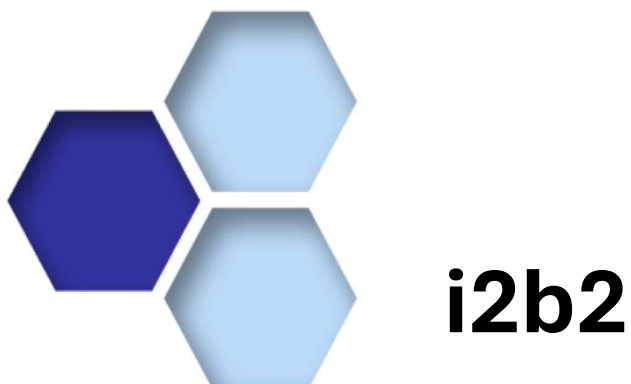


---

## Informatics for Integrating Biology and the Bedside



# User Guide

# **NCBO Extraction 2.1**

## Contents

I. Overview.....	3
II. Prerequisites .....	3
1. BioPortal API Key .....	3
2. Ontology Acronym.....	3
3. Java Runtime Environment 8 or 11 .....	4
III. Download NCBO Software .....	4
IV. Quick Start - Command Examples .....	5
1. <i>ExtractAll</i> .....	5
2. <i>ProcessAll</i> .....	5
V. Command Details and the Full Workflow .....	6
1. <i>ExtractAll</i> .....	6
2. Load Staging File into Staging Table.....	7
3. <i>ProcessAll</i> .....	7
4. Load Metadata File into Metadata Table .....	9
5. Configuring i2b2 to Use your new Metadata .....	10
5.1 Table_Access .....	10
5.2 Schemes .....	11
VI. Software Limitations.....	11
VII. Developer Support.....	12
1. Project Description.....	13
2. Compilation in Eclipse .....	13
3. <code>build.xml</code> Compilation Options .....	14
4. The <code>edu.harvard.i2b2.xml</code> Project.....	15
5. Changing JRE for Running ANT Build Scripts in Eclipse .....	15

## I. Overview

The NCBO Extractor is a tool that allows users to download ontologies from the NCBO BioPortal (<https://bioportal.bioontology.org/>), the world's most comprehensive repository of biomedical ontologies, and import them into i2b2 instances. The workflow is illustrated in Figure 1 below.



**Figure 1.** NCBO Extraction Tool's Workflow from downloading ontology from BioPortal to importing them into an i2b2 instance's metadata table.

Using NCBO's **ExtractAll** command, users download ontologies from the BioPortal. The ontology is saved as a file called the *staging file*. The file can then be imported into a *staging table* in a database. The users then run the **ProcessAll** command to compute for the ontology structure and all i2b2-required fields. The resulting file is a *metadata file*, which can then be imported into the *metadata table* in an i2b2 database.

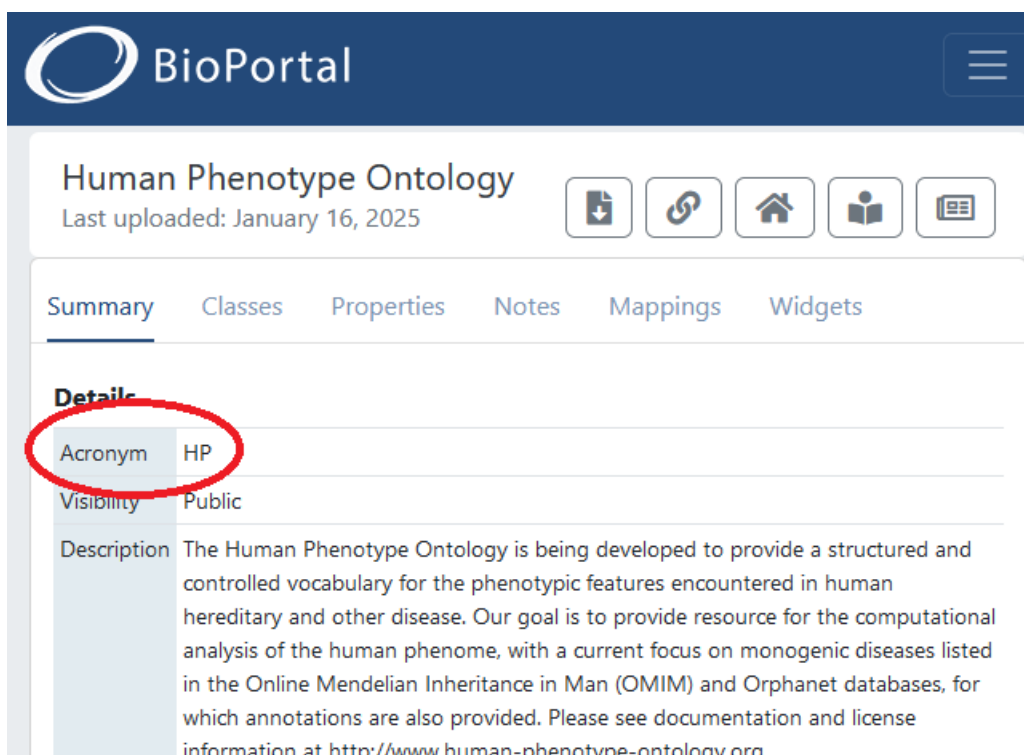
## II. Prerequisites

### 1. BioPortal API Key

Since NCBO Extractor uses NCBO BioPortals REST API, you will need a BioPortal API Key. This can be obtained by creating a free account in BioPortal. Your key will appear in the Account area.

### 2. Ontology Acronym

When you run the ExtractAll command, you will need to specify an ontology acronym, which specifies which ontology to download from BioPortal. You can see the ontology's acronym in its page in BioPortal. For example, the **Human Phenotype Ontology's** acronym is **HP** (Figure 2).



*Figure 2. Location of Human Phenotype Ontology's acronym on BioPortal.*

### 3. Java Runtime Environment 8 or 11

Because NCBO Extractor Tool still relies on a few older libraries, you will need to have the correct Java Runtime Environments (JRE) installed to run NCBO Extractor. You will need Java 8 or Java 11.


Java 8 download: <https://www.java.com/en/download/manual.jsp>

Java 11 download: <https://www.oracle.com/java/technologies/java-archive-downloads-javase11-downloads.html>

Scroll down for the JRE-only versions. JREs also come with Java Development Kits (JDK) downloads.

## III. Download NCBO Software from GitHub

NCBO Extractor is available at GitHub: [i2b2plugins/ncbo-extraction-tool](https://github.com/i2b2plugins/ncbo-extraction-tool)

1. Click **Code** (  ) to reveal a dropdown menu
2. Select **Download ZIP**
3. Unzip the zip file into a folder, say, `ncbo_extractor_2024_main`
4. Navigate to  
`ncbo_extractor_2024_main/NCBO_Extractor_2024_Java_Source/edu.harvard.i2b2.ncboExtraction/dist`

The `dist` folder is where you can run the *ExtractAll* and *ProcessAll* commands. It contains all required byte code and libraries to run. Since it is a standalone folder, you may move it in its entirety to somewhere more conveniently accessible.

## IV. Quick Start - Command Examples

### 1. *ExtractAll*

To run the *ExtractAll* command from the command line, you would type the following into your command prompt or terminal. In examples below, we use commands on Windows as examples. As a result semi-colons ( `;` ) are used as path separator. To run the commands on Linux, Unix, or Mac, please use colons ( `:` ) instead.

```
PATH_TO_JAVA -classpath endorsed_lib/*;genlib/i2b2Common-
core.jar;lib/commons/*;lib/log4j/*;lib/jdbc/*;lib/spring/*;
edu.harvard.i2b2.ncbo.extraction.NCBOOntologyExtractAll -ont ONTOLOGY_ACRONYM -apikey YOUR_API_KEY -
outputFileName OUTPUT_FILE
```

Substitute the capitalized text with your use case. For example, the following command downloads the `HP` ontology with the specified `API key` into the `hp_staging_file.txt` file using the specified `Java 8 runtime`.

```
"C:\Program Files\Java\jdk-1.8\bin\java.exe" -classpath endorsed_lib/*;genlib/i2b2Common-
core.jar;lib/commons/*;lib/log4j/*;lib/jdbc/*;lib/spring/*;
edu.harvard.i2b2.ncbo.extraction.NCBOOntologyExtractAll -ont HP -apikey 12345678-1234-1234-abcd-
1234567890ab -outputFileName hp_staging_file.txt
```

### 2. *ProcessAll*

To run the *ProcessAll* command from the command line, you would type the following into your command prompt or terminal. We use commands on Windows as examples. As a result semi-colons ( `;` ) are used as path separator. To run the commands on Linux, Unix, or Mac, please use colons ( `:` ) instead.

```
PATH_TO_JAVA -classpath endorsed_lib/*;genlib/i2b2Common-
core.jar;lib/commons/*;lib/log4j/*;lib/jdbc/*;lib/jdbc/sqlserver2005/*;lib/spring/*;
edu.harvard.i2b2.ncbo.extraction.NCBOOntologyProcessAll -outputFileName OUTPUT_FILE -prefix PREFIX -
pathFormat PATHFORMAT -rootNodeName ROOT_NAME
```

Substitute the red text with your use case. For example, the following command uses the specified `Java 8 runtime` to process the staged `HP` ontology table to write the output to metadata file

`HP_METADATA.txt` with basecode prefix `HP` and pathformat `Medium` and root node name `HP_(NCBO)`.

```
"C:\Program Files\Java\jdk-1.8\bin\java.exe" -classpath endorsed_lib/*;genlib/i2b2Common-
core.jar;lib/commons/*;lib/log4j/*;lib/jdbc/*;lib/jdbc/sqlserver2005/*;lib/spring/*;*
edu.harvard.i2b2.ncbo.extraction.NCBOntologyProcessAll -outputFileName HP_METADATA.txt -prefix HP -
pathFormat Medium -rootNodeName "HP_(NCBO)"
```

## V. Command Details and the Full Workflow

### 1. *ExtractAll*

The *ExtractAll* command uses the following parameters. The first three parameters are required.

Parameter	Description
-ont	The acronym of the BioPortal Ontology you want to extract
-apikey	Your BioPortal API key
-outputFileName	Name of the output file ( <i>staging file</i> ) to where your downloaded data will be saved
-timeout	Optional timeout parameter in ms (default is 300,000 ms)
-startPage	Optional parameter specifying which page to start downloading (default is 1)

This command downloads a specified ontology one page at a time. The larger the ontology the more pages there are. When the command is executing, you will see the following status printed in your terminal:

```
INFO [main] (?:?) - Writing page: 1 of 448
INFO [main] (?:?) - Writing page: 2 of 448
...
INFO [main] (?:?) - Extraction completed
```

The “**Extraction Completed**” message indicates the extraction process has successfully completed. The status is also logged to the log file *hierarchy.log*.

It is possible that you experience network timeouts while the extraction is running. Any given call that times out will be retried twice; after that the program assumes that the network is down and ends the extraction program. The program defaults to a 300 second timeout. You may choose to change the timeout length through an optional ‘-timeout’ parameter. Timeouts are specified in milliseconds. A 500-second timeout would be specified as “-timeout 500000” on the Extraction tool command line.

If the command gives up due to network timeout problems, you can use the `-startPage` parameter to resume the download starting at your specific page.

When extraction is complete, the output staging file is ready. It is in pipe- delimited (‘|’) format. The first row of the file contains the column headings. Strings are quoted with the double quotes (“”).

## 2. Load Staging File into Staging Table

Create a table in your database to host the data in your staging file. If you are using SQLServer, you may use the following command to create a destination *staging table*:

```
CREATE TABLE [DATABASE_NAME].[SCHEMA_NAME].[STAGING_TABLE_NAME]
(
    "c_hlevel" INT NULL,
    "c_fullname" VARCHAR(700) NULL,
    "c_name" VARCHAR(2000) NOT NULL,
    "c_synonym_cd" CHAR(1) NOT NULL,
    "c_visualattributes" CHAR(3) NOT NULL,
    "c_basecode" VARCHAR(50) NULL,
    "c_facttablecolumn" VARCHAR(50) NOT NULL,
    "c_tablename" VARCHAR(50) NOT NULL,
    "c_columnname" VARCHAR(50) NOT NULL,
    "c_columndatatype" VARCHAR(50) NOT NULL,
    "c_operator" VARCHAR(10) NOT NULL,
    "c_dimcode" VARCHAR(700) NULL,
    "c_tooltip" VARCHAR(700) NULL,
    "sourcesystem_cd" VARCHAR(50) NULL,
    "c_symbol" VARCHAR(2000) NOT NULL,
    "c_path" VARCHAR(700) NULL,
    "i_snomed_ct" VARCHAR(50) NULL,
    "i_snomed_rt" VARCHAR(50) NULL,
    "i_cui" VARCHAR(50) NULL,
    "i_tui" VARCHAR(50) NULL,
    "i_ctv3" VARCHAR(50) NULL,
    "i_full_id" VARCHAR(100) NULL,
    "update_date" VARCHAR(50) NOT NULL,
    "parent_fullId" VARCHAR(50) NULL,
    "m_applied_path" VARCHAR(900) NOT NULL
);
```

For the column `update_date`, you may choose to use the `datetime` datatype.

For Oracle and PostgreSQL users, please use the following mapping table for datatypes.

SQLServer Datatype	Oracle Datatype	PostgreSQL Datatype
INT	NUMBER	INT
VARCHAR	VARCHAR2	VARCHAR
CHAR	CHAR	CHAR
DATETIME	DATE	DATETIME

After your staging table is set up. You may use a database-specific importer or a script to programmatically import your staging file into it.

## 3. ProcessAll

The *ProcessAll* command queries the staging table to build the ontology structure. This computes all parent-child relationships and all i2b2-required fields before writing everything to a metadata file as the output. All information required to access the database is stored in the `edu.harvard.i2b2.ncboExtraction/ExtractionApplicatioContext.xml`. The snippet below shows the default template. You will need to fill in the appropriated values for `driverClassName`, `url`, `username`, `password`, `db_fullSchema`, `stagingTable`. The template below uses Microsoft SQLServer as an example. For Oracle and Postgres, please use their corresponding `driverClassName` and connection `url`.

MS SQLServer users should also be aware that this example template eschews Transport Layer Security (TLS) encrypted connection in its connection string (i.e. "encrypt=false"). Users should configure it appropriately for production settings.

```
<beans>
  <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">
    <property name="driverClassName" value="com.microsoft.sqlserver.jdbc.SQLServerDriver"/>
    <property name="url" value="jdbc:sqlserver://HOST:PORT;encrypt=false"/>
    <property name="username" value="USERNAME"/>
    <property name="password" value="PASSWORD"/>
    <property name="defaultAutoCommit" value="false"/>
    <property name="defaultReadOnly" value="false"/>
  </bean>

  <bean id="database" class="edu.harvard.i2b2.ncbo.model.DBInfoType">
    <property name="db_fullSchema" value="dbo"/>
    <property name="stagingTable" value="ONTOLOGY_STAGING_TABLE"/>
  </bean>

  <bean id="conceptDAO" class="edu.harvard.i2b2.ncbo.dao.ConceptPersistDao">
    <property name="dataSource" ref="dataSource"/>
  </bean>
</beans>
```

Once you have set up `ExtractionApplicationContext.xml`, you can run the *ProcessAll* command. It has the following parameters.

Parameter	Description
-outputFileName	Output file name
-prefix	Optional parameter to prepend a prefix (scheme) to basecodes.
-pathFormat	Optional parameter to set a format for path. Can be s[hort], m[edium] or r[eadable]. Medium is the default.
-rootNodeName	Optional parameter to specify a single root for the ontology. When specified, root nodes of the original ontology will be children of this specified root node. This makes the ontology more organized in i2b2 UI. While optional, this is recommended. Names with more than one word should be quoted.

Setting `-prefix` enables all basecodes in the ontology to be prepended by it. For example, if an original basecode is "0000201", setting "-prefix HP" makes it "HP:0000201". *ProcessAll* only applies the prefix if the basecode is not already prepended with it to avoid the strange-looking results such as "HP:HP:0000201".

A concept's `c_fullname` is a path from the root of the ontology to the concept. Each concept is represented by a shortened identifier in `c_fullname`. The `-pathFormat` parameter sets a length limit for the shortened identifier for all concepts. There are three possible values: Short, Medium, and Readable. Short limits each concept to be 4 characters-long hash. Medium's limit is 20 characters. Readable's limit is 32. Because the total number of characters in the `c_fullname` column is set in the database table, a shorter path format can support ontologies with greater the depth. A heuristic-based limit on ontology depth support is listed in the table below.



-pathFormat options	Number of characters	Support for Ontology Depth	Notes
Short	4	Up to 33 levels	Recommended for ontologies with 100,000 concepts or more
Medium	20	Up to 33 levels	This is the default value if not specified
Readable	32	Up to 20 levels	

The character limit computation is based on a hash algorithm. While careful consideration was made in creating an algorithm that generates the 4-character symbol, it cannot absolutely guarantee a unique `c_fullname` for each term at a given level. Because the Short option only has 4 characters, it is the most likely option to have hash collisions. We therefore recommend those who use Short to query for and manually edit duplicate non-synonymous `c_fullname` entries in your final metadata table using the following SQL statement:

```
select c_fullname, count(1) from YOUR_METADATA_TABLE where c_synonym_cd = 'N' group by c_fullname having count(1) > 1
```

When *ProcessAll* completes, it prints out **Processing complete**. At this point, the output metadata file is ready. It is pipe-delimited (`|`) and has column headings in 1<sup>st</sup> row. Strings are quoted with (`"`).

#### 4. Load Metadata File into Metadata Table

Create a table in your database to host the data in your metadata file. If you are using SQLServer, you may use the following command to create a destination table:

```
CREATE TABLE [DATABASE_NAME].[SCHEMA_NAME].[METADATA_TABLE_NAME]
(
    "c_hlevel" INT NOT NULL,
    "c_fullname" VARCHAR(700) NOT NULL,
    "c_name" VARCHAR(2000) NOT NULL,
    "c_synonym_cd" CHAR(1) NOT NULL,
    "c_visualattributes" CHAR(3) NOT NULL,
    "c_basecode" VARCHAR(50) NULL,
    "c_facttablecolumn" VARCHAR(50) NOT NULL,
    "c_tablename" VARCHAR(50) NOT NULL,
    "c_columnname" VARCHAR(50) NOT NULL,
    "c_columndatatype" VARCHAR(50) NOT NULL,
    "c_operator" VARCHAR(10) NOT NULL,
    "c_dimcode" VARCHAR(700) NOT NULL,
    "c_tooltip" VARCHAR(900) NULL,
    "sourcesystem_cd" VARCHAR(50) NULL,
    "c_symbol" VARCHAR(50) NOT NULL,
    "c_path" VARCHAR(700) NULL,
    "i_snomed_ct" VARCHAR(50) NOT NULL,
    "i_snomed_rt" VARCHAR(50) NOT NULL,
    "i_cui" VARCHAR(50) NOT NULL,
    "i_tui" VARCHAR(50) NOT NULL,
    "i_ctv3" VARCHAR(50) NOT NULL,
    "i_full_id" VARCHAR(100) NOT NULL,
    "update_date" VARCHAR(50) NOT NULL,
    "parent_fullId" VARCHAR(50) NOT NULL,
    "m_applied_path" VARCHAR(900) NOT NULL
);
```

This statement includes all the column in the metadata file. However, some are not necessary for i2b2. You may choose not to include those columns when you create the table and when you perform the data import.

You may use a database-specific importer or a script to programmatically import your metadata file into your metadata table.

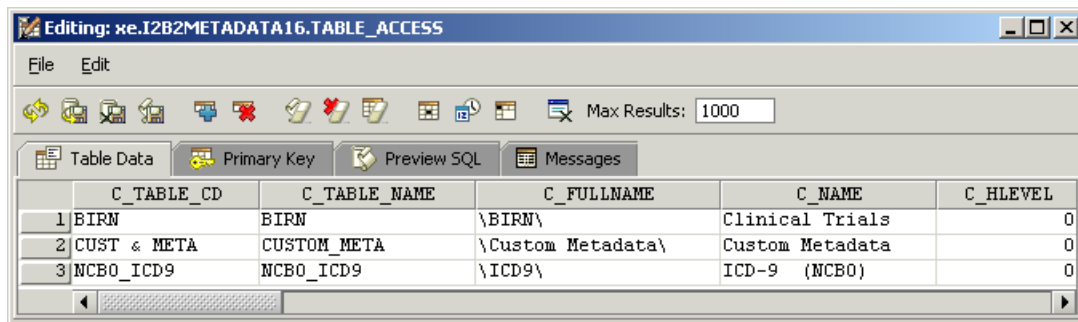
## 5. Helper Scripts for Importation

The "ncbo\_extractor\_2024\_main/scripts/sql\_server" folder contains a `create_table.sql` script that users can use to create a *staging table* or a *metadata table* in SQLServer. It also contains a `Insert_to_metadata_table.py` python script that can be adapted to programmatically parse a *metadata file* and insert its content into a specified SQLServer database table.

## 6. Configuring i2b2 to Use your new Metadata

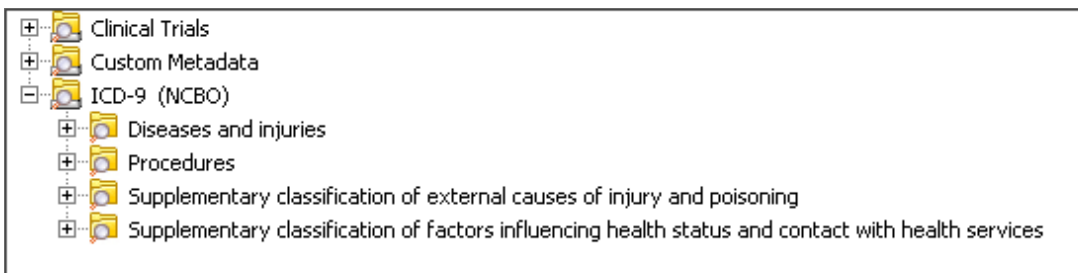
### 6.1 Table\_Access

Reconfigure `table_access` to include the root nodes of your new metadata. If a single root node was specified, it will appear in your final metadata table with `c_hlevel = 0`. If not, root nodes are entries in your final metadata table with `c_hlevel = 1`. There should be one entry per root node. The following example shows a configuration for ICD-9CM. In this example our final target i2b2 metadata table was named 'NCBO\_ICD9', as shown in `c_table_name` column.



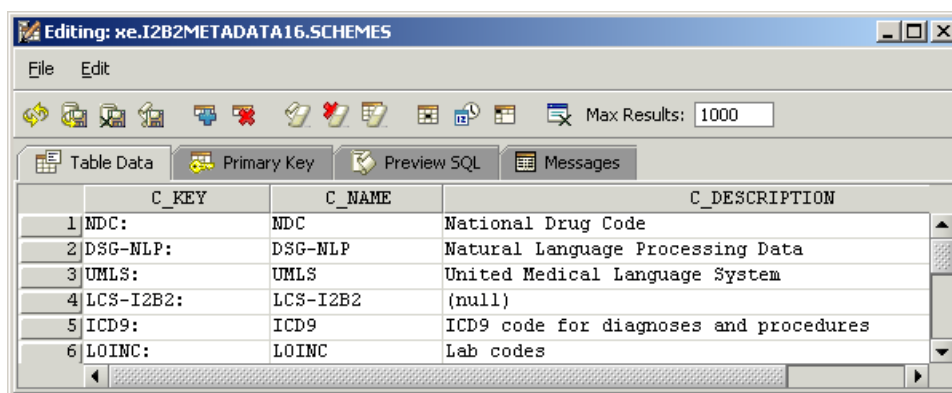
	C_TABLE_CD	C_TABLE_NAME	C_FULLNAME	C_NAME	C_HLEVEL
1	BIRM	BIRM	\BIRM\	Clinical Trials	0
2	CUST & META	CUSTOM_META	\Custom Metadata\	Custom Metadata	0
3	NCBO_ICD9	NCBO_ICD9	\ICD9\	ICD-9 (NCBO)	0

The ontology viewer should show the following structure. Note in this example, we specified a `rootNodeName` of "ICD-9 (NCBO)".



## 6.2 Schemes

Reconfigure the SCHEMES table to include your new ontology's scheme or *prefix*. There should be one entry per scheme. The following example shows a configuration including the ICD9 prefix. This step is only necessary if you want to search ontology concepts by code.



The screenshot shows a software window titled "Editing: xe.I2B2METADATA16.SCHEMES". It has a menu bar with "File" and "Edit", and a toolbar with various icons. Below the toolbar are tabs for "Table Data", "Primary Key", "Preview SQL", and "Messages". The "Table Data" tab is active, displaying a table with three columns: "C\_KEY", "C\_NAME", and "C\_DESCRIPTION". The table contains six rows of data, each representing an ontology scheme.

	C_KEY	C_NAME	C_DESCRIPTION
1	NDC:	NDC	National Drug Code
2	DSG-NLP:	DSG-NLP	Natural Language Processing Data
3	UMLS:	UMLS	United Medical Language System
4	LCS-I2B2:	LCS-I2B2	(null)
5	ICD9:	ICD9	ICD9 code for diagnoses and procedures
6	LOINC:	LOINC	Lab codes

## VI. Software Limitations

1. The NCBO Ontology Extraction Tool only runs on Java 8 or 11. This is due to dependency on older libraries. We will update the tool so that it can run on newer versions of Java.
2. To date, the largest ontology we have extracted contained ~100,000 terms. In this particular case, the resulting metadata file contained 5.7 million entries (terms have synonyms and can appear on several paths). For this reason, we recommend that you exercise caution or avoid extraction of ontologies with more than 100,000 terms.

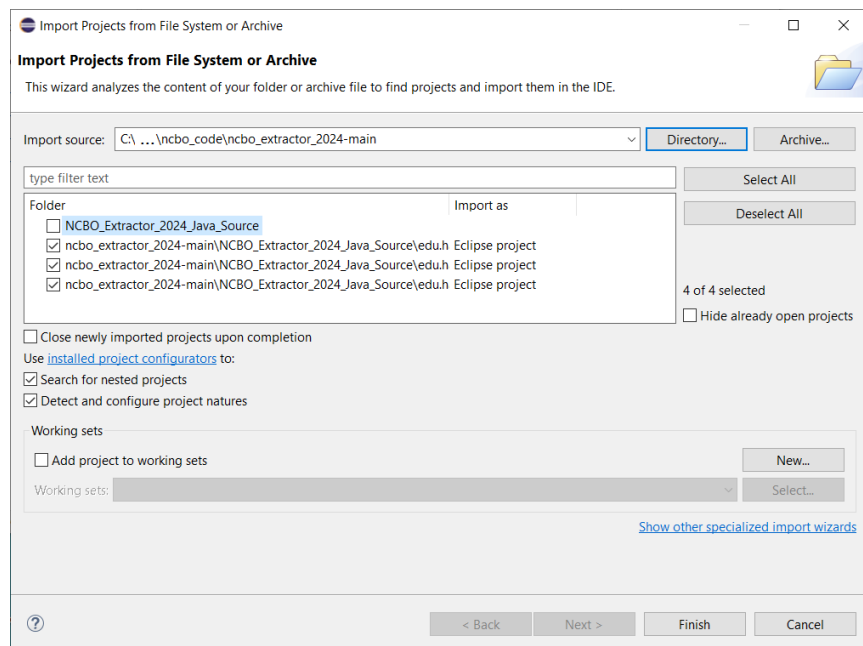
If your desired ontology exceeds this limit of 100,000 terms, it is recommended that you look for or create a bioportal view that contains a subset of the entire ontology and then extract the subset. SNOMED-CT is a prime example of this problem. The metrics page for SNOMED-CT, <https://bioportal.bioontology.org/ontologies/SNOMEDCT>, lists over 375,000 terms for SNOMED-CT. Further down on this same page a set of views (or subsets of SNOMED) are shown.

In addition, when extracting large ontologies, we recommend that you utilize the Short path format (see [Section V.3](#)).

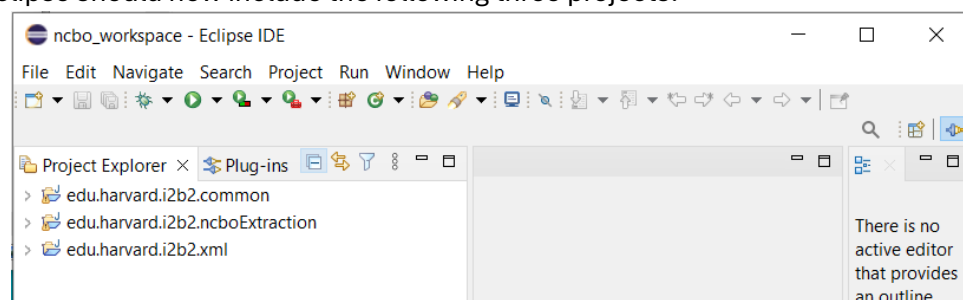
3. The current version of NCBO Extraction Tool has been mainly tested with SQLServer. Scripts for creating tables and importing files are SQLServer-specific. Better support for Oracle and Postgres is currently being worked on.

## VII. Developer Support

1. Developers can follow [Section III](#) to download the NCBO Extraction Tool from GitHub. Developers with a GitHub account can also **clone** or **fork** the repository.
2. The downloaded code is set up to be easily imported into [Eclipse](#), the Integrated Development Environment (IDE) for Java. The following outline the import and the setup process in Eclipse.
3. Let us assume the downloaded zip file is unzipped into the ncbo\_extractor\_2024\_main folder
4. In your Eclipse, click **File -> Open Projects from File System...** to bring up the **Import Projects from File System or Archive** popup
5. In the popup, click **Directory...** and navigate to your ncbo\_extractor\_2024\_main folder. **Deselect** the NCBO\_Extractor\_2024\_Java\_Source folder as shown in the following screen. Then select **Finish** to initiate the import.



6. Your Eclipse should now include the following three projects:



## 1. Project Description

The `common` project produces the `i2b2Common-core.jar`, which is a dependency for the `ncboExtraction` project. Both the `ExtractAll` and the `ProcessAll` commands are found in the main project:

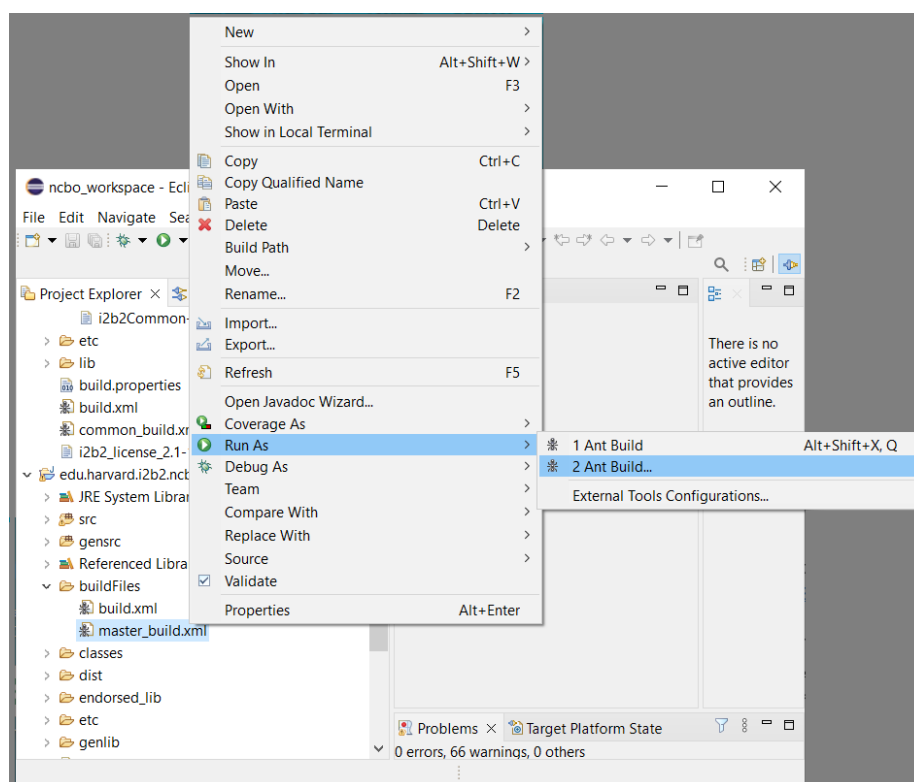
```
edu.harvard.edu.i2b2.ncbo.extraction.NCBOOntologyExtractAll.java
edu.harvard.edu.i2b2.ncbo.extraction.NCBOOntologyProcessAll.java
```

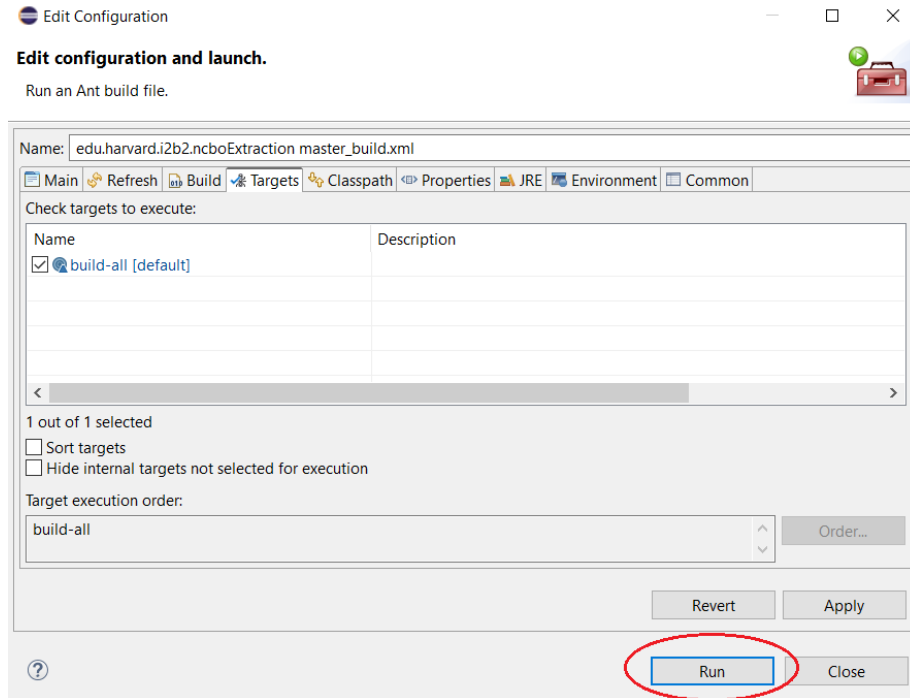
The projects are fully pre-compiled using JDK 8. Developers can right-click the two java files above to run. The `common` project generates `i2b2Common-core.jar`, and the `ncboExtraction` project generates final `NCBOExtraction_[version].jar` file run the NCBO Extraction commands (see [Section IV](#)).

## 2. Compilation in Eclipse

To compile Extractor in Eclipse , right click on

`edu.harvard.i2b2.ncboExtraction/buildFiles/master_build.xml` and select **Run as... -> Ant Build... -> Run**.





The master\_build.xml build file first compiles the source code in edu.harvard.i2b2.common to produce i2b2Common-core.jar and then compiles the sources files in edu.harvard.i2b2.ncboExtraction. After compilation, all newly compiled Java bytecode files should be in their respective .../classes folders. Newly produced i2b2Common-core.jar are placed in edu.harvard.i2b2.common/dist, edu.harvard.i2b2.ncboExtraction/genlib, and edu.harvard.i2b2.ncboExtraction/dist/genlib. Newly compiled NCBOExtraction\_[version].jar file should also be in edu.harvard.i2b2.ncboExtraction/dist.

### 3. build.xml Compilation Options

The build.xml files in the edu.harvard.i2b2.common and edu.harvard.i2b2.ncboExtraction projects contain the following line in the **COMPILE** section to invoke the Java compiler to perform the compilation. This is the line used to compile the projects before release. This line specifies Java compiler 1.8 (aka Java compiler 8) instead of the default compiler in Eclipse.

```
<javac destdir="${classes}" optimize="${javac.opt}" includeantruntime="false" debug="${javac.debug}"
source="1.8" target="1.8" compiler="javac1.8">
```

Developers need to download [JDK 8 from Oracle](#). A free Oracle account is required. This results will be runnable in JRE 8. However, developer must select a JRE to run the ANT script (Section VII 2).

Alternatively, developers can download [JDK 11](#) and configure the ANT Script to use this line (currently commented out in the build.xml files) instead of the one above. The result will be runnable in JRE 11.

```
<javac destdir="${classes}" optimize="${javac.opt}" includeantruntime="false" debug="${javac.debug}"
source="11" target="11" compiler="javac10+">>
```

Should the above lines not specific enough, developers use the following syntax to as specific as possible. Please adapt it to your operating system and Java compiler version:

```
<javac destdir="${classes}" optimize="${javac.opt}" includeantruntime="false" debug="${javac.debug}"
source="1.8" target="1.8" bootclasspath="C:\PATH_TO\Java\jdk-1.8\lib\rt.jar"
executable="C:\PATH_TO\Java\jdk-1.8\bin\javac" compiler="javac1.8">
```

#### 4. The `edu.harvard.i2b2.xml` Project

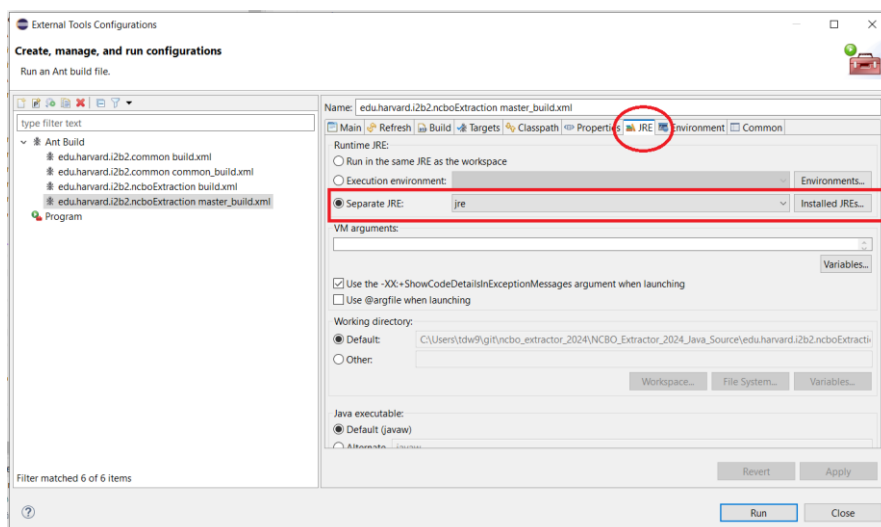
Please note that the `edu.harvard.i2b2.xml` project is not used in the default `master_build.xml`. This project is used to generate the source files in the `edu.harvard.i2b2.ncboExtraction/gensrc` folder. For most development, these source files need not to be re-generated. However, if developers want to change the `xsd` files (REST API message definitions) and therefore requiring regeneration of the corresponding source files in `edu.harvard.i2b2.ncboExtraction/gensrc`, it must be done via JAXB in Java 8. In the [latest i2b2 release note](#), it stipulates that if JAXB is used to gen src due to `xsd` changes, developers must

1. Run gensrc using Java 8
2. Copy the generated src into the `/gensrc` folder
3. Then run ANT script using Java 11 without deleting src files in `/gensrc`

#### 5. Changing JRE for Running ANT Build Scripts in Eclipse

The Eclipse projects are set to compilation compliance level to “JavaSE-1.8” (a.k.a. Java 8). However, recent Eclipse releases have barred the running of ANT build scripts using Java version earlier than 11. To configure Eclipse to use Java 11 or later to run ANT, follow the following steps:

1. Right-click `master_build.xml`
2. **Run as...**
3. **External Tool Configurations...**
4. Select the **JRE** tab and select **Separate JRE** and **Installed JREs...** to choose a JRE with version greater than 8. The JRE that comes with Eclipse JRE work.



## 5. Run

This sets up ANT script to be run on the JRE you designate. You can download different versions of JRE from Oracle (see [Section II.3](#)).