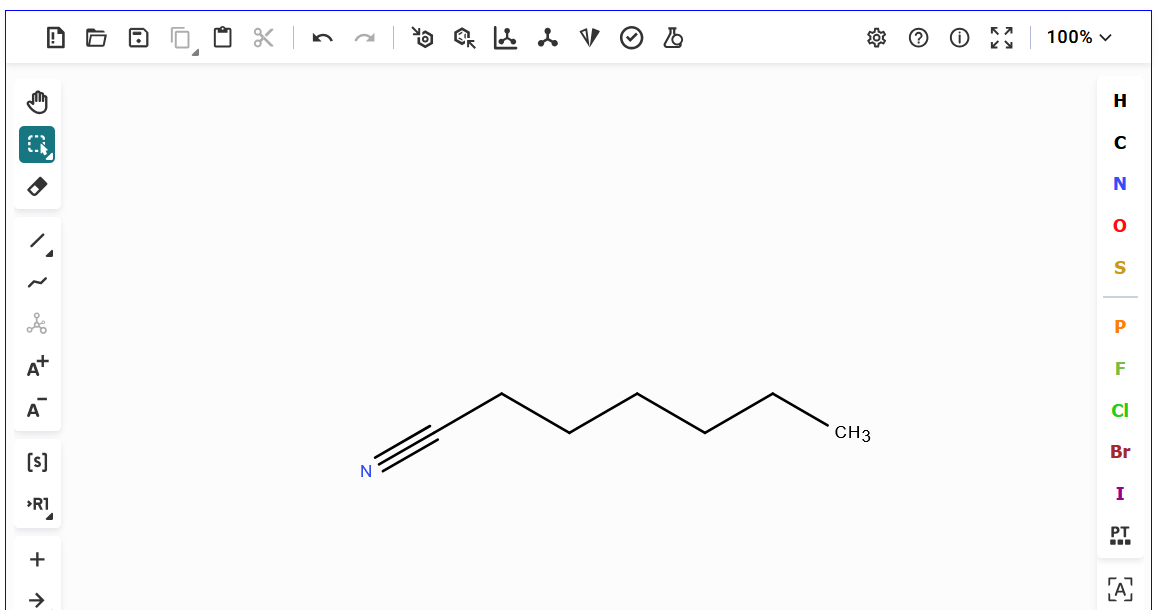
SMILES: CCCCCCC#N (heptanenitrile)

Result:

***Figure 12 – The Improbably Structure***

**How it works:**

The issue was caused by molwitch-cdk, one of our libraries, interpreting ‘#’ as indicating that input string was SMARTS. With this fix, we now perceive bond types correctly.

***Figure 13 – The Correct Structure***

## Bug fix: User Saved List

A user list could previously be saved without specifying a name, resulting in it being saved with the name “undefined.”   
This issue has been fixed by requiring users to enter a name before saving a user list.

Previously, admins could not view user lists created by other users.   
This has also been fixed. The intended business logic is as follows:

* A regular user can only access their own saved user lists.
* A user with the **admin** role can access both their own saved lists and those of all other users.

Improvement: Update fields definition in Impurities tables  
  
**Purpose and Motivation:**

If you have deployed the Impurities Entity Service, please be aware that we have modified the data model for impurities. Two columns in the SRSCID\_IMPURITIES\_TEST table need to be updated to accommodate larger data sizes, as the original data types could not store all the required information.

**How it works:**

In the table class of the backend code corresponding to the SRSCID\_IMPURITIES\_TEST table, Lob is now used for the TEST\_DESCRIPTION and COMMENTS columns instead of String in the Impurities Entity Service. Currently in database, the column data type of these two columns are listed below with respect to different database flavor.

Oracle: CLOB  
PostGeSQL: TEXT

MariaDB: LONGTEXT  
MySQL: LONGTEXT  
  
Please check <https://github.com/ncats/gsrs3-main-deployment/tree/main/impurities/database>to find the right full SQL Script or the Delta Script under folder GSRS\_3.1.2 for your database flavor.

# Notes

JSDraw License and Configuration to Disable JSDraw

GSRS currently supports both JSDraw and Ketcher as structure editors. We hold a one-year license for JSDraw, valid through July 2026. This license permits non-commercial use and applies exclusively to public-facing GSRS instances that are freely accessible to the public.   
  
If your intended use falls outside of this scope, please contact Scilligence to obtain the appropriate licensing. Alternatively, you may use Ketcher and disable JSDraw. To do so, update the configuration in the frontend's config.json file.   
  
"disableJSDraw": true