INFORMATION
SYSTEMS IN
MANAGEMENT

Information Systems in Management (2018) Vol. 7 (3) 180−189
DOI: 10.22630/ISIM.2018.7.3.3

INTEGRATION OF QUERIES TO HETEROGENEOUS DATA SOURCES USING LINQ TECHNOLOGY

MARIAN RUSEK [*)], JAKUB MAGUZA, WALDEMAR KARWOWSKI [*)]

[*)] *Department of Informatics, Warsaw University of Life Sciences* (*SGGW*)

Nowadays, the data are available in a variety of formats such as relational database tables, xml files, rdf files or simply text files. Database systems have their own query languages and tools for the manipulation of data. On the other hand, most of today's applications are created in languages based on the object-oriented paradigm. From the level of the programming language it is important to use different sources of data in a uniform manner. The paper discusses the elements of the various query languages such as SQL XQuery or SPARQL. And then shows the capabilities of LINQ and its role in the creation of abstract data access layer. Then the possibilities of LINQ extension are discussed. As the example, design and implementation of LINQ provider for Allegro is presented.

Keywords: Query languages, LINQ, extension methods, .NET

1. Introduction

Today we have access to large amounts of data in digital form. The main sources of data are electronic documents, Web pages, Web services and databases. However, finding the right information is not always easy. We can search the documents using text search engines but this requires additional tedious analysis, for web pages it is possible to use search engines but it also requires additional browsing dozens of result pages. For better structured data sources, there are available tools for more precise searching. These tools provide query languages which can help us during search process. The most known query language is SQL (Structured

Query Language) a standard for communication with relational databases, standardized by ISO [1]. Query languages are important in the implementation of information management systems. From the other side, currently in system implementation the biggest role play the programming languages based on the object oriented paradigm. In general, the query languages like SQL are not integrated with object oriented programming languages. Especially between object oriented languages and relational databases, we have so called the object-relational impedance mismatch. Programmer must every time "immerse" query in the programming language used for the application. The query results must be converted to the types available in the programming language. This means that an application code is not uniform and developer needs to learn the syntax of query languages. In such situation, a solution which allows querying various data sources and easy integrates with object-oriented environment is desired.

The rest of this paper is organized as follows: in Sect. 2 the most important query languages, together with appropriate data sources are reviewed. In Sect. 3 a LINQ technology is described. In Sect. 4 an implementation LINQ to Allegro is presented. We finish with summary and brief remarks in Sect. 5.

2. Query Languages

Users want to find interesting information from the available data source. The easiest method to do this is to ask about information in natural language. Today such method is possible. User can type in a search engine sentence in natural language, sentence is parsed, key words are recognized, searching is performed and results are presented. But the use of natural language unfortunately does not give satisfactory results due to the lack of precision of the everyday spoken language. Search engines, like Google, have advanced search options which help user defining more structured query. To define precise query we need well defined query language. According to Encyclopaedia Britannica, query language, is a computer programming language used to retrieve information from a database. Wikipedia provides a broader definition of query language: it is computer language used to make queries in databases and information systems. This definition stresses that the data source can be any collection of data in information systems. Unfortunately, to ask a precise query, it is necessary to know what the data structure is. Ideally, a query language should allow users to formulate queries in an intuitive way. One such intuitive way is Query By Example (QBE) [2]. It was defined at IBM for relational databases but many query languages use this idea. There are many visual tools which help user construct query, for example Microsoft Access Query Design Grid [3]. User prepares pattern and asks for "similar" objects to be retrieved from a database, typically user fills the selected fields in the table, example object, document etc., and the database engine returns records containing the information from the filled-in fields. In query languages many depends on how the data are organized. Specific types of database have their own specific query languages. Mentioned earlier SQL is designed for relational databases and is based upon relational algebra and relational calculus of tuples. For databases from different manufacturers, there are many mutations (dialects) of SQL, with many specific extensions (Oracle PLSQL, Microsoft Transact SQL etc.). Declarative SQL syntax has become a model for other query languages. XQuery, that is, the XML Query Language, is a language created to search XML documents. XQuery is WWW Consortium standard [4]. XML documents have a hierarchical structure and the query must contain the path to the items. To do that it uses XPath, another WWW Consortium standard [5]. SPARQL is a recursive acronym for SPARQL Protocol and RDF Query Language. It is query language devoted to data in Resource Description Framework (RDF) format. Similarly to XQuery it is WWW Consortium standard [6]. RDF itself is based on the idea of making short statements about resources in the form subject–predicate–object, known as triples and was initially designed for Semantic Web [7]. We have to note that RDF data model is rather in graph than hierarchical format. Despite the fact that most of the resources in the RDF are serialized in XML format [8], SPARQL has its own syntax based on triples which is not based on XQuery. As was said earlier, object-oriented model is most popular in nowadays applications. However object-oriented databases state now the market margin and generally they have proprietary query languages. Standard for object-oriented databases - Object Query Language (OQL) [9] - was defined by Object Data Management Group but it was never fully implemented in object-oriented databases. Currently so called NoSQL databases become more popular, but they have not one data model. NoSQL databases are representing rather wide spectrum of technologies. Most popular are document databases, they are relatively close to object-oriented model but do not support inheritance. Document data bases typically use JSON (Java Script Object Notation) format or its binary form BSON. Unfortunately almost every document database has its proprietary language for queries. There is an interesting initiative for query and processing language specifically designed for the JSON data model [10], but there are almost no implementations of this proposal. Other interesting NoSQL database category is graph databases. Mentioned earlier RDF data model can be classified in this category. There is Gremlin, graph traversal language implemented in many graph databases [11] which is almost standard. At the end of this short review of query languages we have to note that search engines like Google or Bing have their own APIs for programmers. Such APIs are really query languages, but their capabilities are similar to the advanced search properties in search engine [12, 13].

In conclusion we can say that due to the very diverse data formats, there is no one standard for queries. An additional problem is that the existing query languages do not integrate well with popular programming languages. From programmers point of view the tool that allows queries in an object-oriented manner is needed, to provide good integration of programming language with different data sources.

3. LINQ

LINQ (Language INtegrated Query) is a set of tools to analyze and manipulate data collections. It is a Microsoft .NET Framework component that adds native data querying capabilities to .NET languages, although ports exist for other programming languages and environments. This technology came into use in 2007 with the third version of the C# language as part of the .NET Framework 3.5 [14, 15, 16]. LINQ technology is based on generic types, collections which implements IEnumerable interface, extension methods, anonymous types, delegates and lambda expressions. The first implementation was LINQ to objects and extension methods were defined for enumerable collections of objects. The example presented in fig.1 illustrates LINQ to objects idea.

```
var longerWords = "SHORT TEXT TO LINQ ANALYSIS".Split()
        .Where(x => x.Length > 4).OrderBy(x => x.Length)
        .Select(x => new { word = x, length = x.Length });
```

Figure 1. LINQ to objects example

In the example method Split() divides string into array of words (strings) using space as separator. Array in .NET is IEnumerable and extension method Where can be called on it. Method Where as an argument takes lambda expression (x => x.Length > 4). Result is enumerable and we can call another extension method OrderBy with next lambda expression, and finally call extension method Select with lambda expression, this last lambda expression creates anonymous type: pair of word and its length. The most important methods are Where and Select. We have to note that there are defined the contextual keywords used in query expressions, equivalent to most important LINQ methods. We can rewrite discussed code in declarative style (fig.2).

```
var longerQuery = from w in "SHORT TEXT TO LINQ ANALYSIS".Split()
    where w.Length > 4 orderby w.Length ascending
    select new { word = w, length = w.Length };
```

Figure 2. LINQ to objects declarative style example

There are additional LINQ implementations in .NET framework: LINQ to XML, LINQ to SQL, LINQ to DataSet and LINQ to Entities. There is another issue connected with LINQ implementations, large external data sources cannot be loaded whole into memory. We have to delay the loading data from the file until this data will be actually needed. It is essential that such operations should be performed automatically. In .NET framework was introduced keyword yield and thanks to it we have the automatic deferred execution. For our purpose most interesting is the LINQ to SQL provider. It allows LINQ to be used to query Microsoft SQL Server databases (or other relational databases with appropriate drivers). Since SQL Server data may reside on a remote server, and because SQL Server has its own query engine, LINQ to SQL does not use the query engine of LINQ to objects. Instead, it converts a LINQ query to a SQL query that is then sent to SQL Server for processing. Interface IEnumerable is used to iterate through the collection of objects and is not useful for external data source. For external data sources like relational database the IQueryable interface was defined [17]. It is useful for building a query that is translated into language understandable for a given source for example SQL. IQueryable implements IEnumerable so all the LINQ extension methods are available. Interface IQueryable (fig.3) defines two important fields of type Expression and IQueryProvider (fig.4).

```
public interface IQueryable : IEnumerable {
        Expression Expression { get; }
        Type ElementType { get; }
        // the provider that created this query
        IQueryProvider Provider { get; }
    }
```

Figure 3. Inteface IQueryable. *Source*: [17]

Abstract Expression class has a few dozen subclasses. Most important are: BinaryExpression, ConstantExpression. LambdaExpression, MemberExpression, MethodCallExpression and UnaryExpression. Especiall BinaryExpression gives possibility to build binary tree, such expression trees are at the core of the LINQ extensibility mechanism.

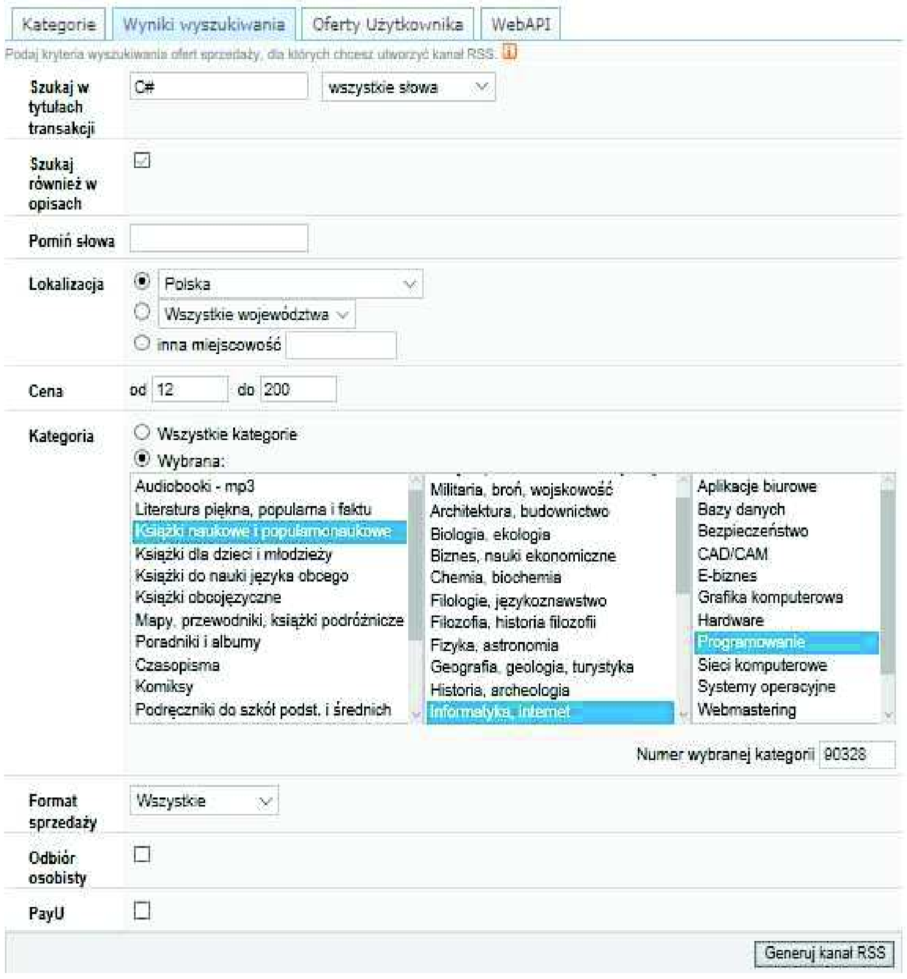
```
public interface IQueryProvider{
  IQueryable CreateQuery(Expression expression);
  IQueryable<TElement> CreateQuery<TElement>(Expression expression);

  object Execute(Expression expression);
  TResult Execute<TResult>(Expression expression);
    }
```

Figure 4. Inteface IQueryProvider. *Source*: [17]

Provider has two methods CreateQuery and ExecuteQuery (fig.4). The expression tree is handed over to LINQ provider, which adapt the LINQ queries to be used with the data source. The main task during LINQ implementation is to define proper provider, particularly Execute method. Moreover implementer has to prepare extension method for at least Where clause.

4. LINQ to Allegro

Allegro is a Polish online auction website. It primarily allows the sale of items owned by the users. Potential buyers can search the site for items of interest to them. Our goal was to create a custom LINQ provider to be able to browse the service resources for buyers.



Podaj kryteria wyszukiwania ofert sprzedaży, dla których chcesz utworzyć kanał RSS.

Szukaj w tytułach transakcji	C# — wszystkie słowa
Szukaj również w opisach	✓
Pomiń słowa	
Lokalizacja	[✓ Polska] / Wszystkie województwa / inna miejscowość
Cena	od 12 do 200
Kategoria	Wszystkie kategorie / [✓ Wybrana:]
Format sprzedaży	Wszystkie
Odbiór osobisty	☐
PayU	☐

Audiobooki - mp3	Militaria, broń, wojskowość	Aplikacje biurowe
Literatura piękna, popularna i faktu	Architektura, budownictwo	Bazy danych
[✓ Książki naukowe i popularnonaukowe]	Biologia, ekologia	Bezpieczeństwo
Książki dla dzieci i młodzieży	Biznes, nauki ekonomiczne	CAD/CAM
Książki do nauki języka obcego	Chemia, biochemia	E-biznes
Książki obcojęzyczne	Filologie, językoznawstwo	Grafika komputerowa
Mapy, przewodniki, książki podróżnicze	Filozofia, historia filozofii	Hardware
Poradniki i albumy	Fizyka, astronomia	[✓ Programowanie]
Czasopisma	Geografia, geologia, turystyka	Sieci komputerowe
Komiksy	Historia, archeologia	Systemy operacyjne
Podręczniki do szkół podst. i średnich	[✓ Informatyka, internet]	Webmastering

Numer wybranej kategorii 90328

Figure 5. Allegro RSS channels generator page. *Source*: [18]

Allegro allows the exchange of information between service and external software and developers can integrate their own solutions with the Allegro data. Allegro provides three ways to take advantage of the service. The first is WebAPI - web service based on SOAP protocol, the second is REST API and the third RSS channels. The first two possibilities are requiring an account on the Allegro and they intended for commercial use by companies offering products for sale. For our purpose best proved ids RSS service available at [18]. It does not require creating an account and is totally free. In the figure 5, visual interface for RSS channels generator is presented. It is easy to generate URL string which can be use as the pattern during generating LINQ queries. Parameters after which we can search for are: Title (name of item), SearchInDescription, ExcludeWords, Price, CategoryID, OfferType, SearchType, CollectInPersonaly, PayU. Those fields can be used in the Where clause. Unfortunately, there is one problem with CategoryID. Allegro uses so many categories and subcategories, that the whole category tree consists of thousands of items. Every category has its own identifier. Preparing the query we need to choose the proper identifier corresponding to the category with the help of the WebAPI methods or REST API, it is easier than using the RSS feed. This is not problem for LINQ implementation but only for an application that uses it. During testing we had to limit categories to a few basic ones. Example of generated pattern (fig.6) includes category books about programming (90328).

```
http://allegro.pl/rss.php/search?string=C%23&category=90328&price_from=1
2&price_to=200&description=1&selected_country=1&search_type=1&postcode_e
nabled=1
```

Figure 6. Allegro RSS pattern

```
<item> <title>C# SPECYFIKACJA JĘZYKA - NOWA</title>
<guid isPermaLink="false">http://allegro.pl/c-specyfikacja-jezyka-nowa-
i7022874587.html</guid>
<link>http://allegro.pl/c-specyfikacja-jezyka-nowa-
i7022874587.html</link>
<pubDate>Sun, 29 Oct 2017 11:05:26 +0100</pubDate>
<description>Sprzedający: <a
href="http://allegro.pl/show_user.php?uid=17637436">BARCELONAS55</a><br
/> Cena Kup Teraz: 29,00 zł<br /> Do końca: 29 dni (wto, 28 lis 2017,
11:05:26)<br /> <a href="http://allegro.pl/c-specyfikacja-jezyka-nowa-
i7022874587.html">Kup Teraz</a><br /> <br /> </description></item>
```

Figure 7. Item from Allegro response

In the response we receive list of items. Each item has the form as in the figure 7. As we can see in every item we have only title (item name), pubDate (date of publication), description and link (item URI). Description includes some additional information like price or auction deadline date, but to read them other actions are needed. This means that we can select only the mentioned four fields contained in the response.

Implementation of LINQ to Allegro is described in [19] it is influenced by the example implementations provided by Microsoft [20]. Main LINQ to Allegro classes are presented in the figure 8. Most important class is AllegroQueryable which implements IQueryable interface [21]. Additional subclass of Expression was not created, because functionality of BinaryExpression and LambdaExpression was enough.

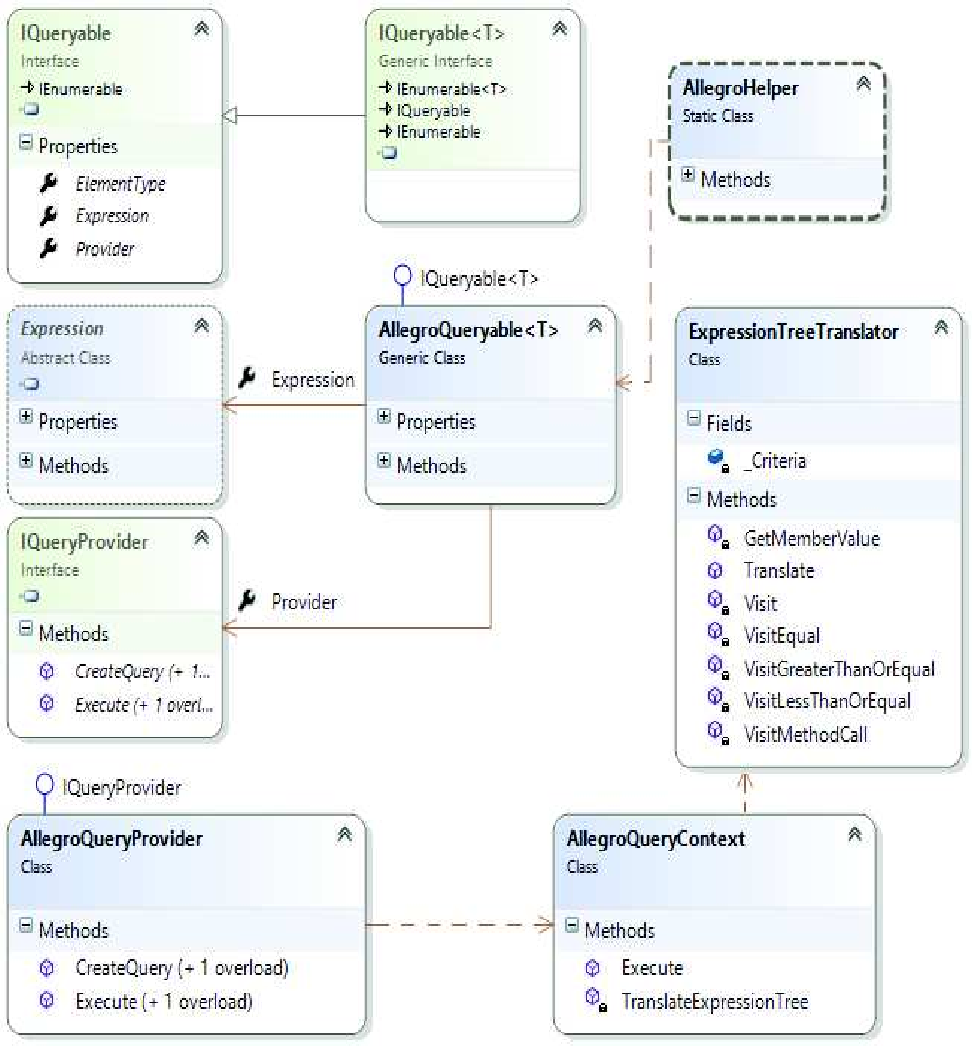


Figure 8. LINQ to Allegro classes

```
class AllegroQueryContext
{
    public static object Execute(Expression expression)
    {
        var stringURL =
        new AllegroQueryContext().TranslateExpressionTree(expression);
        XElement _allegroRSS = XElement.Load(stringURL);
        return AllegroHelper.ParseAllegroXml(_allegroRSS);
    }
    private string TranslateExpressionTree(Expression expression)
    {
        return new ExpressionTreeTranslator().Translate(expression);
    }
}
```

Figure 9. AllegroQueryContext class. *Source*: [2]

Important part of functionality is in AllegroQueryProvider class which implements IQueryProvider interface. AllegroQueryProvider uses AllegroQueryContext class which is responsible for connection with Allegro RSS service (fig.9). Additionally AllegroQueryContext class has TranslateExpressionTree method uses ExpressionTreeTranslator class which with method Visit translates parts of expression and extracts conditions from Where clause. Visit calls VisitEqual method for every parameter: Title, SearchInDescription, ExcludeWords, Price, CategoryID, OfferType, SearchType, CollectInPersonaly, PayU. VisitEquals has inside analysis of conditions for every parameter. More details connected with implementation are available at [21].

5. Conclusions and future work

Example implementations provided by Microsoft allow programmer to create his own providers for each data source having its specific query language or even simple interface. Those examples helped in LINQ to Allegro implementation. Tests have shown that implementation of the LINQ to Allegro works and queries integrate with C# programming language in .NET environment. On the other hand, the implementation is quite extensive, it was necessary to “packaging” simple operation in a fairly extensive class structure. Quite poor Allegro RSS service has caused that the functionality of LINQ to Allegro is limited. Main part of implementation was Where clause. The response data are in the XML format and it would be possible to implement much less complicated solution devoted to search Allegro data. However such solution would lose a unified query style. In the future work, we should extend our implementation by improving Where clause and by Select clause filtering data like price or auction deadline date now included in description.

REFERENCES

[1] ISO/IEC 9075-1:2016 standard: https://www.iso.org/committee/45342/x/catalogue/p/1/u/0/w/0/d/0.

[2] Zloof, M., (1975), *Query by Example*, Conference: American Federation of Information Processing Societies: 1975 National Computer Conference proceedings, 19-22 May 1975, Anaheim, CA, USA.

[3] Introduction to queries: https://support.office.com/en-us/article/Introduction-to-queries-D85E4893-0ED7-4118-8297-785A01357516 (access of 15 November 2017).

[4] XQUERY specification: http://www.w3.org/XML/Query.

[5] XPATH specification: http://www.w3.org/TR/xpath.

[6] SPARQL specification: http://www.w3.org/TR/sparql11-query.

[7] RDF current status: http://www.w3.org/standards/techs/rdf#w3c_all.

[8] RDF 1.1 XML Syntax recommendation: http://www.w3.org/TR/rdf-syntax-grammar.

[9] Cattell R.G.G., Barry D.K. (2000) *The Object Data Standard: ODMG 3.0.* The Morgan Kaufmann Series in Data Management Systems.

[10] The JSON Query Language; http://www.jsoniq.org (access of 15 November 2017).

[11] Apache TinkerPop. The Gremlin Graph Traversal Machine and Language: https://tinkerpop.apache.org/gremlin.html (access of 15 November 2017).

[12] Google Custom Search Tutorial: https://developers.google.com/custom-search/docs/tutorial/introduction (access of 15 November 2017).

[13] Bing Query Language: https://msdn.microsoft.com/en-us/library/ff795667.aspx (access of 15 November 2017).

[14] Language Integrated Query: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/index (access of 15 November 2017).

[15] Marguerie F., Eichert S., Wooley J. (2008) *LINQ in Action*, Manning.

[16] Mukherjee S. (2014) *Thinking in LINQ: Harnessing the Power of Functional Programming in .NET Applications* 1st ed. Edition. Apress.

[17] Microsoft Reference Source .NET Framework 4.7.1: http://referencesource.microsoft.com/#System.Core/System/Linq/IQueryable.cs,eb17cf64586dbd9b (access of 15 November 2017)

[18] Allegro RSS Channels Generator: http://allegro.pl/rss.php/generatorSearch (access of 15 November 2017)

[19] Maguza J. (2017) *LINQ implementation for Allegro system*, MSc thesis, Warsaw University of Life Sciences (in Polish).

[20] Walkthrough: Creating an IQueryable LINQ Provider https://msdn.microsoft.com/en-us/library/bb546158.aspx (access of 15 November 2017).

[21] https://github.com/jakubmaguza (access of 15 November 2017).