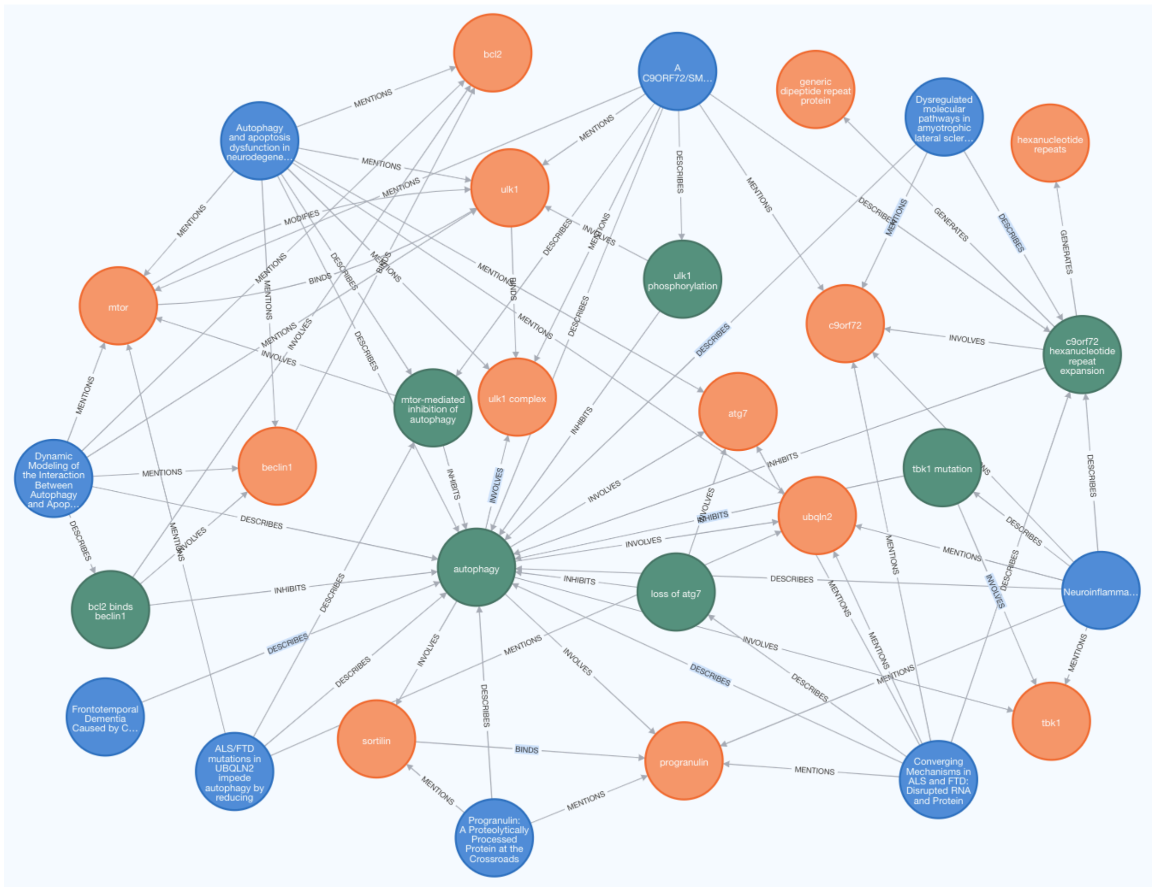
# s c r i b l

A system for the semantic capture of relationships in biological literature

Amber Biology LLC

Version 0.8.0, July 2024


bcl2
A C9ORF72/SM...
generic dipeptide repeat protein
Dysregulated molecular pathways in amyotrophic lateral scler...
hexanucleotide repeats
Autophagy and apoptosis dysfunction in neurodegene...
ulk1
ulk1 phosphorylation
mtor
c9orf72
c9orf72 hexanucleotide repeat expansion
mtor-mediated inhibition of autophagy
ulk1 complex
atg7
beclin1
Dynamic Modeling of the Interaction Between Autophagy and Apop...
tbk1 mutation
ubqln2
autophagy
loss of atg7
bcl2 binds beclin1
Neuroinflamma...
Frontotemporal Dementia Caused by C...
ALS/FTD mutations in UBQLN2 impede autophagy by reducing
sortilin
progranulin
Converging Mechanisms in ALS and FTD: Disrupted RNA and Protein
tbk1
Progranulin: A Proteolytically Processed Protein at the Crossroads
MENTIONS
DESCRIBES
INVOLVES
INHIBITS
BINDS
GENERATES
MODIFIES

## acknowledgments

The development of the **scribl** platform was made possible by the funding and expertise provided by the Association for Frontotemporal Degeneration (AFTD - https://www.theaftd.org/) whose mission is to improve the quality of life of people affected by Frontotemporal Degeneration and drive research to a cure. We are particularly grateful to AFTD leadership team members Debra Niehoff and Penny Dacks for their invaluable direction and guidance, and for getting us access to leading researchers in the field of neurodegenerative disease - all of whose insights and advice were pivotal in the development of this free, open-source research tool.

## introduction

The digital capture of research articles and their attendant metadata in a database is an excellent way to create a searchable catalog of scientific literature. In such a database however, nearly all of the semantic detail contained in the curated articles is lacking. A typical literature search based upon text and keywords can be a blunt instrument, often generating large sets of articles of potential interest that still need to be read more closely in order to determine if the processes or phenomena that they describe are actually relevant to the user. In life science research in particular, a scientific literature database is most immediately useful to a scientist looking for general literature on a broad research area, or conversely, to a scientist who has very precise idea of what they are searching for - perhaps even down to the title of a specific article, the name of a specific gene, or the name of an author who is working on a research problem that is directly relevant to their own interests. In between these two extremes is the scientist whose research is focused on a relatively narrow scientific area, the results of which require a broader, more holistic perspective to make sense of. This latter category arguably encompasses the great majority of life science researchers who find themselves working on a relatively small set of biological processes and entities that are themselves components of a much larger system to which they contribute, and whose properties and behaviors are largely defined by the context of this larger system in which they are embedded.

As a concrete use case for this situation, consider a researcher whose work focuses upon a narrow set of genes and proteins that are implicated in a neurodegenerative disease via some set of processes and mechanisms that have yet to be elucidated. In a laboratory cell assay, this researcher observes that the death of cultured cells is clearly accompanied by some dysfunction in the clearance from these cells of aggregated and misfolded proteins. Based upon this observation and pursuing a hypothesis that a breakdown in autophagy may play a role in this disease, the researcher starts a PubMed[1] search using the term “autophagy” which at the time of writing yields 65,638 articles[2]. This number could certainly be reduced by the inclusion of some additional terms, perhaps even including the names of the genes and proteins that are the focus of the researcher’s experiments. But in the search for mechanistic insights that might link their own data with their disease of interest in a manner that also accounts for the role of autophagy, the researcher may well need to wade through a great deal of reading material in search of the precise pieces that can make sense of this puzzle.

As biology has become ever more quantitative with the advent of lab automation and digital data capture, the rate of generation of scientific data has significantly outstripped our ability to generate real knowledge and insight from that data. As a case in point, the differential expression of a panel of genes between a healthy and a distressed cell might initially appear to be a set of unconnected data points from which it is challenging to draw any conclusions, until the *relationships* between the differentially expressed genes are made clear by mapping them

[1] https://pubmed.ncbi.nlm.nih.gov/
[2] https://pubmed.ncbi.nlm.nih.gov/?term=autophagy

onto a specific cell signaling pathway. It is these relationships between the many components of complex biological systems and between the processes that they participate in, that are central to the properties and behaviors of these systems to which we would apply the term “biology”.

Beyond merely listing the names of the agents (proteins, genes, compounds, receptor complexes etc. etc.) and processes underlying the biology, it would therefore be extremely valuable for the researcher to be able to query a scientific literature database based also upon the relationships between these agents and processes. It is to this end that the simple **scribl** syntax was developed to allow for the curation of these relationships along with the articles that describe them - relationships that would otherwise be buried in the texts of those articles.

To be clear, the biological relationships captured in **scribl** are not intended to replace a reading of the literature but rather to further narrow down the search for relevant biological agents and processes. Nor is **scribl** intended to provide a semantic platform for the construction of formal models of biological process and interacting agents such as the modeling languages Kappa[3] and SBML[4]. The development of such models requires a much narrower focus on a specific set of agents and reactions, and is significantly constrained by the requirement for a level of specific, quantitative detail that most life science articles do not provide. In this sense then, **scribl** could be considered a platform for the development of a kind of coarse-grained model of biological systems that sits somewhere between the very low resolution representation of a system by keywords and literature terms, and the very high resolution representation of a formal, kinetic model. It is worth noting that at the time of writing, even though a biological system might be comprehensively described in great detail by a particular scientific article, the kind of natural language processing[5] technology required to extract an accurate, formal model of that system from the article text has yet to reach the level at which this could realistically be achieved algorithmically.

Nor is **scribl** intended to be a replacement for biological graph databases such as Reactome (https://reactome.org). The Reactome database is actually based upon the same graph database engine that is the current default supported by **scribl**, so **scribl** could actually serve as a useful aid to facilitate the curation of biological pathways from newly-published literature, in a format that is ready for graph data repositories like Reactome. It is also often desirable for researchers to curate literature databases that are focused on narrower, more specific areas of interest, including those that may also be poorly represented in the larger data repositories.

[3] https://kappalanguage.org/
[4] https://synonym.caltech.edu/
[5] https://en.wikipedia.org/wiki/Natural_language_processing

## **scribl**'s graph database approach

The primary purpose of **scribl** is the curation from scientific articles, of the relationships between the various biological agents and processes that they describe, with a view to generating a graph database that captures these relationships as a connected network. In contrast with a traditional relational database[6] with tables of rows and columns, a graph database[7] captures data as a collection of nodes connected by edges that define the relationships between the nodes. The easiest way to appreciate how a graph database works and the advantages that it has over a traditional, relational database, is to see one in action. Shown in Figure 2 below, is a tiny portion of a scientific literature database for a disease that was developed using **scribl**.

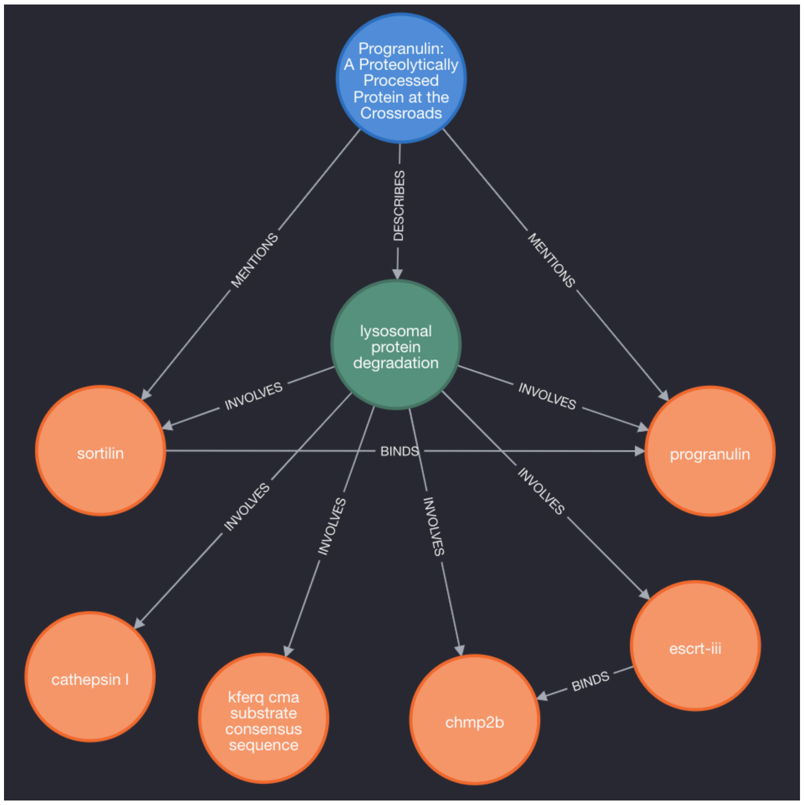

Progranulin: A Proteolytically Processed Protein at the Crossroads
MENTIONS
DESCRIBES
MENTIONS
lysosomal protein degradation
INVOLVES
INVOLVES
sortilin
BINDS
progranulin
INVOLVES
INVOLVES
INVOLVES
INVOLVES
escrt-iii
cathepsin l
kferq cma substrate consensus sequence
chmp2b
BINDS

Figure 2.

In this graph database of articles about neurodegenerative disease, an **article** (shown in blue) entitled “Progranulin: A Proteolytically Processed Protein at the Crossroads of Inflammation and Neurodegeneration” **describes** the **process** (shown in green) “lysosomal protein degradation”, and **mentions** 2 of the **agents** (shown in orange) “sortilin” and “progranulin” that are **involved** in this **process**. The **agents** are the “players” in the biological processes - typically genes and

[6] https://en.wikipedia.org/wiki/Relational_database
[7] https://towardsdatascience.com/an-introduction-to-graph-databases-cd81a0d5aa12

proteins, but also any biochemical entity ranging from a simple ion to a large, macromolecular complex. It can also be seen in the graph, that the agent “sortilin” **binds** the agent “progranulin. The process “lysosomal protein degradation” **involves** 4 other agents that are not **mentioned** in this article and here we can see one of the advantages of a graph database. These other 4 agents were curated from other articles that also **describe** the process “lysosomal protein degradation”, but from a different perspective. Effectively then, the graph database is connecting the information from different articles that describe the same process, as well as showing the relationships between them. With this graph database it is possible to do searches of the kind “Show me all of the agents that are involved in the process ‘lysosomal degradation’ and the articles that describe them”.

The graph itself can also be expanded beyond the bounds defined by the initial search. For example, in Figure 3 below, we have expanded the graph to include the immediate neighbors of the agent “cathepsin l”. We can see that this agent is also mentioned in 2 other articles, and that the agent itself is generated by the process “procathepsin l processing”. This graph expansion can be repeated by expanding successive nodes as needed. In the graph below for example, we might be interested in further expanding the process “procathepsin l processing” as a potential line of research enquiry.

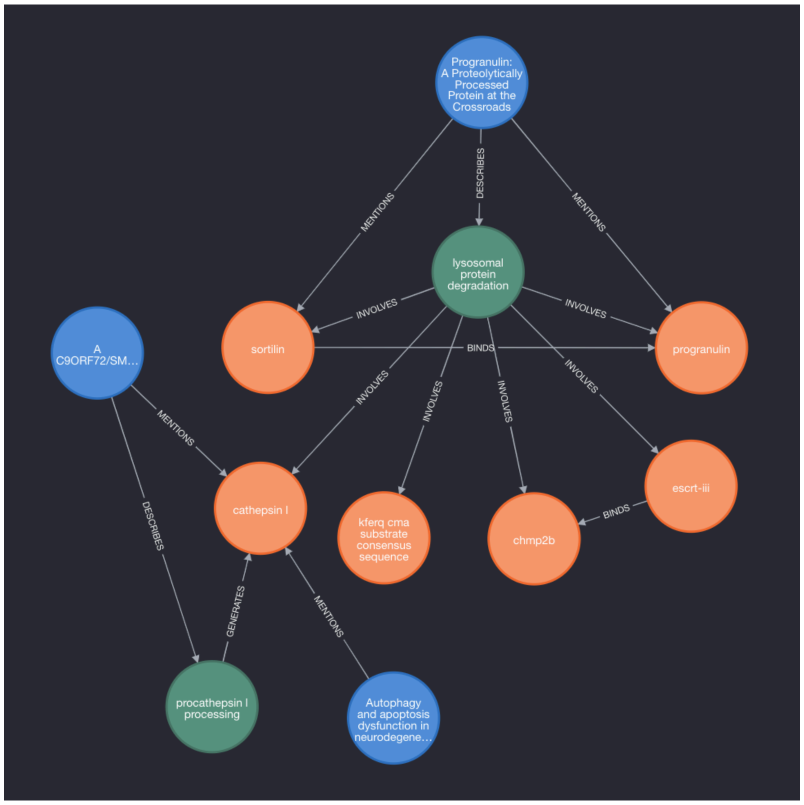

Progranulin: A Proteolytically Processed Protein at the Crossroads
DESCRIBES
MENTIONS
MENTIONS
lysosomal protein degradation
INVOLVES
INVOLVES
sortilin
BINDS
progranulin
A C9ORF72/SM…
INVOLVES
INVOLVES
INVOLVES
INVOLVES
MENTIONS
escrt-iii
cathepsin l
kferq cma substrate consensus sequence
chmp2b
BINDS
DESCRIBES
GENERATES
MENTIONS
procathepsin l processing
Autophagy and apoptosis dysfunction in neurodegene…

Figure 3.

The graph also reveals some of the **causal relationships** between biological processes and agents. In Figure 3 we can see that the process “procathepsin l processing” impacts the process “lysosomal protein degradation” since it generates one of the agents that is involved in that process. It is also possible to move away from the literature database paradigm completely, and explore the documented relationships between biological processes and agents without reference to the articles that describe or mention them. In Figure 4 below, we can trace one of the causal relationships between mutations in the **c9orf72** gene and the dysfunction in nucleocytoplasmic transport that result from them. The hexanucleotide expansion of the **c9orf72** gene generates hexanucleotide repeats that sequester the ran-GTPase-activating protein **rangap1**. The protein **rangap1** facilitates the maintenance of a gradient of **ran**-GTP and **ran**-GDP species between the nucleus and the cytoplasm, a gradient that is essential for nucleocytoplasmic transport. Through this graph whose relationships span multiple articles and connect the information in them, we can see one of the ways that mutations in the **c9orf72** gene can result in nuclear transport dysfunction. It also illustrates that an agent can be as specific as a particular gene or protein, or as generic as an undefined, repeating sequence pattern in a gene. Although no articles are shown in Figure 4, we can of course, expand any node in the graph - if for example, we also wish to retrieve the articles that discuss a particular process or agent.

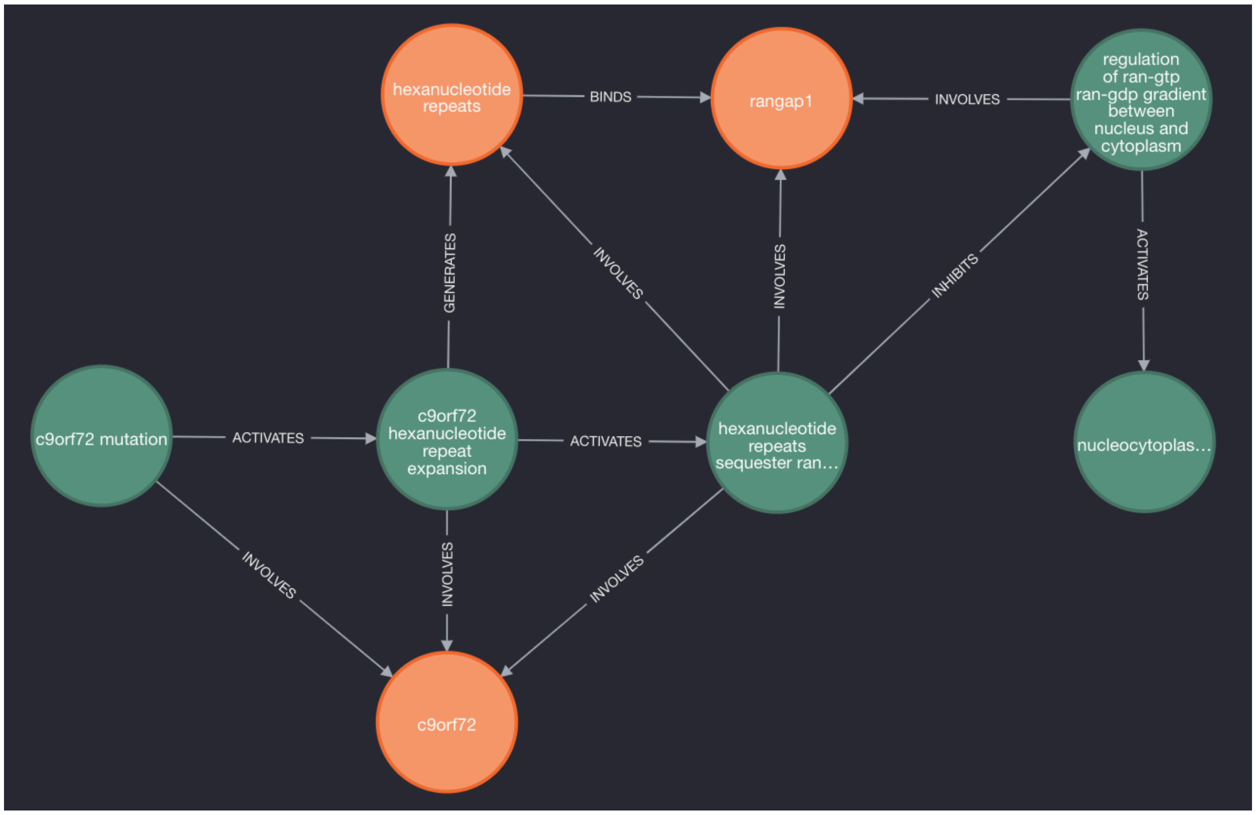
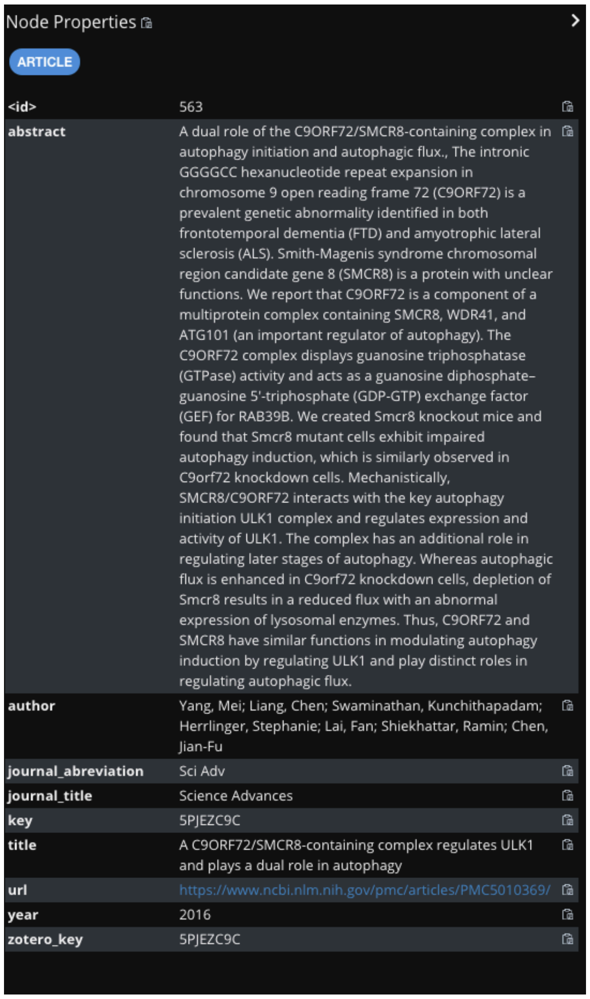


Figure 4.

Both nodes and edges in the graph can be assigned a rich variety of metadata that facilitates more focused searches. The article nodes for example, contain all of the essential fields that would be expected in a scientific literature database - title, abstract, journal title, authors, year of publication etc. etc. and these metadata are almost infinitely customizable in a way that would be very laborious in a traditional, relational database in which the entire database schema would need to be changed in order to introduce a new data field. In Figure 5 below, you can see the current metadata for the database article that was used to retrieve some of the relationships shown in Figure 4 above.



Node Properties

ARTICLE

| | |
|---|---|
| <id> | 563 |
| abstract | A dual role of the C9ORF72/SMCR8-containing complex in autophagy initiation and autophagic flux., The intronic GGGGCC hexanucleotide repeat expansion in chromosome 9 open reading frame 72 (C9ORF72) is a prevalent genetic abnormality identified in both frontotemporal dementia (FTD) and amyotrophic lateral sclerosis (ALS). Smith-Magenis syndrome chromosomal region candidate gene 8 (SMCR8) is a protein with unclear functions. We report that C9ORF72 is a component of a multiprotein complex containing SMCR8, WDR41, and ATG101 (an important regulator of autophagy). The C9ORF72 complex displays guanosine triphosphatase (GTPase) activity and acts as a guanosine diphosphate–guanosine 5'-triphosphate (GDP-GTP) exchange factor (GEF) for RAB39B. We created Smcr8 knockout mice and found that Smcr8 mutant cells exhibit impaired autophagy induction, which is similarly observed in C9orf72 knockdown cells. Mechanistically, SMCR8/C9ORF72 interacts with the key autophagy initiation ULK1 complex and regulates expression and activity of ULK1. The complex has an additional role in regulating later stages of autophagy. Whereas autophagic flux is enhanced in C9orf72 knockdown cells, depletion of Smcr8 results in a reduced flux with an abnormal expression of lysosomal enzymes. Thus, C9ORF72 and SMCR8 have similar functions in modulating autophagy induction by regulating ULK1 and play distinct roles in regulating autophagic flux. |
| author | Yang, Mei; Liang, Chen; Swaminathan, Kunchithapadam; Herrlinger, Stephanie; Lai, Fan; Shiekhattar, Ramin; Chen, Jian-Fu |
| journal_abreviation | Sci Adv |
| journal_title | Science Advances |
| key | 5PJEZC9C |
| title | A C9ORF72/SMCR8-containing complex regulates ULK1 and plays a dual role in autophagy |
| url | https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5010369/ |
| year | 2016 |
| zotero_key | 5PJEZC9C |

Figure 5.

This metadata also allows for all of the search queries that one would make using a traditional (relational) literature database - searching by title, keyword, author, year etc. etc. The big difference with the graph database however, is that the results generated by these queries can subsequently be expanded to show the relationships captured in the articles, as well as the connections and overlapping areas between the contents of the articles.

## integration with open-source literature tools

The **scribl** platform was developed with the intention of creating an open-source tool that would be freely available to all researchers. In keeping with this goal, the excellent, open-source literature database tool **Zotero** (https://www.zotero.org) was chosen for the integration of **scribl** with existing literature database platforms - although in theory there is no reason why **scribl** could not be used with any literature database platform (including proprietary platforms) that supports the manual addition of metadata. **Zotero** was also chosen because it natively supports the creation of collaborative, cloud-based databases that facilitate the global sharing and collaborative curation of data between researchers, via the web.

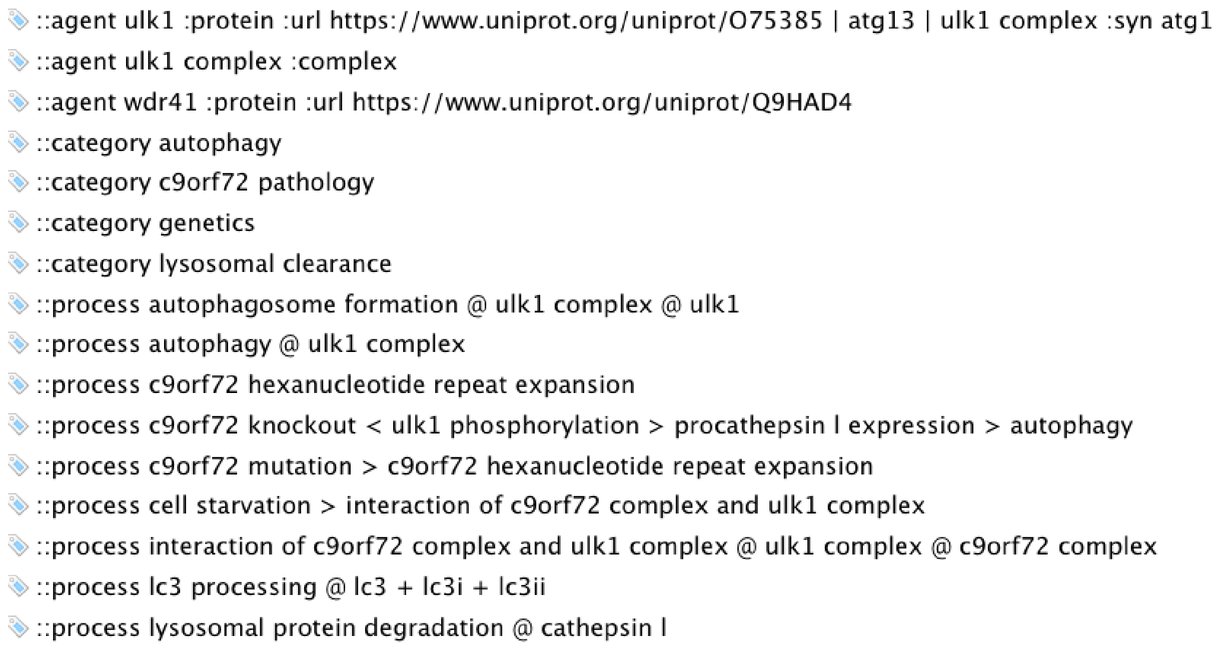
The basis of **scribl** is a very simple syntax for tagging articles with metadata that can be processed to generate a graph database using a data file exported from the **Zotero** relational database. No direct access to **Zotero's** SQLite[8] database engine or application programming interface (API) is required since **scribl** is able to generate a graph database entirely from an exported file. It is also possible to access the **Zotero** database programmatically via the Zotero API and so **scribl** can skip the file export step and access the Zotero database directly The addition of some extra manually-curated metadata to the entries in the **Zotero** database does not in any way impair the normal functionality of a **Zotero** database and **scribl** can (and should) be used in conjunction with, and as a complement to, an existing **Zotero** database.

In the current version of **scribl**, each **scribl** statement is a tag consisting of a simple line of text that is added to the manual tags that a **Zotero** user can attach to each article entry in the database. The relationships between the article itself and the biological processes and agents that it discusses are generated automatically by the addition of **scribl** statements to the manual tags field of a particular article in the database, and do not need to be explicitly defined by the user.

The structure and syntax of **scribl** will be discussed in much more detail in subsequent sections, but in Figure 6 below, you can see what **scribl** statements look like when they have been added to the manual tags field in a **Zotero** article. Following the access to the **Zotero** database (either via a file in CSV[9] format, or by direct API access), the **scribl** statements from all articles in the database are compiled and integrated into an internal graph database format in **Python**[10] that represents all of the literature in the **Zotero** database. After compiling this graph-formatted data structure, the next step is to export this data in a form that can be recognized by one of the currently available graph database platforms. The current version of **scribl** can convertthe graph-formatted data into both the **cypher query language**[11] that is used

[8] https://en.wikipedia.org/wiki/SQLite
[9] https://en.wikipedia.org/wiki/Comma-separated_values
[10] https://www.python.org/
[11] https://neo4j.com/developer/cypher/

for creating, editing and querying the open-source graph database **neo4j**[12], (discussed further in the next section), as well as **GraphML**[13], which can be read by **NetworkX**[14].



```
::agent ulk1 :protein :url https://www.uniprot.org/uniprot/O75385 | atg13 | ulk1 complex :syn atg1
::agent ulk1 complex :complex
::agent wdr41 :protein :url https://www.uniprot.org/uniprot/Q9HAD4
::category autophagy
::category c9orf72 pathology
::category genetics
::category lysosomal clearance
::process autophagosome formation @ ulk1 complex @ ulk1
::process autophagy @ ulk1 complex
::process c9orf72 hexanucleotide repeat expansion
::process c9orf72 knockout < ulk1 phosphorylation > procathepsin l expression > autophagy
::process c9orf72 mutation > c9orf72 hexanucleotide repeat expansion
::process cell starvation > interaction of c9orf72 complex and ulk1 complex
::process interaction of c9orf72 complex and ulk1 complex @ ulk1 complex @ c9orf72 complex
::process lc3 processing @ lc3 + lc3i + lc3ii
::process lysosomal protein degradation @ cathepsin l
```

Figure 6.

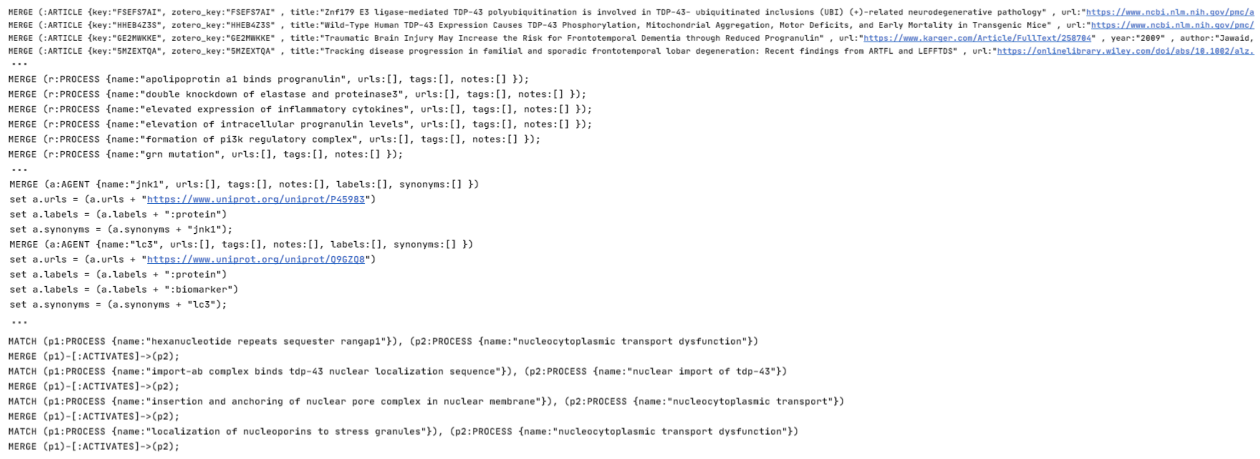
[12] https://neo4j.com/
[13] http://graphml.graphdrawing.org/
[14] https://networkx.org/

## integration with open-source graph databases

In keeping with the requirement that **scribl** and the tools it integrates with are freely available and open-source, we chose the free, open-source graph database **neo4j**[12] as our default graph database engine for **scribl**. As is the case with **Zotero**, **scribl** has a loosely coupled integration with **neo4j** insofar as it does not currently require access to the **neo4j** API or internals. In the current version of **scribl**, the internal graph database generated from **Zotero** data can be exported in cypher[11], which is the language used in **neo4j** for creating, editing and querying the graph database. Although **neo4j** is the only database that this first iteration of **scribl** currently supports, there is no reason why it would not be able to support multiple graph databases via extension of the platform code to enable graph data exports in other formats.

Once the internal graph database has been compiled from data exported from **Zotero**, it can be used to generate a series of cypher statements that can be input directly to **neo4j** (the workflow for creating and updating a **neo4j** database using **scribl**, will be covered in subsequent sections). The exported cypher text encodes all of the articles in the **Zotero** database, including their associated metadata (author, journal, year etc.), along with the agents and processes described by the articles and their relationships. The relationships between the articles and the processes and agents they describe are compiled automatically and do not need to be explicitly encoded by the user. The **scribl** syntax also allows for the categorization of articles in the database under major subheadings, and for the capture of relevant online resources such as other databases that might be referenced in the articles.

The exported cypher text is human-readable text that can be input directly into **neo4j**, Some examples of cypher statements are shown in Figure 7 below (very long statements are shown truncated for improved readability).



```
MERGE (:ARTICLE {key:"FSEFS7AI", zotero_key:"FSEFS7AI" , title:"Znf179 E3 ligase-mediated TDP-43 polyubiquitination is involved in TDP-43- ubiquitinated inclusions (UBI) (+)-related neurodegenerative pathology" , url:"https://www.ncbi.nlm.nih.gov/pmc/a
MERGE (:ARTICLE {key:"HHEB4Z3S", zotero_key:"HHEB4Z3S" , title:"Wild-Type Human TDP-43 Expression Causes TDP-43 Phosphorylation, Mitochondrial Aggregation, Motor Deficits, and Early Mortality in Transgenic Mice" , url:"https://www.ncbi.nlm.nih.gov/pmc/
MERGE (:ARTICLE {key:"GE2MWKKE", zotero_key:"GE2MWKKE" , title:"Traumatic Brain Injury May Increase the Risk for Frontotemporal Dementia through Reduced Progranulin" , url:"https://www.karger.com/Article/FullText/258704" , year:"2009" , author:"Jawaid,
MERGE (:ARTICLE {key:"5MZEXTQA", zotero_key:"5MZEXTQA" , title:"Tracking disease progression in familial and sporadic frontotemporal lobar degeneration: Recent findings from ARTFL and LEFFTDS" , url:"https://onlinelibrary.wiley.com/doi/abs/10.1002/alz.
 ...
MERGE (r:PROCESS {name:"apolipoprotin a1 binds progranulin", urls:[], tags:[], notes:[] });
MERGE (r:PROCESS {name:"double knockdown of elastase and proteinase3", urls:[], tags:[], notes:[] });
MERGE (r:PROCESS {name:"elevated expression of inflammatory cytokines", urls:[], tags:[], notes:[] });
MERGE (r:PROCESS {name:"elevation of intracellular progranulin levels", urls:[], tags:[], notes:[] });
MERGE (r:PROCESS {name:"formation of pi3k regulatory complex", urls:[], tags:[], notes:[] });
MERGE (r:PROCESS {name:"grn mutation", urls:[], tags:[], notes:[] });
 ...
MERGE (a:AGENT {name:"jnk1", urls:[], tags:[], notes:[], labels:[], synonyms:[] })
set a.urls = (a.urls + "https://www.uniprot.org/uniprot/P45983")
set a.labels = (a.labels + ":protein")
set a.synonyms = (a.synonyms + "jnk1");
MERGE (a:AGENT {name:"lc3", urls:[], tags:[], notes:[], labels:[], synonyms:[] })
set a.urls = (a.urls + "https://www.uniprot.org/uniprot/Q9GZQ8")
set a.labels = (a.labels + ":protein")
set a.labels = (a.labels + ":biomarker")
set a.synonyms = (a.synonyms + "lc3");
 ...
MATCH (p1:PROCESS {name:"hexanucleotide repeats sequester rangap1"}), (p2:PROCESS {name:"nucleocytoplasmic transport dysfunction"})
MERGE (p1)-[:ACTIVATES]->(p2);
MATCH (p1:PROCESS {name:"import-ab complex binds tdp-43 nuclear localization sequence"}), (p2:PROCESS {name:"nuclear import of tdp-43"})
MERGE (p1)-[:ACTIVATES]->(p2);
MATCH (p1:PROCESS {name:"insertion and anchoring of nuclear pore complex in nuclear membrane"}), (p2:PROCESS {name:"nucleocytoplasmic transport"})
MERGE (p1)-[:ACTIVATES]->(p2);
MATCH (p1:PROCESS {name:"localization of nucleoporins to stress granules"}), (p2:PROCESS {name:"nucleocytoplasmic transport dysfunction"})
MERGE (p1)-[:ACTIVATES]->(p2);
```

Figure 7.

## the **scribl** schema

The **scribl** schema comprises a hierarchy of the 5 basic entities, **article**, **category**, **resource**, **process**, and **agent**. The valid relationships for each entity are shown in Table 1 below and schematically in Figure 8 that follows it (details of the **scribl** syntax will be discussed later)

| entity | relationship | entity |
|---|---|---|
| **article** | RELATES | **category** |
| **article** | REFERENCES | **resource** |
| **article**<br>**resource** | DESCRIBES | **process** |
| **article**<br>**resource** | MENTIONS | **agent** |
| **process** | ACTIVATES<br>INHIBITS<br>REGULATES | **process** |
| **process** | INVOLVES<br>GENERATES<br>REMOVES | **agent** |
| **agent** | BINDS<br>MODIFIES | **agent** |

Table 1.

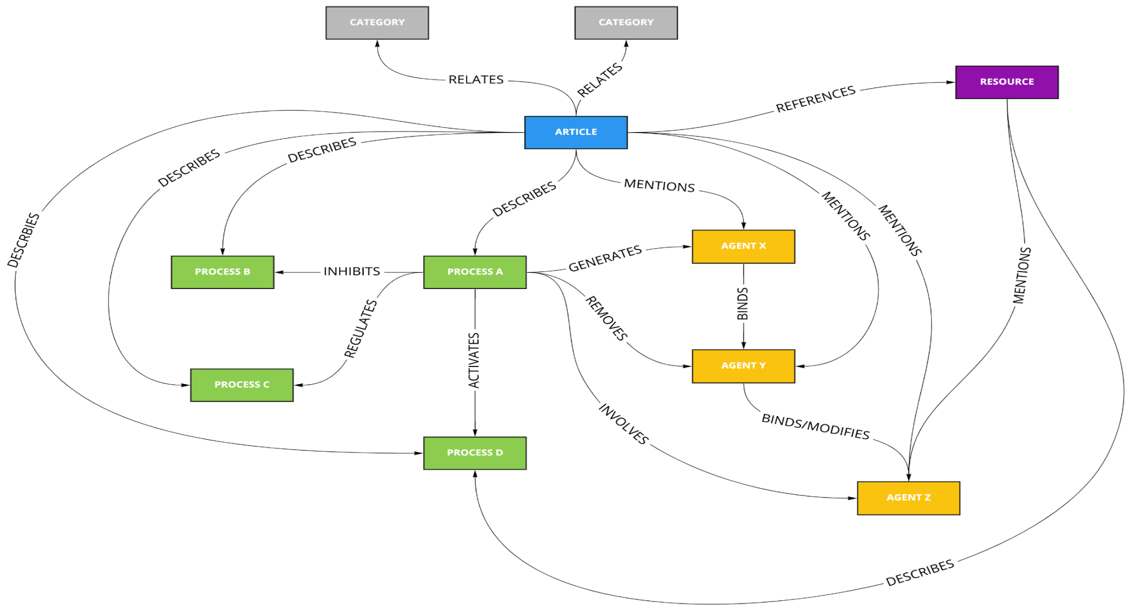


Figure 8.

## article RELATES category

Articles can be categorized into subtopics to make searches easier. An individual article can belong to multiple categories. The **RELATES** relationship is compiled automatically for each article based upon the **category** statements that appear in its tags.

## article REFERENCES resource

Articles often reference an online resource such as a database or a scientific data repository. An individual article can reference multiple resources. The **REFERENCES** relationship is compiled automatically for each article based upon the **resource** statements that appear in its tags.

## article DESCRIBES process

Articles describe biological processes. An individual article typically describes many processes. The **DESCRIBES** relationship is compiled automatically for each article based upon the **process** statements that appear in its tags. A process can be as specific as the expression of a particular gene, or as generic as a general cellular activity such as nuclear export.

## resource DESCRIBES process

Like articles, resources also describe biological processes and an individual resource can describe multiple processes. Unlike articles however, the **DESCRIBES** relationship is not compiled automatically for each **resource** that appears in an article and must be tagged within the resource statement itself.

## article MENTIONS agent

Articles mention agents and an individual article typically mentions many agents. As previously stated, agents are the “players” in biological processes, and an agent can be any biochemical entity from a simple calcium ion to a multi-protein receptor complex or even an organelle like a mitochondrion. The **MENTIONS** relationship is compiled automatically for each article based upon the **agent** statements that appear in its tags.

## resource MENTIONS agent

Like articles, resources also mention agents and an individual resource can mention multiple agents. Unlike articles however, the **MENTIONS** relationship is not compiled automatically for each **resource** that appears in an article and must be tagged within the resource statement itself.

## process ACTIVATES/INHIBITS/REGULATES process

A process can impact the activity of other processes in 3 ways; activating another process, inhibiting another process, or regulating another process. **ACTIVATES** spans the gamut from simply enabling another process to occur (as a prerequisite for it) to accelerating or enhancing the activity of the other process. **INHIBITS** means reducing or ablating the activity of the other process. When one process **REGULATES** another, it is capable of activating **or** inhibiting the activity of the other process, dependent upon its own activity/state. These 3 relationships can be declared in any **process** statement.

## process INVOLVES agent

If an agent plays a role in a particular process, it can be tagged using an **INVOLVES** relationship in a **process** statement. The agent does not need to be essential for the process, it only needs to affect its activity.

## process GENERATES/REMOVES agent

Agents can be generated or removed by processes. For example, the **process** of gene expression **GENERATES** an **agent** - the RNA transcript of that gene; the **process** of lysosomal degradation of a particular protein **REMOVES** an **agent** - the degraded protein. **REMOVES** spans the gamut from reducing the levels of an agent to removing it entirely.

## agent BINDS agent

Agents can bind other agents in the biochemical/kinetic sense. The **BINDS** relationship implies a direct binding between the two agents. If the two agents are components of a complex, but may not necessarily bind each other directly, the complex itself can be defined as another **agent** and each of the individual agents can be assigned a **BINDS** relationship with the complex.

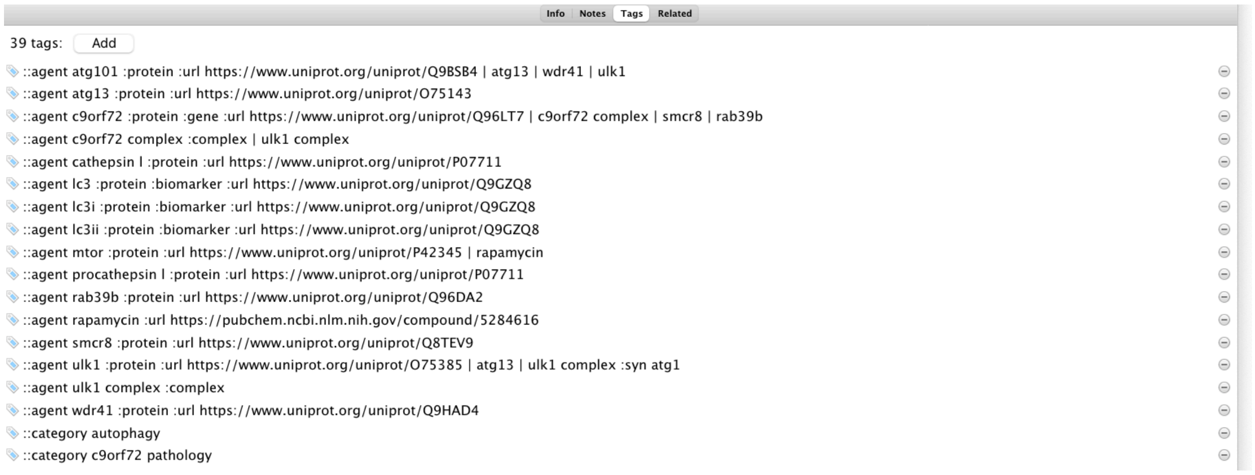
## agent MODIFIES agent

Agents can modify other agents biochemically. For example, a kinase **agent** can phosphorylate a protein **agent**, or a protease **agent** may cleave a protein **agent**. In order for this to happen, the two agents must also be able to come in contact with one another i.e. bind one another. For this reason, a **BINDS** relationship is automatically added between two agents involved in a **MODIFIES** relationship. In other words, the **BINDS** relationship is *implied* by the **MODIFIES** relationship so there is no need to also specify the **BINDS** relationship when assigning a **MODIFIES** relationship.

## scribl syntax

### statements, names, and fields

In the current version of **scribl**, statements appear as individual tags in the “Tags” field of an article, as shown below in Figure 9.



```
::agent atg101 :protein :url https://www.uniprot.org/uniprot/Q9BSB4 | atg13 | wdr41 | ulk1
::agent atg13 :protein :url https://www.uniprot.org/uniprot/O75143
::agent c9orf72 :protein :gene :url https://www.uniprot.org/uniprot/Q96LT7 | c9orf72 complex | smcr8 | rab39b
::agent c9orf72 complex :complex | ulk1 complex
::agent cathepsin l :protein :url https://www.uniprot.org/uniprot/P07711
::agent lc3 :protein :biomarker :url https://www.uniprot.org/uniprot/Q9GZQ8
::agent lc3i :protein :biomarker :url https://www.uniprot.org/uniprot/Q9GZQ8
::agent lc3ii :protein :biomarker :url https://www.uniprot.org/uniprot/Q9GZQ8
::agent mtor :protein :url https://www.uniprot.org/uniprot/P42345 | rapamycin
::agent procathepsin l :protein :url https://www.uniprot.org/uniprot/P07711
::agent rab39b :protein :url https://www.uniprot.org/uniprot/Q96DA2
::agent rapamycin :url https://pubchem.ncbi.nlm.nih.gov/compound/5284616
::agent smcr8 :protein :url https://www.uniprot.org/uniprot/Q8TEV9
::agent ulk1 :protein :url https://www.uniprot.org/uniprot/O75385 | atg13 | ulk1 complex :syn atg1
::agent ulk1 complex :complex
::agent wdr41 :protein :url https://www.uniprot.org/uniprot/Q9HAD4
::category autophagy
::category c9orf72 pathology
```

Figure 9.

Each statement is a separate line/tag in the tags field of the article and starts with a header that designates a **process**, **agent**, **category** or **resource** statement. It should be noted that there is no **article** header, since the tags already appear within the metadata of a specific article and all of the relationships that link that article to the **scribl** entities captured in its tags, can be *automatically* generated.

Each statement header consists of a **double colon**, followed by the **entity type** and the **entity name**, like this:

```
::process lysosomal protein degradation
::agent mtor
::category autophagy
::resource protein data bank
```

Entity names in **scribl** can be any set of whitespace separated words. A word must always start with an alphanumeric character (a-z,A-Z,0-9) but may contain any other printable character except the colon (:), the comma (,), and the semicolon (;).

If a statement gets very long it may become difficult to read or it will not be correctly handled by Zotero, which limits tag lengths to 255 characters. This is easily remedied however since a statement can be split across multiple lines, by repeating the statement header. It should be noted that the statement headers for a multi-line statement must be identical on each line. If the name is spelled differently on one of the lines, **scribl** will treat it as a separate entity.

For example, a valid multi-line **agent** statement looks like this:

```
::agent inpp5d :protein :gene :url https://www.uniprot.org/uniprot/Q92835
::agent inpp5d :syn phosphatidylinositol 3,4,5-trisphosphate 5-phosphatase 1
```

In every statement type, a set of optional fields follows the statement header and these can occur **in any order**, and **more than once** in a statement. There are 3 general fields that are common to all statement types:

**:url** - a valid internet url

**:tag** - one or more free form text tags separated by commas

**:txt** - a free form text annotation

The **:url** field must be followed by a valid URL, like this:

```
:url https://www.uniprot.org/uniprot/Q92835
```

If more than one url is required, multiple **:url** fields can be added.

In the case of agents, particularly for entities like genes, proteins, or defined biochemical structures, the **:url** field can be an invaluable method for identifying a specific gene, protein, or molecule, by linking it to a database such as **uniprot** (https://www.uniprot.org) or **pubchem** (https://pubchem.ncbi.nlm.nih.gov/) as a reference. This not only serves to make clear which specific entity the agent is intended to represent, but it also provides an invaluable data resource for that agent, allowing the user to retrieve additional information such as functional data, sequences, synonyms, chemical structures etc. etc.

The **:tag** field is essentially a comma-separated list of tags. Tags can be single or multiple words, but commas are treated as delimiters between tags.

```
:tag trna synthetase, class 2a trna synthetase, alanyl trna synthetase
```

As with the other fields, multiple **:tag** fields can be included - when for example, the list of tags has become inconveniently long.

The **:txt** field is for text annotations, but it does not support comma-separated values in the way that the **:tag** field does. If multiple annotations are required for a statement, multiple **:txt** fields can be added. Here for example, is the annotation for a biological models **resource**.

```
:txt A repository of mathematical models of biological and biomedical systems
```

## additional fields that are unique to agents

In addition to these 3 general fields that can be used in any statement, there are 2 further fields that are unique to **agent** statements.

The **:syn** field allows for the assignment of synonyms to an **agent**. Experience has shown that this can be a significant problem when using biological databases. The serine/threonine protein kinase ulk1 for example is also known as unc-51-like kinase 1, autophagy-related protein 1, and atg1. The :syn field like the tag field, is a comma-separated list of single or multi-word names that allows the agent to be searched by its alternative names, for example:

```
::agent ulk1 :syn unc-51-like kinase 1, autophagy-related protein 1, atg1
```

As with the **:tag** field, if the list of synonyms is extremely long, the synonyms can be divided into multiple **:syn** fields

Agents can also be classified into types using one or more of the following labels:

```
:protein, :gene, :dna, :rna, :mrna, :complex, :organelle, :biomarker
```

These labels are preceded by a single colon like the fields discussed previously, and are standalone items that require no additional text following them. They designate biological types and an **agent** may have no labels, a single label or multiple labels. Like synonyms, these labels are intended to make database searches easier, for example by narrowing down a list of searched agents to proteins. If the **agent** does not fall into any of the biological classes represented by these labels it will have no labels, for example:

```
::agent gtp :url https://pubchem.ncbi.nlm.nih.gov/compound/6830 :syn guanosine triphosphate
```

The number of different classes of chemical compounds is so vast that it would be impractical to have a unique label for each one. In such cases, the :tag field can be useful; for example:

```
::agent gtp :tag nucleoside, purine, nucleoside triphosphate
```

Conversely, an **agent** of a biological type may appear under the same name in an article as both a gene and its protein product.

```
::agent c9orf72 :gene :protein :url https://www.uniprot.org/uniprot/Q96LT7
```

As with all of the other fields, the **:syn** and label fields can appear in any order in the statement

## relationships

As previously stated, all of the relationships that link an article to the entities defined in its tags using **scribl**, are automatically generated. In contrast, any relationship that a defined entity shares with other defined entities must be explicitly declared in the **scribl** statement for that entity. For example, if a defined **process INVOLVES** a particular **agent** the relationship with that **agent** must be defined in the statement for that **process**. The @ symbol is used as the flag for the **process INVOLVES agent** relationship, so if the **process** "exportin releases cargo into cytoplasm" involves the **agent** "exportin-1", the statement for that **process** might look something like this:

```
::process exportin releases cargo into cytoplasm @ exportin-1
```

The relationship symbols for each type of user-defined relationship are as follows:

| entity | relationship | entity | relationship symbol |
|---|---|---|---|
| **resource** | DESCRIBES | **process** | **&** |
| **resource** | MENTIONS | **agent** | **%** |
| **process** | ACTIVATES | **process** | **>** |
| **process** | INHIBITS | **process** | **<** |
| **process** | REGULATES | **process** | **=** |
| **process** | INVOLVES | **agent** | **@** |
| **process** | GENERATES | **agent** | **+** |
| **process** | REMOVES | **agent** | **-** |
| **agent** | BINDS | **agent** | **\|** |
| **agent** | MODIFIES | **agent** | **~** |

Only one item should follow a relationship flag - the single or multi-word name of the other entity that participates in the relationship. Relationship flags do not support comma-separated lists the way that synonyms do. If a process for example, involves multiple agents, each of them should be defined with a separate @ flag like this:

```
::process tnf-alpha induction of il8 release @ progranulin @ tnf-alpha @ il8
```

It should also be noted that any of the single-character relationship flags must always surrounded on both sides by whitespace, for example:

```
@ ubiquitin
```

is a valid **INVOLVES** relationship statement, whereas:

```
@ubiquitin
```

is not.

The easiest way to understand the user-defined relationships is to see them in action. The following examples illustrate their use in **scribl** statements:

Process activates one process and inhibits another

```
::process rcc1 recycles ran-gdp to ran-gtp > importin releases cargo into nucleus  < nuclear import of fus
```

Process activates multiple other processes (multi-line statement)

```
::process cellular stress > nucleocytoplasmic transport dysfunction > mislocalization of nuclear proteins
::process cellular stress > cellular stress response > transportin-1 localizes to stress granules
::process cellular stress > stress-related phosphorylation of nucleoporins
```

Process activates, inhibits, and regulates other processes

```
::process smcr8 mutation > ulk1 phosphorylation < autophagy = smcr8 expression
```

Process involves several agents (and activates another process)

```
::process tbk1 activation of autophagy @ tbk1 @ optn @ sqstm1 @ ubiquitin > autophagy
```

Process involves one agent and generates another

```
::process procathepsin l processing @ procathepsin l + cathepsin l
```

Process involving an agent, removes one agent and generates another

```
::process phosphatase conversion of pi35p to pi3p  @ fig4 - pi35p + pi3p
```

Agent (a protein and a biomarker) binds two other agents

```
::agent lc3 :protein :biomarker  :url https://www.uniprot.org/uniprot/Q9GZQ8 | sqstm1 | optn
```

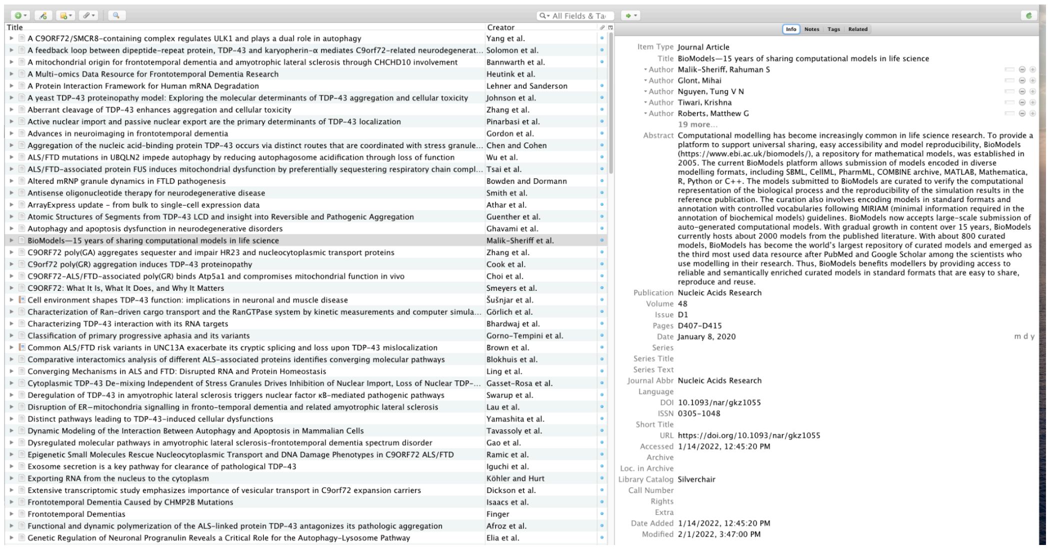
Agent modifies another agent

```
::agent caspase8 :protein :url https://www.uniprot.org/uniprot/Q14790 ~ beclin1
```

# **scribl** workflow

## inserting **scribl** statements as zotero tags

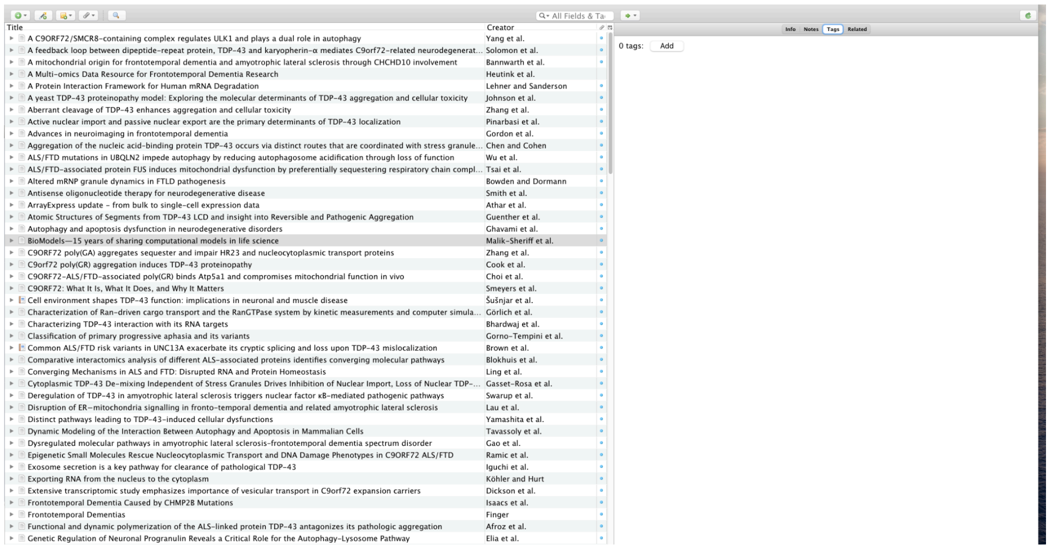
Starting to compile a **scribl** graph database requires only a working copy of the **Zotero** app. Upon opening a **Zotero** database using the local app (downloadable from https://www.zotero.org), each article will have a default view of its curated data that looks something like Figure 10a shown below. Clicking on the “Tags” button at the top of the right pane, switches to the tab that displays the article’s tags, as shown in Figure 10b.



| Title | Creator |
|---|---|
| A C9ORF72/SMCR8-containing complex regulates ULK1 and plays a dual role in autophagy | Yang et al. |
| A feedback loop between dipeptide-repeat protein, TDP-43 and karyopherin-α mediates C9orf72-related neurodegenerat… | Solomon et al. |
| A mitochondrial origin for frontotemporal dementia and amyotrophic lateral sclerosis through CHCHD10 involvement | Bannwarth et al. |
| A Multi-omics Data Resource for Frontotemporal Dementia Research | Heutink et al. |
| A Protein Interaction Framework for Human mRNA Degradation | Lehner and Sanderson |
| A yeast TDP-43 proteinopathy model: Exploring the molecular determinants of TDP-43 aggregation and cellular toxicity | Johnson et al. |
| Aberrant cleavage of TDP-43 enhances aggregation and cellular toxicity | Zhang et al. |
| Active nuclear import and passive nuclear export are the primary determinants of TDP-43 localization | Pinarbasi et al. |
| Advances in neuroimaging in frontotemporal dementia | Gordon et al. |
| Aggregation of the nucleic acid-binding protein TDP-43 occurs via distinct routes that are coordinated with stress granule… | Chen and Cohen |
| ALS/FTD mutations in UBQLN2 impede autophagy by reducing autophagosome acidification through loss of function | Wu et al. |
| ALS/FTD-associated protein FUS induces mitochondrial dysfunction by preferentially sequestering respiratory chain compl… | Tsai et al. |
| Altered mRNP granule dynamics in FTLD pathogenesis | Bowden and Dormann |
| Antisense oligonucleotide therapy for neurodegenerative disease | Smith et al. |
| ArrayExpress update – from bulk to single-cell expression data | Athar et al. |
| Atomic Structures of Segments from TDP-43 LCD and insight into Reversible and Pathogenic Aggregation | Guenther et al. |
| Autophagy and apoptosis dysfunction in neurodegenerative disorders | Ghavami et al. |
| BioModels—15 years of sharing computational models in life science | Malik-Sheriff et al. |
| C9ORF72 poly(GA) aggregates sequester and impair HR23 and nucleocytoplasmic transport proteins | Zhang et al. |
| C9orf72 poly(GR) aggregation induces TDP-43 proteinopathy | Cook et al. |
| C9ORF72-ALS/FTD-associated poly(GR) binds Atp5a1 and compromises mitochondrial function in vivo | Choi et al. |
| C9ORF72: What It Is, What It Does, and Why It Matters | Smeyers et al. |
| Cell environment shapes TDP-43 function: implications in neuronal and muscle disease | Šušnjar et al. |
| Characterization of Ran-driven cargo transport and the RanGTPase system by kinetic measurements and computer simula… | Görlich et al. |
| Characterizing TDP-43 interaction with its RNA targets | Bhardwaj et al. |
| Classification of primary progressive aphasia and its variants | Gorno-Tempini et al. |
| Common ALS/FTD risk variants in UNC13A exacerbate its cryptic splicing and loss upon TDP-43 mislocalization | Brown et al. |
| Comparative interactomics analysis of different ALS-associated proteins identifies converging molecular pathways | Blokhuis et al. |
| Converging Mechanisms in ALS and FTD: Disrupted RNA and Protein Homeostasis | Ling et al. |
| Cytoplasmic TDP-43 De-mixing Independent of Stress Granules Drives Inhibition of Nuclear Import, Loss of Nuclear TDP-… | Gasset-Rosa et al. |
| Deregulation of TDP-43 in amyotrophic lateral sclerosis triggers nuclear factor κB-mediated pathogenic pathways | Swarup et al. |
| Disruption of ER–mitochondria signalling in fronto-temporal dementia and related amyotrophic lateral sclerosis | Lau et al. |
| Distinct pathways leading to TDP-43-induced cellular dysfunctions | Yamashita et al. |
| Dynamic Modeling of the Interaction Between Autophagy and Apoptosis in Mammalian Cells | Tavassoly et al. |
| Dysregulated molecular pathways in amyotrophic lateral sclerosis-frontotemporal dementia spectrum disorder | Gao et al. |
| Epigenetic Small Molecules Rescue Nucleocytoplasmic Transport and DNA Damage Phenotypes in C9ORF72 ALS/FTD | Ramic et al. |
| Exosome secretion is a key pathway for clearance of pathological TDP-43 | Iguchi et al. |
| Exporting RNA from the nucleus to the cytoplasm | Köhler and Hurt |
| Extensive transcriptomic study emphasizes importance of vesicular transport in C9orf72 expansion carriers | Dickson et al. |
| Frontotemporal Dementia Caused by CHMP2B Mutations | Isaacs et al. |
| Frontotemporal Dementias | Finger |
| Functional and dynamic polymerization of the ALS-linked protein TDP-43 antagonizes its pathologic aggregation | Afroz et al. |
| Genetic Regulation of Neuronal Progranulin Reveals a Critical Role for the Autophagy-Lysosome Pathway | Elia et al. |

| | |
|---|---|
| Item Type | Journal Article |
| Title | BioModels—15 years of sharing computational models in life science |
| Author | Malik-Sheriff, Rahuman S |
| Author | Glont, Mihai |
| Author | Nguyen, Tung V N |
| Author | Tiwari, Krishna |
| Author | Roberts, Matthew G |
| | 19 more... |
| Abstract | Computational modelling has become increasingly common in life science research. To provide a platform to support universal sharing, easy accessibility and model reproducibility, BioModels (https://www.ebi.ac.uk/biomodels/), a repository for mathematical models, was established in 2005. The current BioModels platform allows submission of models encoded in diverse modelling formats, including SBML, CellML, PharmML, COMBINE archive, MATLAB, Mathematica, R, Python or C++. The models submitted to BioModels are curated to verify the computational representation of the biological process and the reproducibility of the simulation results in the reference publication. The curation also involves encoding models in standard formats and annotation with controlled vocabularies following MIRIAM (minimal information required in the annotation of biochemical models) guidelines. BioModels now accepts large-scale submission of auto-generated computational models. With gradual growth in content over 15 years, BioModels currently hosts about 2000 models from the published literature. With about 800 curated models, BioModels has become the world’s largest repository of curated models and emerged as the third most used data resource after PubMed and Google Scholar among the scientists who use modelling in their research. Thus, BioModels benefits modellers by providing access to reliable and semantically enriched curated models in standard formats that are easy to share, reproduce and reuse. |
| Publication | Nucleic Acids Research |
| Volume | 48 |
| Issue | D1 |
| Pages | D407–D415 |
| Date | January 8, 2020 m d y |
| Series | |
| Series Title | |
| Series Text | |
| Journal Abbr | Nucleic Acids Research |
| Language | |
| DOI | 10.1093/nar/gkz1055 |
| ISSN | 0305-1048 |
| Short Title | |
| URL | https://doi.org/10.1093/nar/gkz1055 |
| Accessed | 1/14/2022, 12:45:20 PM |
| Archive | |
| Loc. in Archive | |
| Library Catalog | Silverchair |
| Call Number | |
| Rights | |
| Extra | |
| Date Added | 1/14/2022, 12:45:20 PM |
| Modified | 2/1/2022, 3:47:00 PM |

Figure 10a.



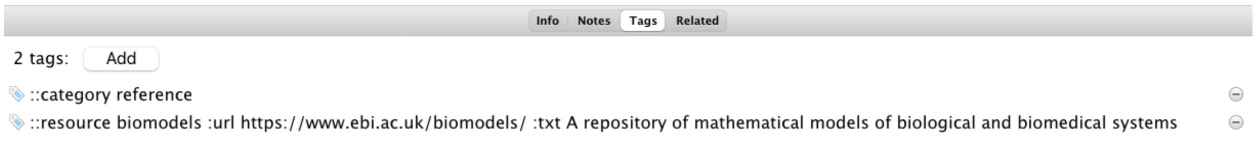
0 tags:

| Title | Creator |
|---|---|
| A C9ORF72/SMCR8-containing complex regulates ULK1 and plays a dual role in autophagy | Yang et al. |
| A feedback loop between dipeptide-repeat protein, TDP-43 and karyopherin-α mediates C9orf72-related neurodegenerat… | Solomon et al. |
| A mitochondrial origin for frontotemporal dementia and amyotrophic lateral sclerosis through CHCHD10 involvement | Bannwarth et al. |
| A Multi-omics Data Resource for Frontotemporal Dementia Research | Heutink et al. |
| A Protein Interaction Framework for Human mRNA Degradation | Lehner and Sanderson |
| A yeast TDP-43 proteinopathy model: Exploring the molecular determinants of TDP-43 aggregation and cellular toxicity | Johnson et al. |
| Aberrant cleavage of TDP-43 enhances aggregation and cellular toxicity | Zhang et al. |
| Active nuclear import and passive nuclear export are the primary determinants of TDP-43 localization | Pinarbasi et al. |
| Advances in neuroimaging in frontotemporal dementia | Gordon et al. |
| Aggregation of the nucleic acid-binding protein TDP-43 occurs via distinct routes that are coordinated with stress granule… | Chen and Cohen |
| ALS/FTD mutations in UBQLN2 impede autophagy by reducing autophagosome acidification through loss of function | Wu et al. |
| ALS/FTD-associated protein FUS induces mitochondrial dysfunction by preferentially sequestering respiratory chain compl… | Tsai et al. |
| Altered mRNP granule dynamics in FTLD pathogenesis | Bowden and Dormann |
| Antisense oligonucleotide therapy for neurodegenerative disease | Smith et al. |
| ArrayExpress update – from bulk to single-cell expression data | Athar et al. |
| Atomic Structures of Segments from TDP-43 LCD and insight into Reversible and Pathogenic Aggregation | Guenther et al. |
| Autophagy and apoptosis dysfunction in neurodegenerative disorders | Ghavami et al. |
| BioModels—15 years of sharing computational models in life science | Malik-Sheriff et al. |
| C9ORF72 poly(GA) aggregates sequester and impair HR23 and nucleocytoplasmic transport proteins | Zhang et al. |
| C9orf72 poly(GR) aggregation induces TDP-43 proteinopathy | Cook et al. |
| C9ORF72-ALS/FTD-associated poly(GR) binds Atp5a1 and compromises mitochondrial function in vivo | Choi et al. |
| C9ORF72: What It Is, What It Does, and Why It Matters | Smeyers et al. |
| Cell environment shapes TDP-43 function: implications in neuronal and muscle disease | Šušnjar et al. |
| Characterization of Ran-driven cargo transport and the RanGTPase system by kinetic measurements and computer simula… | Görlich et al. |
| Characterizing TDP-43 interaction with its RNA targets | Bhardwaj et al. |
| Classification of primary progressive aphasia and its variants | Gorno-Tempini et al. |
| Common ALS/FTD risk variants in UNC13A exacerbate its cryptic splicing and loss upon TDP-43 mislocalization | Brown et al. |
| Comparative interactomics analysis of different ALS-associated proteins identifies converging molecular pathways | Blokhuis et al. |
| Converging Mechanisms in ALS and FTD: Disrupted RNA and Protein Homeostasis | Ling et al. |
| Cytoplasmic TDP-43 De-mixing Independent of Stress Granules Drives Inhibition of Nuclear Import, Loss of Nuclear TDP-… | Gasset-Rosa et al. |
| Deregulation of TDP-43 in amyotrophic lateral sclerosis triggers nuclear factor κB-mediated pathogenic pathways | Swarup et al. |
| Disruption of ER–mitochondria signalling in fronto-temporal dementia and related amyotrophic lateral sclerosis | Lau et al. |
| Distinct pathways leading to TDP-43-induced cellular dysfunctions | Yamashita et al. |
| Dynamic Modeling of the Interaction Between Autophagy and Apoptosis in Mammalian Cells | Tavassoly et al. |
| Dysregulated molecular pathways in amyotrophic lateral sclerosis-frontotemporal dementia spectrum disorder | Gao et al. |
| Epigenetic Small Molecules Rescue Nucleocytoplasmic Transport and DNA Damage Phenotypes in C9ORF72 ALS/FTD | Ramic et al. |
| Exosome secretion is a key pathway for clearance of pathological TDP-43 | Iguchi et al. |
| Exporting RNA from the nucleus to the cytoplasm | Köhler and Hurt |
| Extensive transcriptomic study emphasizes importance of vesicular transport in C9orf72 expansion carriers | Dickson et al. |
| Frontotemporal Dementia Caused by CHMP2B Mutations | Isaacs et al. |
| Frontotemporal Dementias | Finger |
| Functional and dynamic polymerization of the ALS-linked protein TDP-43 antagonizes its pathologic aggregation | Afroz et al. |
| Genetic Regulation of Neuronal Progranulin Reveals a Critical Role for the Autophagy-Lysosome Pathway | Elia et al. |

Figure 10b.

If the article contains some automatically curated tags, they will appear in this pane. These tags will be ignored by the **scribl** parser but they will generate warnings during parsing, as well as making it harder to keep your **scribl** statements organized (we have found from experience that it is easier to delete them and start afresh).

To start adding tags, just click on the “Add” button at the top of the pane and you can type in your **scribl** statements to capture the information in the article, as you are reading it. Figure 11, shows the tags pane after adding a **category** and a **resource** statement. The resource statement contains a **:url** field that links to the online resource and a **:txt** field annotation that describes the resource itself.

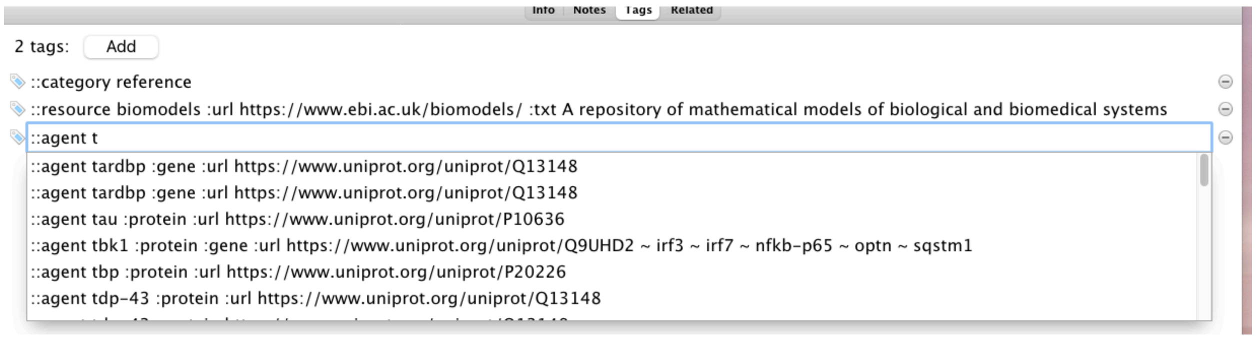


```
::category reference
::resource biomodels :url https://www.ebi.ac.uk/biomodels/ :txt A repository of mathematical models of biological and biomedical systems
```

Figure 11.

You will notice that **Zotero** has the very nice feature of automatically sorting tags alphabetically, no matter in which order you enter them. This conveniently groups together all statements of a particular type (agent, category, process, resource) and sorts them alphabetically within the group, making it very easy to locate a particular statement when the collection of statements in an article starts to get large.

Another extremely useful feature in **Zotero**, is that it offers autocompletion for statements. As you start typing an agent statement for example, if the partially-typed agent name matches any other agent in any other article in the database, one or more autocomplete options will be offered. This is useful not only for reducing the amount of typing, but it also maintains consistency in the spelling of statement names. Figure 12. shows an example of autocomplete in action as an agent statement is being added, but this feature works when starting to type any of the 4 statement types.



```
::category reference
::resource biomodels :url https://www.ebi.ac.uk/biomodels/ :txt A repository of mathematical models of biological and biomedical systems
::agent t
::agent tardbp :gene :url https://www.uniprot.org/uniprot/Q13148
::agent tardbp :gene :url https://www.uniprot.org/uniprot/Q13148
::agent tau :protein :url https://www.uniprot.org/uniprot/P10636
::agent tbk1 :protein :gene :url https://www.uniprot.org/uniprot/Q9UHD2 ~ irf3 ~ irf7 ~ nfkb-p65 ~ optn ~ sqstm1
::agent tbp :protein :url https://www.uniprot.org/uniprot/P20226
::agent tdp-43 :protein :url https://www.uniprot.org/uniprot/Q13148
```

Figure 12.

The consistent spelling of agents and processes in particular, is very important for the compilation of the relationships between them. If, for example, an agent name has a different spelling in two separate articles, the relationships it shares with other processes and agents that span the 2 articles, will not be correctly compiled. The autocomplete is not a perfect solution to this by any means, but it does go a long way to ensuring the consistency of name spellings between articles. When adding process or agent names to relationships within statements, the autocomplete feature cannot be relied upon. When **scribl** compiles the statements from the **Zotero** database upon loading its data, it does however check every name in every relationship against the catalog of all known entities of the appropriate type, to make sure that the name exists in the catalog. It also checks every relationship to make sure that it is valid for the type of statement in which it is being used. This extensive error-checking will be discussed in more detail later on, but it does allow the user to pinpoint the exact articles and statements in which any spelling and/or syntax errors occur, and fix them before reloading the database.

It is a requirement that all named processes and agents that appear in the relationships defined in the tags of a particular article, also have their own process and agent statements that define them within that article. What does this mean?

For example, say a process is being defined that involves a particular agent and generates another one - like this:

```
::process procathepsin l processing @ procathepsin l + cathepsin l
```

It is required that the agents **procathepsin l** and **cathepsin l** are also defined within the **scribl** statements for the article in which this **process** statement appears - like this:

```
::agent cathepsin l :protein :url https://www.uniprot.org/uniprot/P07711
::agent procathepsin l :protein :url https://www.uniprot.org/uniprot/P07711
```

When you start to type either of these agent statements, the autocomplete feature in **Zotero** may indicate that these agents are already defined in other articles, at which point it might be tempting to skip adding them because they are already defined elsewhere.

**This will not work! Any process or agent named in a relationship, must also have its own process or agent statement within that article.**

This means that the curation of data for each article is a standalone, consistent dataset that does not depend upon the data curated in any other article in the database. Without this requirement, if one curator were, for example, to edit or remove an agent from the data that they had curated in another article, this would destroy the relationships defined by another curator who was relying upon their presence in this other article. This requirement facilitates a distributed and collaborative curation model for **scribl** by forcing each curator to be explicit about the entities and relationships they are curating, and not depending upon what they might assume about the intentions of other curators.

## scribl curation in more detail

**category**: Categories are broad subtopic definitions whose purpose is really to recapitulate in the graph database, the subtopic headings in the literature database. **Zotero** for example, does not export subheadings as tags, but the use of the **::category** statement in **scribl** enables these groupings to persist in the graph database.

```
::category rna processing
```

**resource**: A resource is a source of data or information that is referenced in an article. Resources include databases and other data repositories, the contents of which may also be related to the article's biological processes (via the DESCRIBES relationship) and/or agents (via the MENTIONS relationship). A **::resource** statement will almost always have at least one **:url**, and it is also useful to include **:txt** and **:tag** fields to better describe it. A resource may contain data relevant to many processes and agents, but these should be limited to the processes and agents that are specifically discussed in the article. An article may discuss a resource in general terms without reference to any particular processes and agents, in which case there is no need to catalog its processes and agents. Any number of articles may reference the same resource and in general it is best to link the resource only to the processes and agents that are specifically described in a particular article. Remember to use multi-line statements if the lists of processes and agents get too long for single lines.

```
::resource protein data bank :url https://www.rcsb.org/search
::resource protein data bank :txt archive of structural data of biological macromolecules

::resource model of rangtp gradient :txt A model of the nucleocytoplasmic ranGTP gradient
::resource model of rangtp gradient :url https://www.ebi.ac.uk/biomodels/BIOMD0000000192
::resource model of rangtp gradient % ran % rangap1 % gtp & nucleocytoplasmic transport
::resource model of rangtp gradient & ran binds gtp & rangap1 hydrolyzes ran-gtp to ran-gdp
```

**process**: As previously discussed, a process can be as specific as the binding of two agents or as generic as a broad cellular function such as nuclear export. It is often stated in research articles that a particular protein regulates a process. What this really means is that the protein regulates the process by binding and/or modifying another protein or biochemical entity. Since **scribl** agents do not regulate processes directly, all that is necessary in order to capture such a relationship is to define the binding and/or modification interaction in a **::process** statement that INVOLVES the relevant agents. For example, to capture the statement "AMPK regulates autophagy", we define the process by which this regulation occurs and the agents involved in it.

```
::agent ampk :protein  :url https://www.uniprot.org/uniprot/Q9Y478 | ulk1
::agent ulk1 :protein :syn kiaa0722
::process ampk binds ulk1 @ ampk @ ulk1 = autophagy
::process autophagy
```

**agent**: The **::agent** statements have additional fields because agents can be harder to pin down than other entities. One significant problem in biology for example, is the plethora of names that are often used for the same agent, as well as the endless variety of variants - mutants and splice-variants in proteins for example. The **:syn** field allows the user to enter agent synonyms and the **scribl** graph database not only allows the user to catch errors such as misspellings, but it can also scan all of the synonyms in the database to catch for example:

- the same agent appearing more than once but under different names
- synonyms that occur in agents with different names

For this reason, **it is very important to use the :syn field extensively**, wherever possible, to capture alternative names for agents. It is not necessary to include the long names of agents in the synonyms, since these can be captured in the **:txt** or even the **:tag** fields. Another useful tip is to include possible alternative spellings of agent names that include modifiers that may or may not be separated by dashes, such as ulk1, mcp1, il12, etc. For example:

```
::agent ulk1 :syn ulk-1
::agent mcp1 :syn mcp-1
::agent il12 :syn il-12, interleukin-12
```

The label fields allow the user to also specify any biological types for agents, to facilitate searching by biological type. It is up to the user whether to define separate agents for genes and the proteins that they encode. This can be complicated by the fact that many proteins are simply named after their genes, for example, **c9orf72** refers both to the chromosomal locus of the gene, as well as the protein that it encodes. In other cases, the gene and its protein may have completely separate names and can be easily distinguished, for example, the gene **TARDBP** that encodes the protein **TDP-43** (or **TDP43**).

Given this problem, and since agents can have multiple labels, one approach is to treat a protein that has the same name as its gene, as the same entity - and to rely upon the context for which type is relevant. This approach can work very well - for example:

```
::agent c9orf72  :protein :gene :url https://www.uniprot.org/uniprot/Q96LT7 | smcr8
::agent smcr8 :protein :url https://www.uniprot.org/uniprot/Q8TEV9
::process c9orf72 binds smcr8 @ c9orf72 @ smcr8
::process c9orf72 hexanucleotide repeat expansion @ c9orf72 < nuclear transport
::process nuclear transport
```

The agent **c9orf72** is labeled as both a protein and a gene, since the protein and the gene share the same name. In the first process **c9orf72 binds smcr8** the agent is clearly acting as protein binding another protein. In the second process **c9orf72 hexanucleotide repeat expansion**, it is clearly acting as a gene.

Since no two agents may share the same name, if another curator labels an agent you have already defined with an additional type, the agents will be merged anyway. One alternative option would be to create some kind of clunky naming scheme such as:

```
::agent c9orf72_protein :protein :url https://www.uniprot.org/uniprot/Q96LT7
::agent c9orf72_gene :gene :url https://www.uniprot.org/uniprot/Q96LT7
```

But this is **not** recommended.

When it comes to non-biological agents such as chemical compounds, it would be a challenge to realistically implement definitions for all possible chemical types. In such cases however, the **:syn**, **:tag**, and **:txt** fields can all be your friend. For example:

```
::agent gtp :url https://pubchem.ncbi.nlm.nih.gov/compound/6830 :syn guanosine triphosphate
::agent gtp :txt chemical formula C10 H16 N5 O14 P3 :tag nucleoside, purine
```

The importance of the **:url** field in an agent statement is also clear from the examples above, since it serves to show the database user exactly which entity is intended.

## importing the literature database

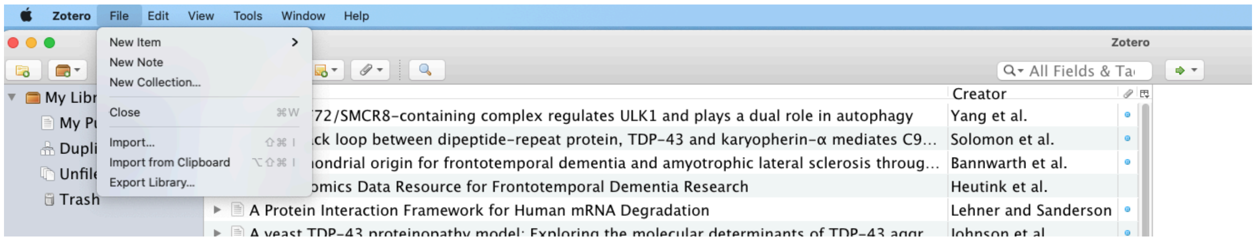
The current version of **scribl** supports the import of **Zotero** database via both a direct querying via the Zotero API, as well as via a file exported from the Zotero database formatted as comma-separated values (CSV)[9]. Based upon the notion that a literature database is never truly “complete”, **scribl** is designed to update its internal graph database incrementally with each import of literature data. In other words, as the literature database grows, **scribl** determines which articles and data are new and should be added to its internal graph database. This means that **scribl** data imports can be performed as needed while the literature database is being developed. This is also a good practice since there will inevitably be a few errors in each batch of new data, and these are easier to track down and fix in smaller batches.

## exporting directly from the database

**scribl** can directly query the database via the Zotero API. This method is described later.

## exporting Zotero to a CSV file

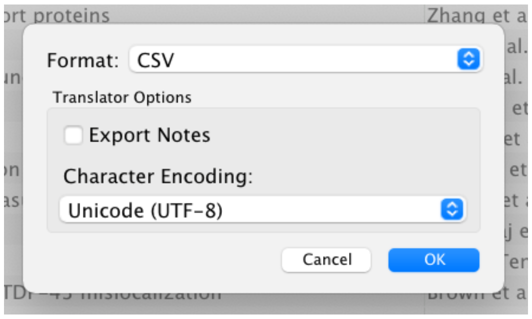
To export the database of articles with their **scribl** statements, use the **File** menu in **Zotero** and select **Export Library**, as shown in Figure 13 below.



| | |
|---|---|
| 72/SMCR8–containing complex regulates ULK1 and plays a dual role in autophagy | Yang et al. |
| ck loop between dipeptide–repeat protein, TDP–43 and karyopherin–α mediates C9... | Solomon et al. |
| ondrial origin for frontotemporal dementia and amyotrophic lateral sclerosis throug... | Bannwarth et al. |
| omics Data Resource for Frontotemporal Dementia Research | Heutink et al. |
| A Protein Interaction Framework for Human mRNA Degradation | Lehner and Sanderson |
| A yeast TDP–43 proteinopathy model: Exploring the molecular determinants of TDP–43 aggr | Johnson et al. |

Figure 13.

When the export pop-up menu appears, select the **CSV** option for **Format**, and the **Unicode (UTF-8)** option for **Character Encoding**, as shown in Figure 14.



Format: CSV

Translator Options

Export Notes

Character Encoding: Unicode (UTF-8)

Figure 14.

After selecting **OK** you can then select a folder and file name for the exported CSV data.

## obtaining and setting up the **scribl** software

In order to obtain and run **scribl**, a basic, working knowledge of Python, GitHub, and Python virtual environments is required.

**scribl** is a freely available, open-source platform written entirely in Python[10], which is itself a freely-available, open source programming language. All that is needed to run **scribl** is a Python 3 runtime, and the **scribl** source code that is currently available at **GitHub** (https://github.com/amberbiology/scribl). The easiest way to get started with **scribl** is to clone the GitHub repository referenced above and set up a virtual environment to run it in, that includes other Python dependencies, such as **pyparsing**[15], **pyzotero** and others.

There are a number of different ways to set up the **scribl** software but the basic steps are as follows, and all of the required software and tools are free and open-source. Note that the most up-to-date installation instructions are always maintained in the **README.md** (https://github.com/amberbiology/scribl/tree/main#readme) in the GitHub repository, and in the event of a discrepancy, those instructions should always supercede the following:

1. Install **Python 3** if you don't already have it (https://www.python.org)
2. Install **pip** if you don't already have it (https://pypi.org/project/pip)
3. Install **Git** if you don't already have it (https://git-scm.com)
4. Use Python to create a virtual environment in which to run scribl.
   - python3 -m venv /path_to_my_virtual_environments/scribl
5. Activate the new virtual environment:
   - source /path_to_my_virtual_environments/scribl/bin/activate
6. Clone the GitHub **scribl** repository to your local drive[16]
   - git clone https://github.com/amberbiology/scribl.git
7. Change into the scribl directory and install **scribl** and its Python dependencies into the new virtual environment:
   - cd scribl
   - pip install .

Now you can run the **scribl** Python code within your new virtual environment either by running the Python interpreter in the virtual environment (in the example above, this would be located at: /path_to_my_virtual_environments/scribl/bin/python3) - or if you are running the code from within a Python development environment like PyCharm (https://www.jetbrains.com/pycharm) you would select the new virtual environment in the **Project Interpreter** options.In addition, we have provided an installed script, **scribl**, that can be used to run several common **scribl** tasks - such database creation, and outputs, directly from the command-line, without needing to write any Python. More details are available on the README.md and via running scribl --help.

[15] https://pyparsing-docs.readthedocs.io/en/latest/pyparsing.html
[16] Once the package is available via PyPI steps 6 and 7 can be replaced with "pip install scribl"

## **scribl** software overview

As shown in Figure 15 below, the 2 major workflows for the **scribl** software are creating a new graph database and updating an existing one.

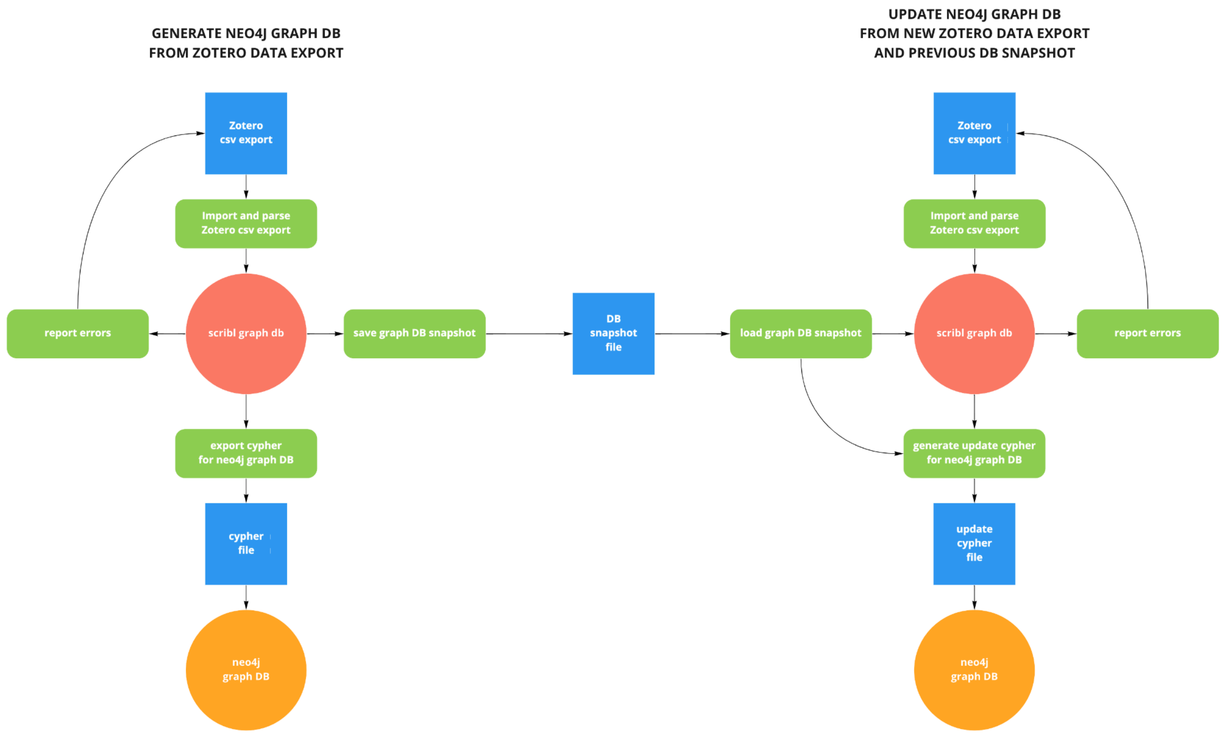


Figure 15.

In the left hand workflow, **Zotero** export datais loaded into **scribl**, which checks for, and pinpoints, errors as it parses the data. The articles and their associated **scribl** tags are converted into an internal graph database format which can be viewed and also saved as a snapshot of the current database. This internal graph database can also be used to generate the cypher[11] language statements that are used to create the graph database in **neo4j**[12].

In the right-hand workflow, the **neo4j** is updated with any newly added data, by taking the previous internal database snapshot, and comparing it with the newly-loaded **Zotero** data. In this case, **scribl** will generate only the cypher statements that are needed to define the new entities and relationships in the **neo4j** graph database, or to update any additional data or relationships that were added to existing ones.

The **scribl** Python code is not written as a standalone application, but rather a system of Python modules that can be integrated into a workflow. These modules provide all of the functionality needed for **i**. processing the exported literature data; **ii**. parsing the **scribl** statements attached to each article; **iii**. generating the internal graph data structure; and **iv**. exporting the graph data in formats that can be recognized by a graph database engine such as **neo4j**[12] or via Python network library, such as **NetworkX**. The **scribl** Python code also includes a number of tools for error-checking and inspection of the graph data, and these are described in more detail in the next section.

In order to demonstrate how the **scribl** modules can be used in the workflows shown in Figure 15, the Python code also includes a template for a kind of simple, administrative front end for a graph database that uses **scribl** as its data resource. A set of simple Python functions capture the basic database operations described above, and can be used to initialize a new database, import data from **Zotero**, check the data for errors, and export the cypher statements for input into **neo4j**.

At the time of writing, there are two graph databases supported by **scribl**: **neo4j** and **GraphML/NetworkX** (and could be extended to even more graph databases in the future). To connect to **neo4j** scribl, can export its internal graph data as **cypher**[11], the language used to create, edit, and query the **neo4j** database. To allow import directly into the various Python modules for handling graph data structures, such as **NetworkX**, **scribl** can output a subset of the database to the **GraphML** format. We have provided some preliminary support in Python for reading and visualizing the GraphML within **scribl** itself. At the time of writing, this is a work-in-progress, and does not yet have complete feature-parity with the neo4j output, and therefore the rest of the guide largely focuses on using **scribl** in the context of neo4j.

## the scribl Python modules

The **scribl** Python package contains the following modules:

process_zotero
parse_scribl
process_graphdb_data
manage_graphdb
graph_db_commands

The manage_graph_db module contains some high-level methods for creating and managing a graph database that utilizes the more granular methods in the other modules. These granular methods will be documented first, and this will make it easier to understand what is happening at the higher level in the manage_graph_db module.

## processing exported data from Zotero

The ZoteroCSV class is imported from the process_zotero module and instantiated by providing the path to a CSV formatted file containing the Zotero export data. For example:

```
from scribl.process_zotero import ZoteroCSV
zotero_csv_data = '/user/me/zotero/exports/my_latest_zotero_export.csv'
zotero_data = ZoteroCSV(zotero_csv_data)
```

The default choice of columns that are processed and their mapping to the article fields in the graph database, are defined in default_keymap in the scribl package's init.py. These can be changed however by using the ZoteroCSV object's map_keys() method. For example:

```
zotero_keys = ['Key', 'Title', 'Url', 'Publication Year', 'Author', 'Publication Title']
cypher_keys = ['zotero_key', 'title', 'url', 'year', 'author', 'journal_title']
zotero_data.map_keys(zotero_keys, cypher_keys)
```

It should be noted that the first 2 keys in each set cannot be changed, since these define critical fields used by **scribl** to store **Zotero** articles.

## parsing **scribl** statements

A ScriblParser object is imported from the parse_scribl module, and its parse() method can be used to parse **scribl** statements. In **Zotero**, tags are separated by a semicolon (;) so this is the default delimiter for the parse() method. This can however be changed should the parser be supplied **scribl** statements with a different delimiter.

```
from scribl.parse_scribl import ScriblParser
scribl_parser = ScriblParser()
scribl_parser.parse(my_scribl_text)
```

The parser will automatically check for errors such as unrecognized agents or processes, as well as ensuring that the defined relationships are valid for the entities to which they are applied. It will also flag any statements that are longer than 255 characters, the current maximum for **Zotero**. All warnings and errors can be inspected in the parser's data dictionary, for example:

```
print(scribl_parser.data['warnings']
print(scribl_parser.data['errors']
```

The parser has a get() method to retrieve the data for a parsed statement, for example:

```
ampk = scribl_parser.get('::agent', 'ampk')
print(ampk)
{'urls': ['https://www.uniprot.org/uniprot/Q9Y478'], 'labels': [':protein'], 'tags': [], 'notes': [], …
```

There's also a catalog() method to list all the entities of a given type, for example:

```
print(scribl_parser.catalog('::agent')[:5])
['ambra', 'ampk', 'atg1-atg13 complex', 'atg13', 'atg14']
```

And finally, a parse_summary() method that provides the counts for each statement type, as well as the counts for any warnings and errors, for example:

```
print(scribl_parser.parse_summary)
{'::category': 2, '::agent': 35, '::process': 34, '::resource': 0, 'errors': 0, 'warnings': 0}
```

## generating the graph data structure

The GraphDB class in the process_graphdb_data module is a higher-level object for creating and storing graph data structures from the article data and **scribl** data imported from **Zotero**, as well as providing the functionality for exporting it for use in other graph database platforms. As such, it aggregates much of the functionality in the process_zotero and parse_scribl modules. The GraphDB class also provides functionality for saving and loading snapshots of its graph data, as well as some tools for inspecting and error-checking it.

A GraphDB object can be instantiated with a path to a file of exported **Zotero** CSV data:

```
zotero_csv_data = ‘/user/me/zotero/exports/my_latest_zotero_export.csv’
graphdb = GraphDB(zotero_csv_data)
```

Or, to load a previously saved database snapshot:

```
graphdb_snapshot = ‘/user/me/graphdb/snapshots/my_latest_graphdb_snapshot.dat’
graphdb = GraphDB(graphdb_snapshot, export_type=scribl.DB_EXPORT)
```

The default export_type (defined in the **scribl** package’s init) is scribl.ZOTERO_EXPORT. When loading a file of exported **Zotero** data, the GraphDB class also provides options for defining the mapping of **Zotero** data columns to fields in the graph database. For inspecting the contents of the graph database, the GraphDB class also provides get() and catalog() methods. These work very similarly to those in the ScriblParser class, except that they now enable the user to inspect the *entirety* of the graph data (aggregated from all of the articles), instead of just the data parsed from a single article.

```
print(graphdb.get(‘agent’, ‘ulk1’)
{'urls': ['https://www.uniprot.org/uniprot/O75385'], 'tags': [], 'notes': ['ulk1 is phosphorylated by mtor'],'labels': [':protein'], 'synonyms': ['ulk1', 'atg1']}
```

```
print(graphdb.catalog('agent')[:5])
['alpha-synuclein', 'ambra', 'ampk', 'arp2', 'arp3']
```

The GraphDB catalog() method can also catalog all of the different types of relationships that have been generated from the **scribl** statements.

```
 relationships = graphdb.catalog('BINDS', relationship=True)
 for r in relationships[:5]:
            print('BINDS', r)
BINDS ('ambra', 'beclin1')
BINDS ('ambra', 'uvrag')
BINDS ('atg1-atg13 complex', 'atg13')
BINDS ('atg1-atg13 complex', 'atg9')
BINDS ('atg1-atg13 complex', 'fip200')
```

For examining the relationships generated within the graph data, the GraphDB class provides a show_relationships() method that lists all of the relationships for the specified entity in a Python dictionary format, for example:

```
ulk1_relationships = graphdb.show_relationships('agent', 'ulk1')
print(ulk1_relationships['MODIFIES'])
[('mtor', 'ulk1')]
```

The entire graph data structure (all articles, entities, and their relationships) can be saved as a single file using the GraphDB save_db() method:

```
snapshot_filepath = '/user/me/graphdb/snapshots/my_latest_graphdb_snapshot.dat'
graphdb.save_db(snapshot_filepath)
```

The GraphDB class also provides the load_db() method for reading in a snapshot file as a new graph database. It is important to note that the load_db() method returns a **new** graph database and does **not** re-instantiate the current database. This method allows the current version of the database to be compared with a previous snapshot of the database, which facilitates the export of database updates to **neo4j** that contain only the data that was added to the database since the last snapshot. This enables incremental updates to be made to the external graph database (currently **neo4j**) without having to build it entirely from scratch each time the exported **Zotero** database contains new data. How this works will become apparent when we get to the description of the generate_cypher() method.

```
snapshot_filepath = '/user/me/graphdb/snapshots/previous_graphdb_snapshot.dat'
previous_db = graphdb.load_db(snapshot_filepath)
```

The generate_cypher() method can be used either to export the entire graph data set as input to **neo4j**, or it can be used to export just the data that has been added since a previous database snapshot was taken. If no previous database snapshot is supplied to the generate_cypher() method, it will generate cypher text for the entire, current graph data structure. If a previous database snapshot is supplied, only the cypher text corresponding to the differences between the current and previous version of the graph data is generated:

```
cypher = graphdb.generate_cypher()
```

The example above generates the cypher text for the current version of the entire graph data structure. Loading a previous snapshot of the graph data structure allows the user to generate the cypher text only for the new data.

```
snapshot_filepath = '/user/me/graphdb/snapshots/previous_graphdb_snapshot.dat'
previous_db = graphdb.load_db(snapshot_filepath)
cypher = graphdb.generate_cypher(diff_db=previous_db)
```

The export_cypher() method returns a Python list of cypher statements. If you want to export the cypher as text with each statement on a separate line, the list of statements can be passed to the export_cypher_text() method for conversion into text.

```
cypher_text = graphdb.export_cypher_text(cypher)
```

This is also useful for writing the exported cypher statements into a file for input into **neo4j**, as we will see later in the description of the graph_db_commands module.

The GraphDB class provides 2 essential error-checking tools that should be part of the workflow for the creation and/or maintenance of a graph database generated using **scribl**. The use of different agent names in biology has already been raised as an issue and it is for this reason that the ability to add synonyms to **scribl** agents is so important, and is **strongly encouraged**. The check_synonyms() method enables the user to see if the same agent exists in the database under different synonyms. The check_synonyms() method looks for synonyms that are used as agent names, and for agents that share the same synonyms despite having different names. In some cases, these anomalies will be the result of the same agent being present in the database under different names, but in some cases they will also identify errors in which agents are incorrectly named or have the wrong synonyms. For example:

```
synonym_check = graphdb.check_synonyms()
```

The check_synonyms() method returns a dictionary with the keys 'synonym appears in different agents' and 'synonym appears as an agent', as shown below.

```
print(synonym_check['synonym appears in different agents'])
{'dup_sym': ['ambra', 'caspase-1']}
print(synonym_check['synonym appears as an agent'])
[]
```

The other essential error-checking tool is the check_agent_labels() method that provides a list of all agents that have no biological labels. In many cases these will not be errors, since the agents are non-biological entities such as chemical compounds. It is however very easy to forget to add biological labels to biological agents, and the check_agent_labels() method is a quick and easy way to check this.

```
print(graphdb.check_agent_labels())
['ca2+', 'rapamycin', ‘gtp’, ‘gdp’, ‘actinomycin’]
```

When loading a new set of exported **Zotero** data, it is recommended to always use the check_synonyms() and check_agent_labels() methods as validation tools in tandem with the automatic error checking. This enables errors and inconsistencies to be identified and fixed in the **Zotero** database, before reloading and running these checks again.

## managing a **scribl** graph database

The manage_graphdb module in the **scribl** Python package, is provided as a simple template to demonstrate how the **scribl** modules might be used to create and maintain a graph database based upon **Zotero** and **neo4j**. The basic workflows follow the schemas shown in Figure 15, with some additional operations added for file management, as follows:

If no database already exists:

- Create a new graph database folder structure and metadata

Then:

- Import the latest **Zotero** CSV data as a timestamped file into the exports folder
- Load the latest **Zotero** data and check for general and synonym errors
- Fix the errors in **Zotero** and re-import the CSV data (keeping the same timestamp)

If generating or reloading the entire neo4j graph database from scratch:

- Export the **scribl** graph data structure as cypher
- Import the cypher into **neo4j**
- Save the current, timestamped data as a new snapshot

If updating the **neo4j** graph database with only the data added since the last upload:

- Load the latest **scribl** database snapshot
- Export only the cypher for the new data using the previous snapshot for reference
- Import the cypher into **neo4j**
- Save the current, timestamped data as a new snapshot

The manage_graphdb module has a template class GraphDBInstance whose methods, described below, cover the functionality needed to implement a workflow of the kind described above.

Note that if no **return** is specified, the method returns None.

```
GraphDBInstance.__init__(str: folder_path,
                         bool: overwrite=False,
                         bool: verbose=False)
```

This method takes a folder_path as an argument, and will either open an existing **scribl** graph database if one exists at that location, or it will create one if not. Optionally can be set to overwrite the current one (defaults to False), and verbose output can be set (defaults to False).

**Returns**: GraphDBInstance: an instance of the class

```
GraphDBInstance.set_metadata(str: db_name,
                             str: curator,
                             str: description,
                             bool: overwrite=False)
```

This method takes three arguments: db_name, curator, and description and writes them to the metadata.txt file in the database's configuration folder. It will also create an annotations.txt file in that folder if one does not already exist. Can optionally overwrite.

```
GraphDBInstance.add_annotation(str: your_name,
                               str: note_text)
```

The GraphDBInstance class provides a simple template for the use of metadata and annotations for the database as a whole. The add_annotation() method provides the functionality of a rudimentary database log, enabling time-stamped annotations to be added to the database by the curators and editors.

```
GraphDBInstance.generate_metadata_cypher()
```

The metadata and annotations can also be exported as a metadata node into **neo4j**, so that anybody using the **neo4j** graph database can also have access to it.

**Returns**: str: the line-separated metadata in Cypher text format

```
GraphDBInstance.import_zotero_csv(str: import_file_path,
                                  bool: overwrite=False,
                                  bool: verbose=False)
```

This method takes a **Zotero** export file from a defined location and adds it to the database exports folder. If the overwrite argument is set to True (the default is False) the import operation will use the same timestamp as the latest import. This is to allow one or more cycles of error-checking and editing of the **Zotero** database, and its re-import, without having to create a bunch of new time-stamped files. Also accepts the verbose argument.

**Returns**: str: the newly imported file path

```
GraphDBInstance.import_zotero_library(int: zotero_library_id,
                                      str: zotero_library_type,
                                      str: zotero_library_api=None,
                                      bool: overwrite=False,
                                      bool: verbose=False)
```

This method connects directly to the **Zotero** library via the Zotero API and then internally adds it to the database exports folder. The two required arguments are zotero_library_id (an int library number), zotero_library_type (this is a str, with value of either group or user). There are then three optional arguments zotero_library_api (a str, only needed when supplying an API key for accessing private libraries), and the previously described overwrite and verbose arguments (both default to False). Otherwise this functions in the same way as import_zotero_csv().

```
GraphDBInstance.load_zotero_csv(str: zotero_csv_filename=None,
                                list: zotero_keys=None,
                                list: cypher_keys=None,
                                bool: verbose=False)
```

This method has no required arguments, it loads, parses, and error-checks the latest time-stamped **Zotero** data file in the database exports folder if no specific file name is provided as an argument. The method also allows optional zotero_csv_filename, and alternative **Zotero** keymap provided as a list of strings (zotero_keys) and equivalent list of Cypher strings (cypher_keys) to be provided if the user wishes to override the defaults. Lastly it includes the optional verbose argument.

**Returns**: tuple: a tuple consisting of two lists of warnings and errors.

```
GraphDBInstance.save_db_snapshot(bool: verbose=False)
```

This method has no required arguments and saves the current database as a snapshot in a Python data format, in the database snapshots folder. The saved file will have the same timestamp as the latest **Zotero** export. It optionally can take the verbose flag (defaults to False).

**Returns**: str: the new saved filename.

```
GraphDBInstance.load_db_snapshot(str: db_snapshot_filename=None,
                                 bool: verbose=False)
```

This method has no required arguments and loads an entire **scribl** database from the latest time-stamped file in the database snapshots folder, unless a specific snapshot filename is provided (db_snapshot_filename). The method does not overwrite the current database, but simply returns the new database that was loaded from the snapshot file. This method is used to load the latest, previous database snapshot when generating cypher that contains only data added since the last version of the database.

**Returns**: dict: the loaded snapshot file as a dictionary

```
GraphDBInstance.export_cypher_text(dict: diff=None,
                                   bool: verbose=False,
                                   bool: filepath=None)
```

This method has no required arguments and exports either the entire **scribl** graph data structure as cypher for input to **neo4j** - or - if a previous snapshot (as a dict) is also provided in the method's optional diff argument, it will generate only the cypher necessary to update the database from its previous version (as defined in the database snapshot). It also, optionally, can be set to produce verbose output (defaults to False), and also save the output to filepath (defaults to not saving to a file).

**Returns**: str: the Cypher text

```
GraphDBInstance.export_graphml_text(bool: verbose=False,
                                    bool: filepath=None)
```

This method exports the entire **scribl** graph data structure as **GraphML** for input to Python's own **NetworkX** - this method does not support the diff argument. It also, optionally, can be set to produce verbose output (defaults to False), and also save the output to filepath (defaults to not saving to a file).

**Returns**: str: the GraphML XML text

```
GraphDBInstance.export_graphml_figure(bool: verbose=False,
                                      bool: filepath=None)
```

This method generates an output figure of the entire **scribl** graph data structure using the **NetworkX** library, based on the GraphML representation generated by export_graphml_text(). It also, optionally, can be set to produce verbose output (defaults to False), and also saves the image to filepath

```
GraphDBInstance.backup_db(bool: verbose=False)
```

This method is essentially the same as the save_db_snapshot() method, except that it allows the user to save a time-stamped version of the current **scribl** database as a snapshot in the database backup folder. It also, optionally, can be set to produce verbose output (defaults to False).

**Returns**: str: the path to the new backup

GraphDBInstance.inspect_db(list: list_contents=[],
bool: verbose=False,
int: contents_length=5)

This method enables the contents of the currently loaded **scribl** database to be inspected. By default, this method returns a Python dictionary of counts for each entity and relationship type in the database, as well as the counts of any warnings and errors. Listings of the catalog for any named entity or relationship can also be generated by optionally adding to the list_contents argument, either a single name in the case of entities, or a tuple containing the key 'relationships' and the name of the relationship type - for example:

```
gdb.inspect_db(list_contents=['agent', 'process', ('relationships','BINDS')])
```

This method also optionally takes a verbose flag which displays more detailed information on each entity, and in this verbose mode, the contents_length can be modified (the output is limited to 5 by default).

**Returns**: dict: containing counts for each entity

GraphDBInstance.check_synonyms(bool: verbose=False)

Unlike regular error-checking, synonym checking is **not** run automatically when the **scribl** database is loaded. It is recommended that the check_synonyms() method be run each time new data is loaded, to flag any problems and allow them to be fixed in **Zotero**. This method takes the optional verbose flag (set to False).

**Returns**: dict: containing synonym information

GraphDBInstance.check_agent_labels(bool: verbose=False)

Unlike regular error-checking, agent label checking is also **not** run automatically when the **scribl** database is loaded. It is recommended that the check_agent_labels() method be run each time new data is loaded, to flag any problems and allow them to be fixed in **Zotero**. This method takes the optional verbose flag (set to False).

**Returns**: list: agents missing labels

## semantic graphs for literature searches

The ultimate goal of using **scribl** to capture all of these relationships between articles, resources, processes, and agents, is the generation of semantic graphs that support relationship-based literature searches that are more biologically-focused than the keyword and metadata-based searches offered by traditional literature databases. A semantic graph database enables the researcher to run literature searches based upon the *meaning* captured in the articles rather than just the presence or absence of certain terms or keywords. For example, imagine that a researcher is interested in the cellular events that inhibit autophagy, and the genes and proteins that might play a role in this.

A traditional literature search query might look something like this:

```
article contains_terms('autophagy', 'inhibition')
```

and would likely return a collection of articles in which the terms 'autophagy' and 'inhibition' would appear, albeit not necessarily together. Some of the articles may well contain the term 'inhibition' in a context that is not necessarily related to autophagy, while some of the articles may discuss autophagy more generally but not necessarily in the context of its inhibition. The use of the conjoined term 'autophagy inhibition' may miss a lot of articles in which this specific term is absent, even if the search algorithm was capable of matching keywords and synonyms - recognizing for example that terms like 'prevent', 'hinder', or 'impede' are synonyms for 'inhibit'. After this initial search, a closer reading of the subset of retrieved articles that do turn out to be relevant would then be necessary in order to identify candidates for the genes and/or proteins that are actually involved in autophagy inhibition.

For comparison, a semantic graph search using a graph database created with **scribl**, allows the researcher to frame such a search like this:

```
articles that mention agents involved in processes that inhibit autophagy
```

With this single query, a graph database built for the **neo4j** platform using **scribl**, can retrieve a connected graph like the one shown in Figure 16 below. This graph (from the partially curated current version of the database) consists of 22 **articles** (shown in yellow) that mention 7 **agents** (shown in red) involved in 7 biological **processes** (shown in blue) that inhibit autophagy (also shown in blue since it is itself a biological process). Not only does the graph identify the agents (proteins, genes etc.) and biological processes involved in the inhibition of autophagy and the articles that describe them, but it also shows some of the relationships between the individual agents and processes (which agents participate in which processes) and which specific articles describe each of them.

If we zoom in on a portion of the graph (as shown in Figure 17) we can see for example that the agent **c9orf72** is mentioned in several of the retrieved articles and that it inhibits autophagy through the process **c9orf72 hexanucleotide expansion**. If we double-click on one of the articles that mentions **c9orf72** (as shown in Figure 18) we can view the complete database entry for the article with all of the associated metadata that you would expect to pull up from a literature database search - authors, title, publication year, journal title etc. along with an internet url that takes you straight to the article online.

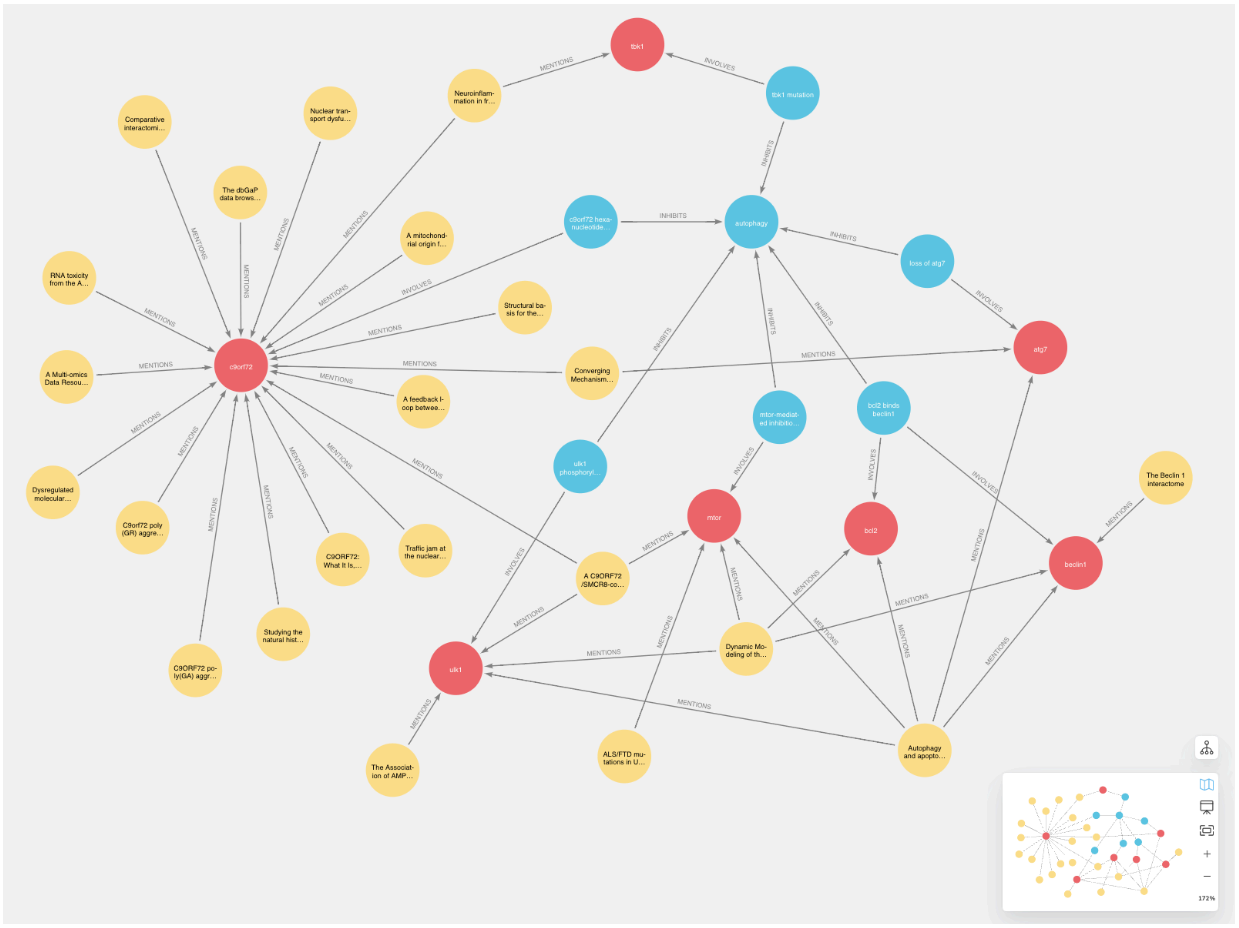


Figure 16.

The entities shown as nodes in the graph actually have a much richer, interconnecting set of relationships (shown as labeled edges in the graph) than what is shown based upon the query. The set of relationships that is shown is only a subset of all relationships between the nodes in the graph, that is bounded by the terms of the query itself. In this case for example, we only asked to see the **MENTIONS**, **INVOLVES**, and **INHIBITS** relationships. Once a graph has been retrieved however, it can be expanded to include other relationships and entities, as will be shown subsequently.

It should also be noted that the query itself was framed in an idiom that is very close to natural language. This is because **neo4j** has the ability to build the more syntactically rigorous cypher queries that it actually uses for search queries, from more natural, informal statements in which it can recognize entity labels (e.g. **AGENT**, **PROCESS**), entity names (e.g. **c9orf72**, **autophagy**), and relationship labels such as **INVOLVES**, **INHIBITS**, **MENTIONS**.

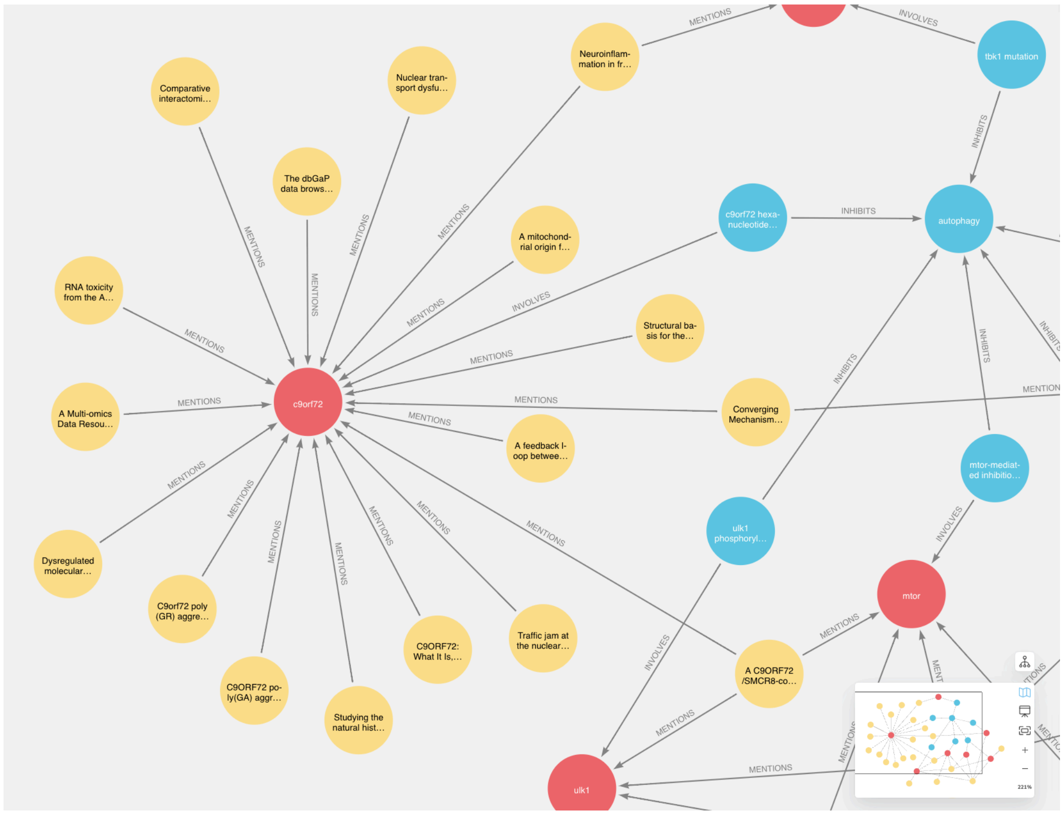
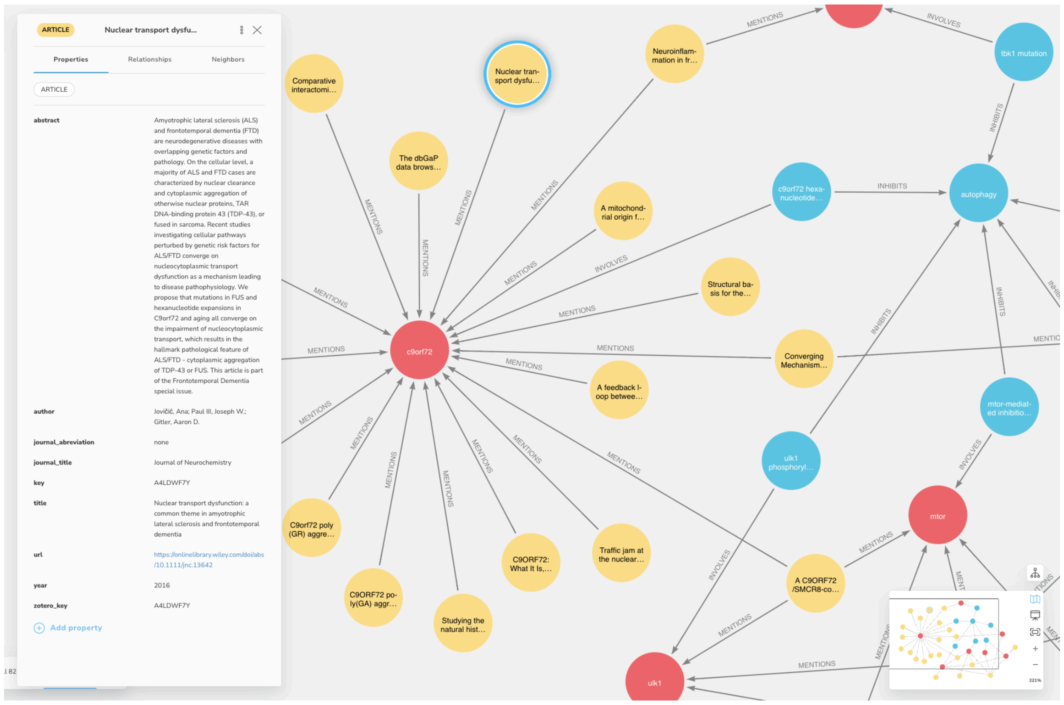


Figure 17.



| | |
|---|---|
| abstract | Amyotrophic lateral sclerosis (ALS) and frontotemporal dementia (FTD) are neurodegenerative diseases with overlapping genetic factors and pathology. On the cellular level, a majority of ALS and FTD cases are characterized by nuclear clearance and cytoplasmic aggregation of otherwise nuclear proteins, TAR DNA-binding protein 43 (TDP-43), or fused in sarcoma. Recent studies investigating cellular pathways perturbed by genetic risk factors for ALS/FTD converge on nucleocytoplasmic transport dysfunction as a mechanism leading to disease pathophysiology. We propose that mutations in FUS and hexanucleotide expansions in C9orf72 and aging all converge on the impairment of nucleocytoplasmic transport, which results in the hallmark pathological feature of ALS/FTD - cytoplasmic aggregation of TDP-43 or FUS. This article is part of the Frontotemporal Dementia special issue. |
| author | Jovičić, Ana; Paul III, Joseph W.; Gitler, Aaron D. |
| journal_abreviation | none |
| journal_title | Journal of Neurochemistry |
| key | A4LDWF7Y |
| title | Nuclear transport dysfunction: a common theme in amyotrophic lateral sclerosis and frontotemporal dementia |
| url | https://onlinelibrary.wiley.com/doi/abs/10.1111/jnc.13642 |
| year | 2016 |
| zotero_key | A4LDWF7Y |

Figure 18.

The retrieved graph can be expanded by either selecting nodes in the graph and expanding them to show more of their relationships and neighboring nodes, or by extending the terms of the query itself. In Figure 19 for example, we have expanded the node for the **mtor** agent. This reveals other articles that also mention **mtor,** and that **mtor** binds the agent **rapamycin** and binds and modifies the agents **ulk1** and **atg13**.

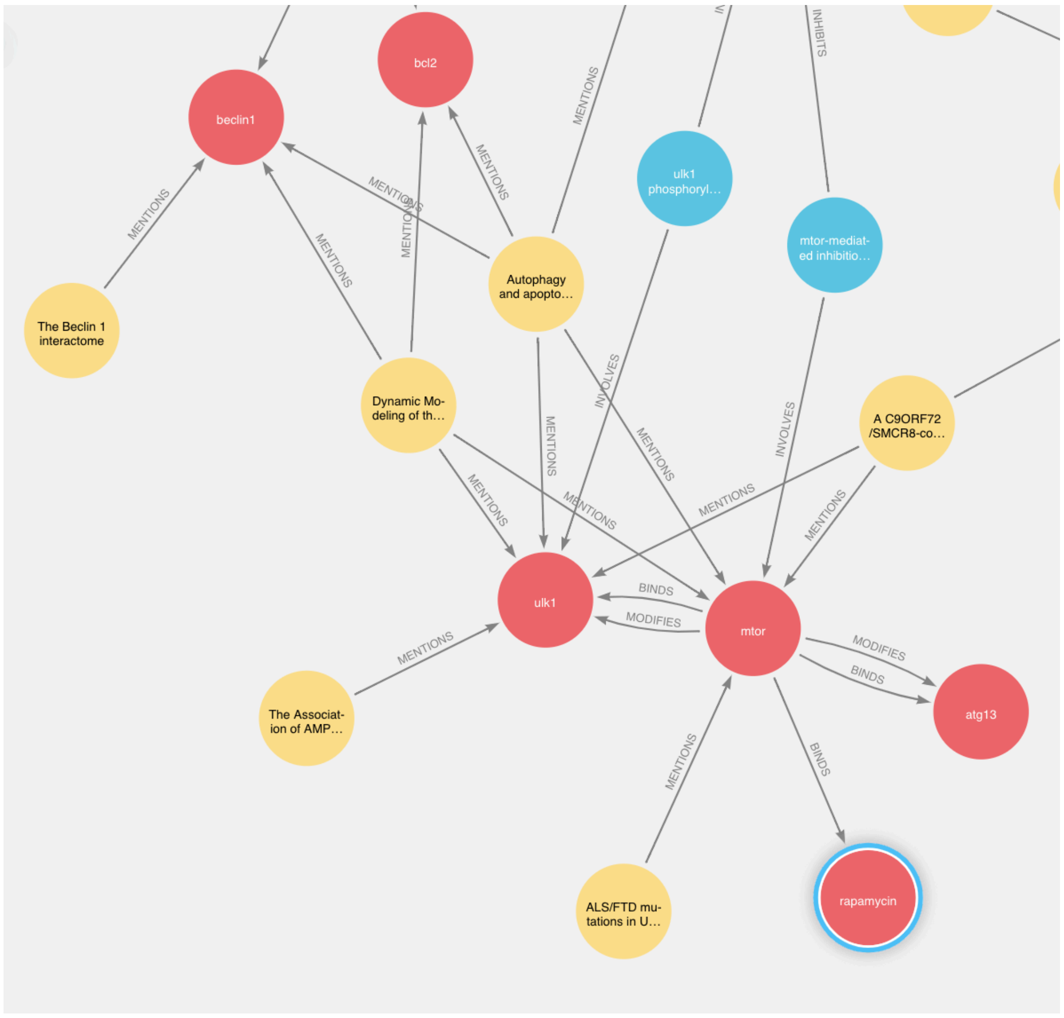


Figure 19.

We could also have expanded the set of agents returned by the query by expanding the terms of the query itself to include not only the agents involved in biological processes that inhibit autophagy, but also the agents that interact with them. For example, in the following query:

article mentions agent - agent involved in a process that inhibits autophagy

We are expressing that the query should return not only agents that are involved in the inhibition of autophagy, but also the agents that interact with them. As shown in Figure 20 below, with this query we are effectively starting to build a graph that shows not only the literature on the inhibition of autophagy, but also the cellular interactome for the inhibition of autophagy.

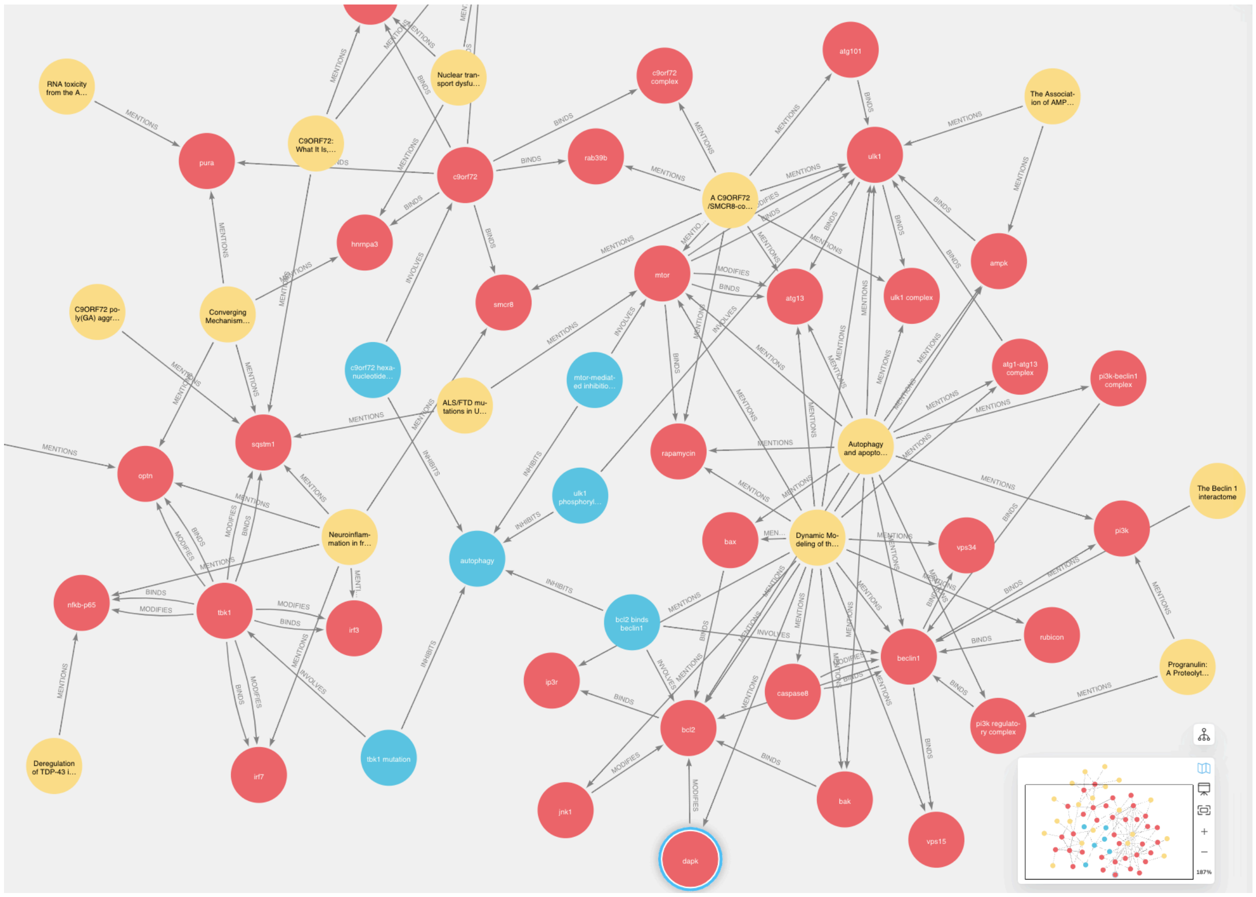


Figure 20.

The formal query language for **neo4j** is cypher, the same language that is used for adding and editing entities and their relationships in the graph database. As previously described, the syntax for cypher is far more formal and rigorously defined than the kind of natural language queries described here. As such however, it also offers more fine-grained control over the data that can be retrieved. The previous informal query statement for example, could be written in cypher like this:

```
MATCH (a:ARTICLE)-[x:MENTIONS]-(b:AGENT)--(c:AGENT)<-[y:INVOLVES]-
(d:PROCESS)-[z:INHIBITS]->(e:PROCESS{name:'autophagy'}) return a,b,c,d,e,x,y,z
```

The informal query interface is a great deal easier to use for researchers with only limited experience of graph databases. The modicum of time and effort required to learn cypher however, is definitely worthwhile for a researcher who would like to run more advanced queries and take advantage of some of the powerful graph algorithms that can be used to query and analyze their graph data in more sophisticated ways. While a full description of the cypher language is outside the scope of this guide, there are many excellent cypher documents, lessons, and tutorials available online, first and foremost at the **neo4j** website[12].

# appendix a: scribl formal grammar

For reference the formal grammar for the **scribl** language used to capture semantic relationships, is provided below in Backus–Naur form[17]

```
start_name_char ::=     "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" |
"a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m" | "n" |
"o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z" | "A" | "B" |
"C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" | "K" | "L" | "M" | "N" | "O" | "P" |
"Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z" |

symbol ::=  "!" | """ | "#" | "$" | "'" | "(" | ")" | "*" | "." | "/" | "?" | "[" |
"\" | "]" | "^" | "_" | "`" | "{" | "}"

url_symbol ::=    "!" | """ | "#" | "$" | "(" | ")" | "*" | "." | "/" | ";" | "?" |
"_" | ">" | "<" | "=" | "@" | "+" | "-" | "&" | "%"

relation_flag ::= ">" | "<" | "=" | "@" | "|" | "~" | "+" | "-" | "&" | "%"

space ::= " "

scribl_token ::= ":"

comma ::= ","

statement_type ::= "agent" | "category" | "process" | "resource"

url_flag ::= "url"

tag_flag ::= "tag"

txt_flag ::= "txt"

syn_flag ::= "syn"

label ::= "protein" | "gene" | "dna" | "rna" |
"mrna" | "complex" | "organelle" | "biomarker"

whitespace ::= <space> | <whitespace> <space>

header ::= <scribl_token> <scribl_token> <statement_type>

name_char ::= <start_name_char> | <symbol> | <relation_flag>

name_word ::= <start_name_char> | <word> <name_char>

name ::= <name_word> | <name> <whitespace> <name_word>

url_start ::= "http:" | "https:"

url_char ::= <start_name_char> | <url_symbol>
```

[17] https://en.wikipedia.org/wiki/Backus%E2%80%93Naur_form

```
url_body ::= <url_char> | <url_body> <url_char>

url_link ::= <url_start> <url_body>

comma_delimiter ::= <comma> | <comma> <whitespace> | <whitespace> <comma> <whitespace>

comma_list ::= <name> | <comma_list> <comma_delimiter> <name>

url_field ::= <scribl_token> <url_flag> <whitespace> <url_link>

tag_field ::= <scribl_token> <tag_flag> <whitespace> <comma_list>

txt_field ::= <scribl_token> <txt_flag> <whitespace> <name>

syn_field ::= <scribl_token> <syn_flag> <whitespace> <comma_list>

agent_label ::= <scribl_token> <label>

relationship ::== <relation_flag> <whitespace> <name>

statement_header ::= <header> <name>

optional_field = <url_field> | <tag_field> | <txt_field> | <syn_field> |
<agent_label> | <relationship>

statement_body ::= <optional_field> | <statement_body> <whitespace> <optional_field>

statement ::= <statement_header> <whitespace> <statement_body>
```