# Konduit VQB: a Visual Query Builder for SPARQL on the Social Semantic Desktop

**Oszkár Ambrus**
oszkar.ambrus@deri.org

**Knud Möller**
knud.moeller@deri.org

**Siegfried Handschuh**
siegfried.handschuh@deri.org

Digital Enterprise Research Institute (DERI)
National University of Ireland, Galway (NUIG)

## ABSTRACT

With the adoption of Nepomuk as an organic part of KDE the semantic desktop became a reality to a great number of users and is employed by a growing number of applications. Thus, the amount of semantic data is constantly growing on the desktop. Therefore users need a way to access this data outside of the limiting use cases of the applications employing Nepomuk-KDE.

We aim to assist users in building queries and running them to make use of RDF data that would otherwise be partially or completely hidden. In this paper, as an initial iteration of our efforts, we present four approaches to building SPARQL queries visually, based on two different categorizations: schema-based vs. instance-based and SELECT vs. CONSTRUCT queries. We present the used interfaces, visual languages and query generation methods associated to each of approaches as well as the autocompletion techniques for the instance-based query builders.

## Author Keywords

Visual Query Builder, SPARQL, Nepomuk

## INTRODUCTION

The Social Semantic Desktop [4] is a paradigm transposing Semantic Web concepts unto the desktop. Ontologies thus conceptualize information and semantic data is stored in RDF. It loosens the borders between applications and provides a unified environment. The Nepomuk project [6] outlines the requirements and functionalities of the Social Semantic Desktop and defines an architecture specification that fulfills these requirements. Nepomuk-KDE[1] is a reference implementation of Nepomuk. It provides a platform to create and handle all kinds of metadata. It uses RDF stores for the metadata persistence and provides a middleware for applications to build upon, allowing them to store and access the semantic data on the desktop (or, alternatively, the Web).

[1]http://nepomuk.kde.org/

Workshop on Visual Interfaces to the Social and Semantic Web (VISSW2010), IUI2010, Feb 7, 2010, Hong Kong, China. Copyright is held by the author/owner(s).

Konduit [10] is a tool for building visual workflows for RDF data within Nepomuk-KDE, allowing for a flexible access to the local RDF data as well as mashing up with web-based data. It features a visual programming environment and allows for various manipulations (merging, filtering, mashing up, creating visual workflows, etc.) as well as executing different actions (executing scripts, automatizing emails, etc.) using the queried RDF data. A query builder is used to generate SPARQL queries for querying components which act as data sources in the RDF workflows, producing data that is made use of further in the workflow.

We want to provide a way for users to build these queries in an intuitive way, with having no or little knowledge about the querying language (i.e., SPARQL [11]). Although this does not mean a complete abstraction from the underlying details, we aim to assist users with limited technical knowledge as well as those who know SPARQL and RDF (with an emphasis on the latter, though) and provide an interface that suits the needs of both. We try to provide a tool with the necessary features similar tools provide, that also supports RDF, provides search assistance on a whole repository (as opposed to a single ontology) and is also integrated into Nepomuk-KDE (within Konduit and beyond) allowing for local data-driven querying as well as sharing queries online.

We explore several approaches due to the structured nature of RDF data and the difficulty of searching and querying it in an intuitive and transparent manner. We present, as a first attempt in our research on visual query builders, four interfaces aiming to achieve the above goal, with different approaches to query building and varying degrees of complexity. The first two of them are schema-based, allowing for building queries using restrictions based on the ontology structure. The second two use a triple construction-based approach; the user constructs the restricting triples of the SPARQL query assisted by suggestions using both schema and instance information from the underlying repository.

## RELATED WORK

There are a number of tools that aim to assist users in building queries for semantic data. Many of them provide novel and intuitive approaches and demonstrate useful features, such as NITELIGHT [12] or RDF-GL [7] aiming to represent SPARQL constructs through graphical metaphors. MashQL [8], GRQL [1] and GLOO [5] propose queries as trees starting from a given class and restricting it incrementally on the branches. SPARQLViz [2] provides a click-through wizard for composing queries and SEWASIE [3] features a limited ontology-based query formulation.

Nevertheless, several of the tools only support querying based on a single ontology, some of them do not support RDF and SPARQL. Some of them require extensive manual editing of the queries, or do not feature clear relationships between query parts. Moreover, excepting SPARQLViz, all query builders are web-based (or only usable within their own system), not allowing for the integration of desktop data.

Our two schema-based interfaces are mostly built on the intuition of MashQL and GRQL, in exposing schema structures and possible restrictions branching from an initial class. The instance-based query builders resemble SPARQLViz in providing forms for the user to complete, but feature a single, less confusing and simple interface, with clear connections between the query parts.

## RUNNING EXAMPLE

Suppose we are searching for a contact from the local repository whose name contains the letter ‘K’ and has written a publication on the semantic desktop.

## QUERY BUILDING

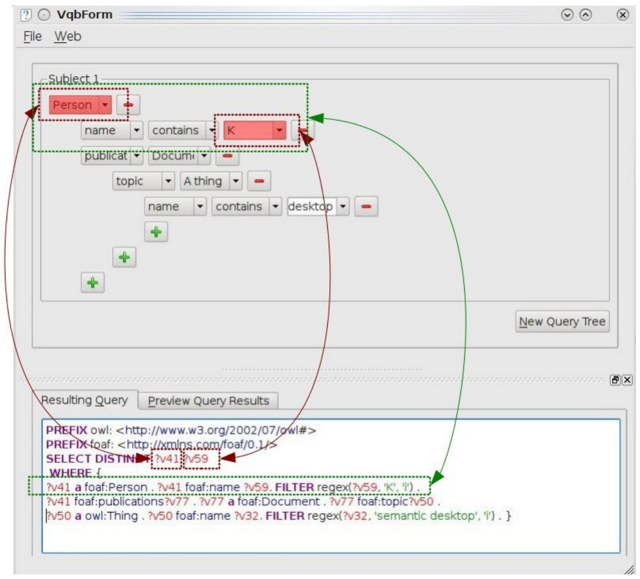
The interface of the query builder application features a central part for the visual query builder and previewers for the query and its results, as well as menu actions and a status bar. The central query builder has four incarnations as presented in the following sections, employing different approaches to building SPARQL queries.

The most common searching interface, a search box for simple keyword searches to retrieve semantic information is highly ambiguous and needs extensive research, so abstracting completely from the structure of semantic data is not yet our intention. Also, we can’t yet provide a fully-featured mature semantic querying application, as we aim to explore the most appropriate ways to do it and research the possibilities and limitations in accomplishing this task.

The two schema-based approaches use schema information from the ontologies in the Nepomuk system, allowing users to compose tree-based queries in restricting the properties of the resulting objects. This allows users to explore the local schema structures. These approaches feature a simplification of SPARQL, in allowing only to restrict the initial selection descending in a tree-like fashion.

The instance-based approaches are based on constructing triples without necessarily knowing RDF or SPARQL (similarly to the Wikipedia Visual Query Builder[2]. They use schema information as well as instance information in suggesting users possibilities in completing the subjects, predicates and objects of the constraining triples of the SPARQL query. This autocompletion allows for users to explore the data stored on the local RDF repository. The instance-based query builders also allow for the construction of triples that

[2]`http://dl-learner.org/Projects/dbpedia`



```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT DISTINCT ?v41 ?v59
 WHERE {
?v41 a foaf:Person . ?v41 foaf:name ?v59. FILTER regex(?v59, 'K', 'i') .
?v41 foaf:publications?v77 . ?v77 a foaf:Document . ?v77 foaf:topic?v50 .
?v50 a owl:Thing . ?v50 foaf:name ?v32. FILTER regex(?v32, 'semantic desktop', 'i') . }
```

**Figure 1. Schema-based SELECT query builder.**

don’t exist in the repository, adding properties that have not been defined or converting an instance from one ontology to another, e.g., transforming `?v foaf:name ``Smith''` to `?v nco:fullName ``Smith''`.

## SCHEMA-BASED SELECT QUERIES

The schema-based SELECT query builder (Figure 1) is the most user-friendly approach to building SPARQL queries, aimed at users having the least knowledge of semantic technologies. We have devised a simplification of the SPARQL language, which allows for this particular kind of builder. It allows for selecting a class (which will be the type of the queried variable) and restricting it in a tree-like manner through its properties.

Queries are built using schema information from Nepomuk. The possible classes and predicates to be chosen are queried and presented to the user.

We start from the assumption that users most often want to find certain information belonging to an entity of certain classification and/or having a number of known restricting characteristics. For example, one would want to search for a contact person (entity) with a given name (restricting characteristic), similarly to doing a free text search.

The interface therefore provides a way to build queries as trees, starting with the type of entity the user inquires for and progressing with restrictions on the branches of the tree.

### The Visual Language

The visual language covers a subset of SPARQL. Listing 1 provides a formal description of the queries built through the visual facilities of the interface. Note that the missing terminal definitions $ClassName$ and $Predicate$ are IRI references, $LiteralValue$ is a string literal and $VariableName$ is a SPARQL variable (such as `?v` or `$x`). $Relation$ denotes a relation such as `contains`, `equals`, etc.

```
Query              ::= Outputs Conditions
Outputs            ::= RootNode | LiteralNode
Conditions         ::= GraphPattern
GraphPattern       ::= QueryTree+
QueryTree          ::= RootNode TreeNumber
RootNode           ::= ClassName VariableName Restrictions
Restrictions       ::= QueryNode*
QueryNode          ::= ClassRestriction | LiteralRestriction
ClassRestriction   ::= Predicate RootNode
LiteralRestriction ::= Predicate Relation LiteralNode
LiteralNode        ::= VariableName LiteralValue
```

**Listing 1. EBNF description of the visual language (non-terminals)**

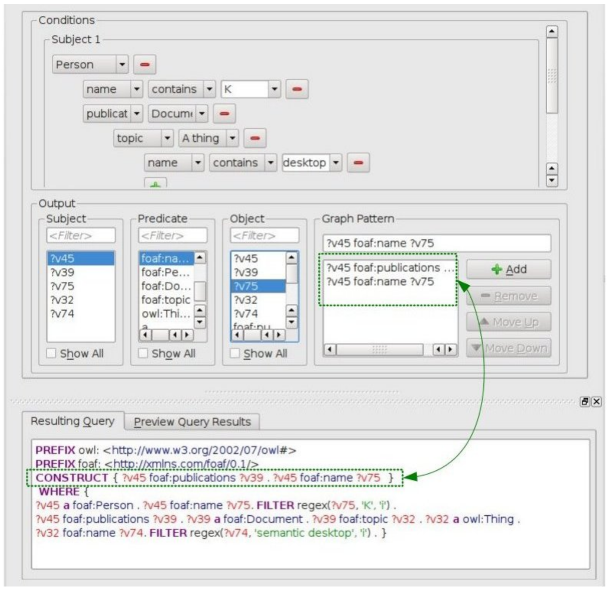
## Query Generation
Queries are generated based on the visual description according to the defined language. The SELECT part of the queries will be a set of variables extracted from the components (class combo boxes or literal text boxes) selected as $Outputs$. It is made up of the variable names in the $RootNodes$ and $LiteralNodes$, for class combo boxes or literal text boxes, respectively. This happens in a transparent way, as variables are extracted automatically from the selected components.

The WHERE clause is a set of RDF triples generated from the query tree structures present in the query form. We have the following three cases: (1) For the root $RootNode$ the generated triple is `VariableName a ClassName` and for every restriction a triple is generated starting with `VariableName` and continuing as presented in the following points (e.g. `?v41 a foaf:Person` generated for the root node in Figure 1). (2) A $ClassRestriction$ completes the parent's triple with `Predicate VariableName` (where `VariableName` is the variable of the $RootNode$ belonging to the new restriction) (e.g. `?v41 foaf: publications ?v77` generated for the `publication` restriction in Figure 1). (3) A $LiteralNode$ completes the triple with `Predicate VariableName` adding a regular expression filter string according to the chosen $Relation$ and $LiteralValue$ (e.g. `FILTER regex(?v59, 'K', 'i')` generated for the first restriction shown in Figure 1).

## User Interface
The query builder form (Figure 1) allows for adding several query trees (staring from different classes), and restricting them by their properties. If a property has a literal range, the user can enter a value and restrict it on a relation, such as `equals` or `contains`. If its range is not a literal, they can add restrictions.

Outputs are selected by right clicking on a combo box representing a class or a value. The corresponding variable will be added to the output, and the combo box will be highlighted.

## SCHEMA-BASED CONSTRUCT QUERIES
The approach to building schema-based CONSTRUCT queries is very similar to the one shown in the previous section. The difference lies in the way in which the outputs are selected.



```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
CONSTRUCT { ?v45 foaf:publications ?v39 . ?v45 foaf:name ?v75  }
 WHERE {
?v45 a foaf:Person . ?v45 foaf:name ?v75. FILTER regex(?v75, 'K', 'i') .
?v45 foaf:publications ?v39 . ?v39 a foaf:Document . ?v39 foaf:topic ?v32 . ?v32 a owl:Thing .
?v32 foaf:name ?v74. FILTER regex(?v74, 'semantic desktop', 'i') . }
```

**Figure 2. Schema-based CONSTRUCT query builder.**

## Visual Language Extension
The visual language is extended, as the output part of the queries is built using graph patterns constructed from triples.

```
Outputs      ::= GraphPattern
GraphPattern ::= Triple+
Triple       ::= Subject Predicate Object
Subject      ::= VariableName
Predicate    ::= PredicateName | VariableName | ClassName
Object       ::= PredicateName | VariableName | ClassName
        | LiteralValue
```

**Listing 2. EBNF description of the output part (non-terminals)**

## Query Generation Extension
The way outputs are generated has been changed for this query builder: in this case outputs are triples and consist of the triples described in the graph pattern.

## User Interface Extension
The user interface is extended with a component for composing output triples, as shown in Figure 2. It lists all variables for the subject field, all variables, predicates and classes for the predicate field and all variables, predicates, classes and literal values for the object field. Users can select desired triples and add them to the output list.

## INSTANCE-BASED SELECT QUERIES
Building queries with the instance-based SELECT builder relies on schema and instance information from the underlying RDF repository.

## The Visual Language
Variable, class and predicate names are IRIs describing the corresponding entities, as described for the previous builders, where the meaning of $Relation$ is also explained. Instance IRIs are taken from the repository as autocompletion popups based on user input. The $LiteralValue$ represents valid

SPARQL literals, such as strings or integers (e.g. `"Example"`). See Listing 3 for the formal EBNF description.

```
Query           ::=  Outputs  Conditions
Outputs         ::=  VariableName+
Conditions      ::=  GraphPattern
GraphPattern    ::=  Triple+
Triple          ::=  Subject  Predicate  Object
Subject         ::=  VariableName  |  ClassName  |  InstanceIRI
Predicate       ::=  VariableName  |  PredicateName
Object          ::=  VariableName  |  ClassName  |  InstanceIRI
     |  Literal  |  FilterExpression
Literal         ::=  LiteralValue  {DataType}?
FilterExpression ::=  Relation  LiteralValue
```

**Listing 3. Visual language of the instance-based SELECT query builder**

### Query Generation
Queries are constructed by enumerating the variable names for the SELECT part and taking the list of triples for the WHERE part. Filter expressions are built by adding a regular expression filter string according to the chosen relation.

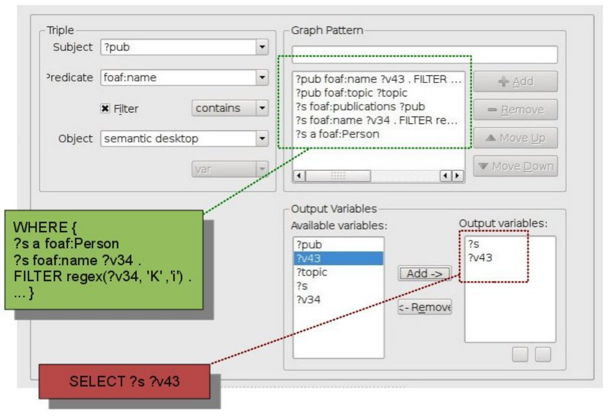
### Autocompletion
The interface features an incremental autocompletion for the WHERE part based on user input described in [9], because of the infeasibility of listing all the instances. Whenever the user types something into any of the text boxes, the system pops up all the possible options of RDF entity names, classes, properties or instance identifiers and values. This helps the user in exploring the underlying data set.

Autocompletion is achieved by running an incremental query every time the user enters text. The system queries for all entity names (classes, predicates) as well as all instance identifiers or values that contain the user input and match the graph pattern constructed so far (for the final query to have results). The user is then presented with the list of possible options, this list incrementally growing as more matches are found from the RDF repository.

### User Interface
The user interface features a component for building conditional (WHERE) triples, as shown in Figure 3. The text fields provide autocompletion popups for user input, based on what the repository contains, against the triples that have already been added to the output list. The user can select a filter option using the desired relation or can specify the type of the object, the latter defining the way it will be formatted and/or suffixed in the output.

Outputs are selected from an output list that is populated with all the variables occurring in the conditional triples.

## INSTANCE-BASED CONSTRUCT QUERIES
### Visual Language Extension
The visual language is extended with triple patterns for the output part as well. They are identical to the $GraphPattern$ nonterminal defined for instance-based SELECT queries. There is one exception, namely the lack of $FilterExpressions$, since such expressions do not exist in the output part of SPARQL queries for obvious reasons.



Subject: ?pub
Predicate: foaf:name
[✓ Filter] contains
Object: semantic desktop
var

```
?pub foaf:name ?v43 . FILTER ...
?pub foaf:topic ?topic
?s foaf:publications ?pub
?s foaf:name ?v34 . FILTER re...
?s a foaf:Person
```

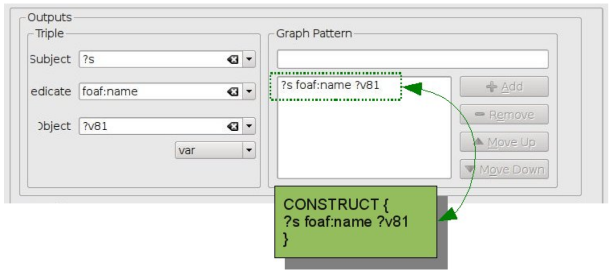
Available variables: ?pub, ?v43, ?topic, ?s, ?v34

Output variables: ?s, ?v43

```
WHERE {
?s a foaf:Person
?s foaf:name ?v34 .
FILTER regex(?v34, 'K' ,'i') .
... }
```

```
SELECT ?s ?v43
```

**Figure 3. Instance-based SELECT query builder. Resulting query is identical to the one shown in Fig. 1.**



Subject: ?s
Predicate: foaf:name
Object: ?v81
var

```
?s foaf:name ?v81
```

```
CONSTRUCT {
?s foaf:name ?v81
}
```

**Figure 4. Instance-based CONSTRUCT query builder — output part.**

### Query Generation Extension
The query generation is extended by taking the triples from the Output graph pattern and enclosing them in the CONSTRUCT part of the query.

### Autocompletion Extension
This interface also supports autocompletion for the output part of the query, similarly to its sibling interface. There is a slight modification, however, since the output triples√values do not need to satisfy any conditions, only being present in forming the output triples. Thus, it is not required to comply with the rest of the graph pattern, so all existing entities are suggested to the user, that match the given input.

### User Interface Extension
The user interface is extended with an output construction part, shown in Fig. 4, having an almost identical structure to the triple building component for the conditional (WHERE) part, excluding the filtering option.

## DISCUSSION
The schema-based SELECT query builder allows for simple querying and restricting the desired properties with user-defined input. The advantage is that it is simple, intuitive and satisfies the large number of occasions when the user wants to search for something based on certain properties. Selecting the outputs is also straightforward and clear. The disadvantage is that it is limiting and inflexible, only allowing querying as trees, thus being unsuitable for some cases a

proficient user would meet.

The schema-based CONSTRUCT query builder allows to construct triples, this being required in many cases (in Konduit, all data are RDF triples). The advantages/disadvantages are similar to its sibling approach, adding the complication of selecting the correct variables and classes for the output triples, but adding the flexibility of formatting the output.

For the instance-based SELECT query builder we have the advantages of conditioning the results in a flexible, data-driven manner (with autocompletion based on the data in the repository), using user-defined variables and types, and reusing variables. The disadvantage is that it features a direct correspondence of the underlying RDF structure, making it more complicated than the schema-based interfaces.

The instance-based CONSTRUCT query builder is the most flexible, triple-based query assistant, making it possible to compose advanced queries, with the obvious disadvantage of being the least accessible to naïve users.

## CONCLUSIONS

We have presented four techniques to assist users in building SPARQL queries to retrieve information from the ever-growing collection of semantic data. Aimed at beginners and proficient users as well, the interfaces feature a range of approaches that ease the composition of queries. The query builders are based on the local (or possibly, remote) repository, facilitating the discovery of the RDF store.

The first two approaches relied solely on schema information, helping the users to query for instances of existing classes and restricting them with the available properties. One of these is intended for writing SELECT queries by selecting a set of outputs, the other one is for CONSTRUCT queries presenting the used variables, predicates and classes in three lists for selecting the subject, predicate and object.

The other two approaches use instance information as well in providing autocompletion popups based on user input to suggest possible options, taking into consideration the triples previously added. The SELECT query builder simply allows for selecting the variables used in the query for output, and the CONSTRUCT query builder allows for composing triples from the variables used and all existing entities in the repository.

We plan to perform a usability evaluation in determining the most appropriate tool from the ones presented for building SPARQL queries within Konduit and Nepomuk-KDE. We will present it to users with no RDF/SPARQL background as well as users with deep knowledge in semantic technologies to decide on the future direction in what approach and features best suits a SPARQL query builder aimed at a fairly wide variety of users.

### Acknowledgments

The work presented in this paper has been funded (in part) by Science Foundation Ireland under Grant No. SFI/08/CE/I1380 (Líon-2) and (in part) by the European project NEPOMUK No. FP6-027705.

## REFERENCES

1. N. Athanasis, V. Christophides, and D. Kotzinos. Generating on the fly queries for the semantic web: The ICS-FORTH graphical RQL interface (GRQL). *Lecture notes in computer science*, pages 486–501, 2004.
2. J. Borsje, H. Embregts, and S. F. Frasincar. Graphical query composition and natural language processing in an rdf visualization interface, 2006.
3. T. Catarci, P. Dongilli, T. Di Mascio, E. Franconi, G. Santucci, and S. Tessaris. An ontology based visual tool for query formulation support. In *ECAI*, volume 16, page 308, 2004.
4. S. Decker and M. R. Frank. The networked semantic desktop. In *WWW Workshop on Application Design, Development and Implementation Issues in the Semantic Web*, 2004.
5. A. Fadhil and V. Haarslev. Gloo: A graphical query language for owl ontologies. In B. C. Grau, P. Hitzler, C. Shankey, and E. Wallace, editors, *OWLED*, volume 216 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2006.
6. T. Groza, S. Handschuh, K. Möller, G. Grimnes, L. Sauermann, E. Minack, C. Mesnage, M. Jazayeri, G. Reif, and R. Gudjónsdóttir. The NEPOMUK project — on the way to the social semantic desktop. In T. Pellegrini and S. Schaffert, editors, *Proceedings of I-Semantics' 07*, pages pp. 201–211. JUCS, 2007.
7. F. Hogenboom, V. Milea, F. Frasincar, and U. Kaymak. RDF-GL: A SPARQL-Based Graphical Query Language for RDF.
8. M. Jarrar and M. D. Dikaiakos. Mashql: a query-by-diagram topping sparql. In *ONISW '08: Proceeding of the 2nd international workshop on Ontologies and nformation systems for the semantic web*, pages 89–96, New York, NY, USA, 2008. ACM.
9. K. Möller. *Lifecycle Support for Data on the Semantic Web*. PhD thesis, National University of Ireland, Galway, 2009.
10. K. Möller, S. Handschuh, S. Trug, L. Josan, and S. Decker. Demo: Visual programming for the semantic desktop with Konduit. In *5th European Semantic Web Conference (ESWC2008), Tenerife, Spain*, volume 5021 of *LNCS*, pages 849–553. Springer, June 2008.
11. E. Prud'hommeaux and A. Seaborne. SPARQL query language for RDF. Recommendation, W3C, January 2008. `http://www.w3.org/TR/rdf-sparql-query/`.
12. P. R. Smart and Russell. A visual approach to semantic query design using a web-based graphical query designer. In *EKAW '08: Proceedings of the 16th international conference on Knowledge Engineering*, pages 275–291, Berlin, Heidelberg, 2008. Springer-Verlag.