

KGOL: A Knowledge Grid Operating Language

Hai Zhuge and Jie Liu

Knowledge Grid Research Group, Key Lab of Intelligent Information Processing,
Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
 {zhuge, lj}@ict.ac.cn

Abstract: This paper presents the design and implementation of a Knowledge Grid operating language and the programming environment KGOL. It provides not only a friendly user interface for end-users to easily access and manage the Knowledge Grid resources but also a programming environment for application developers to implement the Knowledge Grid applications. The KGOL programming environment consists of a parser, an interpreter, an execution engine, and a result generator. Comparisons between the KGOL, the SQL, and the XML-based query languages XQL and LOREL show the distinguished advantages of the KGOL.

Keywords: Knowledge Grid, Grid Operating Language, XML, Web

1 Introduction

The Semantic Web, Web Service and Grid are three ways to the next-generation Web. The Semantic Web is to make the Web resources machine-understandable by using the markup languages like XML, RDF, and DAML (<http://www.semanticweb.org>) so as to enable intelligent Web Services. Web Services are applications that interact with each other using Web standards such as XML, SOAP (Simple Object Access Protocol), WSDL (Web Service Description Language) and UDDI (Universal Description Discovery and Integration). The purpose is to provide an open platform for the development, deployment, intelligent interaction, and management of globally distributed e-services. The Grid aims at the controlled sharing, coupling, and managing of globally distributed resources based on the high-performance computing environment (<http://www.gridforum.org>).

Knowledge Grid is a Grid mechanism based on the Semantic Web for uniformly sharing and managing versatile knowledge resources across the Internet. The knowledge resources include information, services and knowledge. The Knowledge Grid organizes knowledge resources in a Knowledge Space, a three-dimensional resource space [16]. Each point in the Knowledge Grid space represents resources at a certain location, a certain knowledge level and a certain knowledge category. The prototype of the Knowledge Grid, named SVEGA-KG, has been implemented and is available at <http://kg.ict.ac.cn>.

The Knowledge Grid operating language and the programming environment KGOL is an important portion of the SVEGA-KG. The KGOL includes two kinds of user interfaces: a

friendly interface for end-users to use the KGOL statements separately to carry out resource operations, and a programming environment for application developers to implement the Knowledge Grid applications. A KGOL program is composed of a sequence of KGOL statements, KGOL functions, and the control-statements [3], [9], [14]. The KGOL programming environment includes the following major components.

1. An event-based parser, which parses an KGOL program and then generates a source tree.
2. An interpreter, which travels the source tree to get the operators and the corresponding parameters.
3. An execution engine, which performs the KGOL statements and generates a typed output instance.
4. A result generator, which generates a result in the form of a complete XML document or a set of XML fragments from the output instance.

2 KGOL Grammar Specifications

The syntax and semantics of the KGOL are designed by referring to that of the SQL. For example, the statement of getting knowledge resources from a Knowledge Grid (*KG*) at the *Concept* level and the *Software* category can be expressed as follows:

```
GET K from KG
WHERE level="Concept" and category="Software".
```

A major form of the KGOL statement is described as follows:

```
<KGOLStatement>: :=<Operator>[ALL | DISTINCT][Resource]
                [FROM|INTO] <Knowledge Grid Name>[AT UGL]
                [WHERE <Condition-Expression>]
                [SORT BY <Knowledge Grid Element>][ASC | DESC];
<Operator>: := Get | Put | Delete | Undelete;
<Condition-Expression>: :=Level-coordinate-expression
                        |Category-coordinate-expression
                        |Keywords-expression | Name-expression.
```

The Condition-Expression represents a Boolean expression specifying the level coordinate and the category coordinate of the Knowledge Grid as well as the other constraints such as keyword patterns [7]. The KGOL also has the *assign*-statement, the *create*-statement, the *browse*-statement, the *update*-statement, the *join*-statement, the *open*-statement, and the *log*-statement. The detailed KGOL grammar is given in appendix A and the meaning of each statement is described as follows:

1. The *get*-statement is used to retrieve the Knowledge Grid resources at a UGL satisfying the Condition-Expression. A *get*-statement can be embedded in another KGOL statement by replacing the UGL with the *get*-statement. The nested query can retrieve resources distributed at several Knowledge Grids as if they were in one Knowledge Grid [11]. The result of each operation is a temporary Grid view expressed by a complete XML document or a set of XML fragments.

2. The *put*-statement is used to put the resources into a Knowledge Grid at a UGL. The Condition-Expression determines the level coordinate and the category coordinate of the Grid. The *assign*-statement is usually used before carrying out the *put* operation.
3. The *delete*-statement is used to delete useless or redundant resources of a certain Knowledge Grid at a UGL satisfying the given condition. The option portion “*PACK*” determines the resources previously deleted logically to be deleted permanently.
4. The *undelete*-statement is used to undelete resources that are not permanently deleted.
5. The *assign*-statement is used to bind the value of the expression to a variable.
6. The *create*-statement is used to create a new Knowledge Grid at a Universal Grid Location (UGL). The information of the level coordinate and the category coordinate is acquired from the Condition-Expression portion. A new Knowledge Grid can be created by inheriting from an existing one with structure or content inheritance. The option portion “*With Structure&Content*” indicates the inheritance of both.
7. The *browse*-statement is used to browse a view of the specified Knowledge Grid. The Condition-Expression restricts the level range and the category range of the Grid.
8. The *update*-statement is used to update the resources of the Knowledge Grid at a UGL satisfying the Condition-Expression, with a *set-clause* specifying the update changes.
9. The *join*-statement is used to join different Grids into a new Knowledge Grid. With the *join*-statement, users can query a collection of inter-related Grids.
10. The *open*-statement is used to open the whole or a part of the Knowledge Grid satisfying the Condition-Expression to the other Grids.
11. The *log*-statement is used to log a Knowledge Grid at a UGL on or off the worldwide Grid. The universal resource view does not include the resources in the Grid that has been logged off.

3 KGOL Programming

3.1 An Example of KGOL program

Let us start the discussion with a simple example of the KGOL program:

Begin

```
CREATE G=(Knowledge Grid, Knowledge Grid, Information Grid, Service Grid)
At http://kg.ict.ac.cn/Research/newGrid;
GET K from Grid WHERE level="Concept" and category="Grid" and name="Grid";
PUT K into new Grid WHERE level="Concept" and category="Knowledge Grid";
GET K from new Grid WHERE level="Concept" and category="Knowledge Grid";
```

End

This program implements the following functions:

1. Create a new Grid named *new Grid* at <http://kg.ict.ac.cn/Research>. The default four level coordinates are “*Concept*”, “*Axiom*”, “*Rules*”, and “*Method*”. Simultaneously, the four category coordinates are “*Semantic-Grid*”, “*Knowledge-Grid*”, “*Information-Grid*” and “*Service-Grid*”.
2. Get the knowledge resource from the *Grid* at the “*Concept*” level and the “*Grid*” category.
3. Put the knowledge resource into the *new Grid* at the “*Concept*” level and the “*Knowledge-Grid*” category.

4. Get the knowledge resource from the *new Grid* to testify the correctness of the above three operations.

3.2 KGOL Programming Interface

A KGOL program is a composition of the KGOL statements. The control flow of the KGOL is specified as follows:

```

Sequential-process: <KGOLStatement>; {<KGOLStatement>;}
Branch-statement:  IF <Condition-Expression>
                   THEN<KGOLStatement>
                   ELSE<KGOLStatement>
                   END IF;
Loop-statement:    DO <KGOLStatement>
                   WHILE <Condition-Expression>;
Begin-End-statement: {Sequential-process}

```

Fig. 1 shows the interface of the KGOL programming environment, which includes a programming area in the center portion and four operation buttons at the bottom. The “LOAD” button is for loading a KGOL program. The “RUN” button is for executing the KGOL program. The “RESET” button is for clearing the programming area and the “SAVE” button is for storing the current KGOL program.

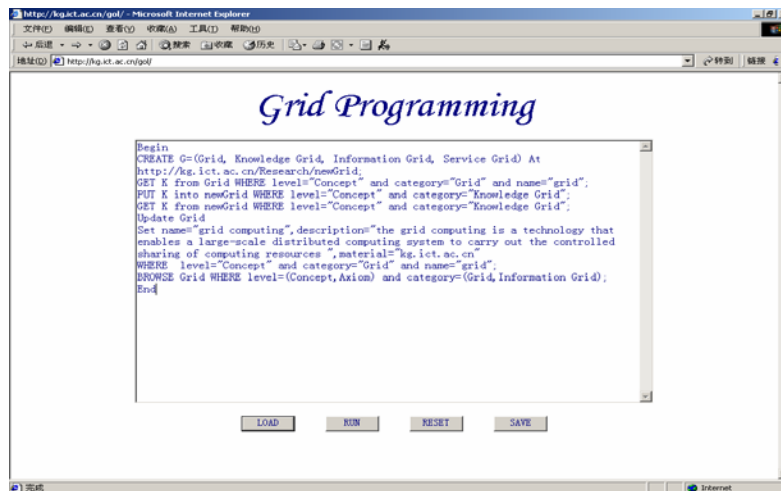


Fig. 1. Interface of the KGOL programming environment

4 Programming Environment

4.1 General Architecture

The general architecture of the KGOL programming environment is illustrated in Fig. 2. Compared with the architecture of the research prototypes introduced in [6], [8], [10], [15], the distinguished feature of the KGOL is that it is based on a Knowledge Grid resource space. The event-based parser checks and parses the input KGOL program syntactically and then

generates a source tree. The interpreter travels the source tree to acquire all the operators and the corresponding parameters, which are validated by using instruction templates. The execution engine performs the interpreted operators and produces a typed output instance.

The result generator generates a result of an XML document or XML fragments according to the output instance and then shows it with the XSL.

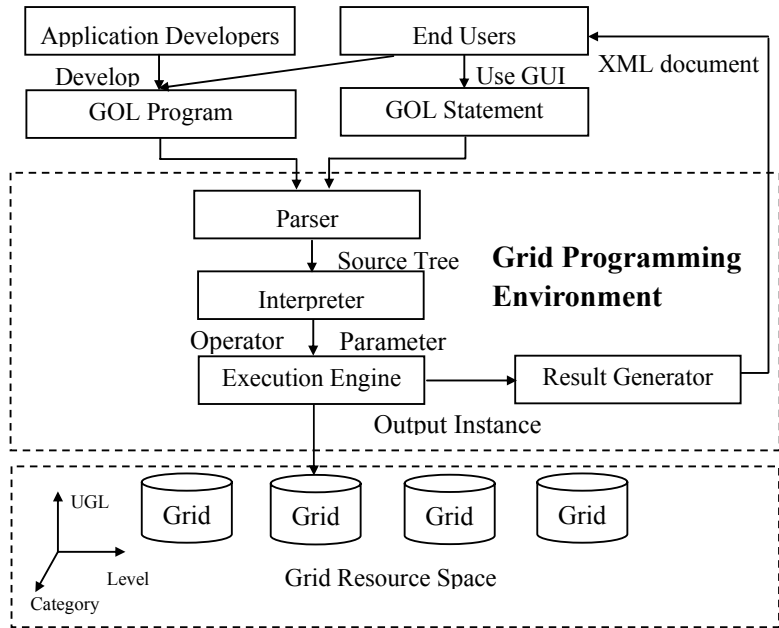


Fig. 2. General architecture of the Knowledge Grid programming environment

4.2 Parser: Generator of Source Tree

The parser is responsible for analyzing the KGOL program syntactically and generates a source tree as shown in Fig. 3. The KGOL program is stored in the form of the source tree, an abstract command tree, described by the XML [1], [2], [4], [5]. The nodes of white rectangles represent the element nodes and the others of gray rectangles represent the terminal value nodes. The execution of the KGOL program is a process of parsing, interpreting, executing, and generating the source tree.

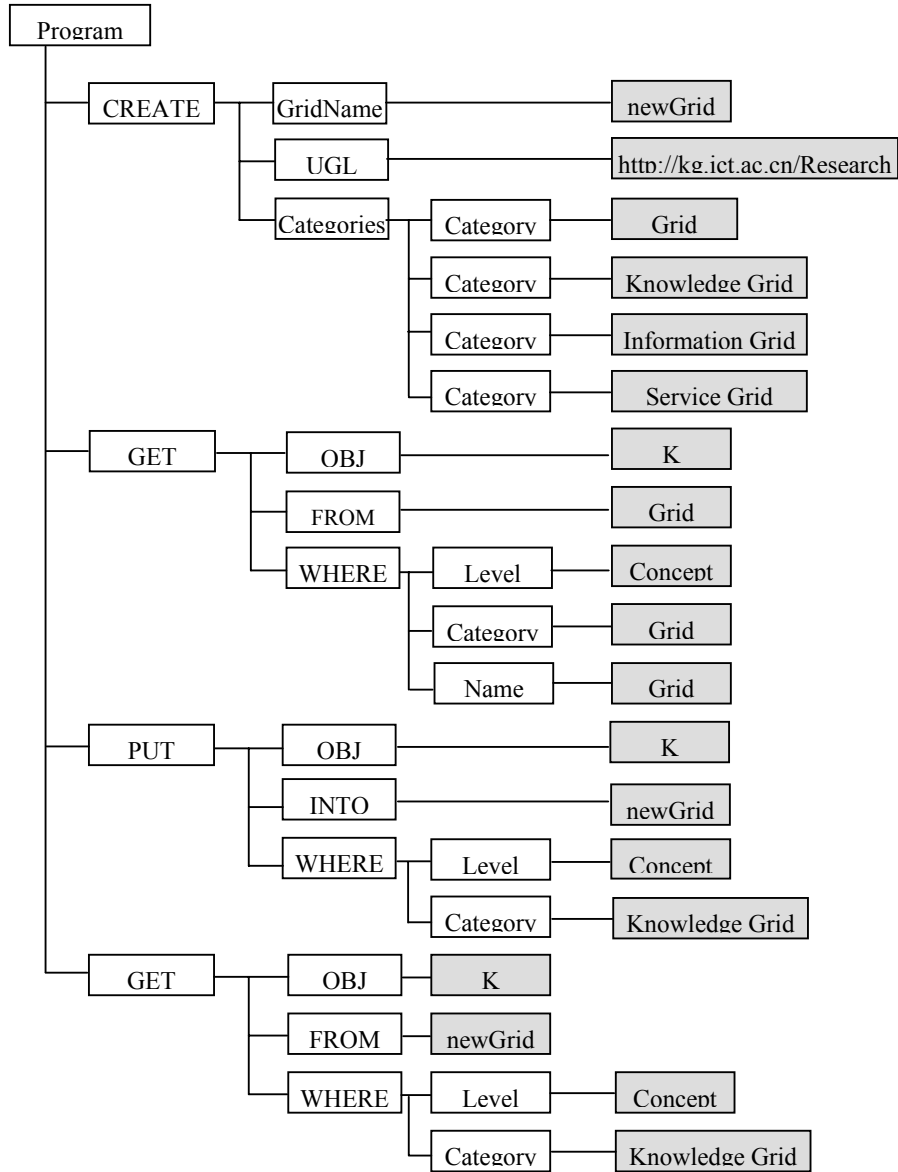


Fig. 3. Source tree generated from the input GOL program

4.3 Interpreter

The interpreter travels the source tree in the breadth-first and then in the depth-first search order to acquire the operators and the corresponding parameters in the KGOL program. The main interpreting algorithm is as follows:

```

BEGIN
  Load the Source Tree;
  Travel the Source Tree in breadth-first search order;
  Get the set S of all the Operators;

```

```

FOR each Operator OP in the set S
  Find the matching instruction from the Instruction Template;
  IF find THEN
    Travel the sub tree of node OP in depth-first
    search order;
    Get all the parameters of OP;
  ELSE Generates Error;
  END IF
NEXT
END .

```

4.4 Execution Engine

The Execution Engine performs the operations after acquiring the operators and the corresponding parameters of the KGOL program. The main execution process includes three steps: Knowledge Grid locating, Knowledge Grid loading, and API invoking. Fig. 4 depicts the execution process of the KGOL.

1) Knowledge Grid Locating

Before carrying out each parsed operation, the execution engine first locates the corresponding Knowledge Grids. Relational tables: *GridCategory*, *Grid* and *Category* are used to implement the locating of the relevant Grids. Each table has an ID and other associated attributes.

2) Knowledge Grid Loading

Knowledge Grid loading is to load the relevant Grids from the distributed Knowledge Grid resource space. The *load* method of the XML DOM interface (<http://www.w3.org/DOM/>) loads the corresponding XML documents from the specified location.

3) API Invoking

After loading the Knowledge Grids, the Execution Engine calls the corresponding API of different Function Modules and generates a typed output instance of each parsed operation.

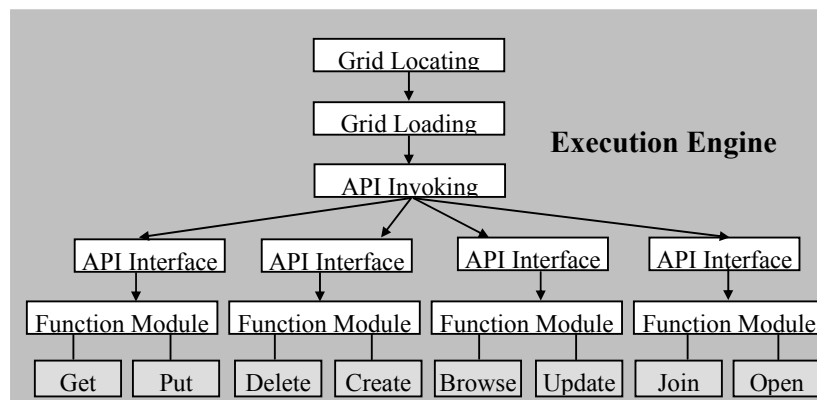


Fig. 4. Process of KGOL program execution

4.5 Result Generator

The result generator is responsible for applying the XSL to match nodes in the output instance generated by the execution engine. It can generate complex data and multiple types of XML outputs. The result is in the form of either a complete XML document or a set of XML fragments representing the sub-results.

5 Implementation

The KGOL programming environment has been implemented by using JSP, ASP and Java. The demo of the environment is available at <http://kg.ict.ac.cn/KGOL/demo>. A friendly GUI for end-users to separately use KGOL statements to operate resources has been implemented in the Knowledge Grid platform SVEGA-KG, which is available for public use at <http://kg.ict.ac.cn>.

The XML is adopted to represent the worldwide and the local Knowledge Grid. An XML DOM parser is used to construct an in-memory parsed tree for the whole documents. Currently, the KGOL program runs on the Windows2000 Advanced Server with an RDBMS of SQL-Server 7.0, and the client programs run separately on the PCs distributed on the Internet.

6 Comparisons

The KGOL is not only an SQL-like language but also a programming environment, based on a semi-structured data model combined with the XML syntax. The KGOL borrows the syntax and semantics from the standard SQL language. The statements of the KGOL are SQL-like and have the SQL SELECT-FROM-WHERE pattern. The KGOL can perform the operations similar to the classical relational database operations, such as nested queries, aggregates, set operations, join and result ordering.

The KGOL also borrows the following features from the XML query languages [3]:

1. Management of structured and semi-structured data.
2. Support abstract data types.
3. The XML-based data format and the result semantics.
4. Support the skelom functions to associate a unique ID to a given Knowledge Grid.
5. Support document selection.
6. Support partial specified path expression.

A distinguished feature of the KGOL is that it can operate various types of hierarchical resources including information, knowledge, and services with a universal resource view. The specific data model of the KGOL is a three-dimensional resource space. But, the standard SQL only operates flat relational tables, and the XML query languages only manage XML documents. Another feature of the KGOL is that it is an update language with control statements.

7 Conclusions and Future Work

This paper presents the design and implementation of the KGOL, an XML-based Knowledge Grid resource operating language and programming environment, for both application developers and end-users to implement Knowledge Grid applications. The comparisons of the KGOL, the XQL, the LOREL and the SQL show that the KGOL is an easy tool for accessing and managing resources with a mobile way and a universal resource view. The prototype of the KGOL has been implemented in the platform SVEGA-KG (<http://kg.ict.ac.cn>).

Future works mainly include three aspects: extending the expressiveness of the language [12], [13], [17] supporting the coming standard of the Semantic Web, and embedding the KGOL in other host languages.

References

1. M. Altinel and M.J. Franklin, Efficient Filtering of XML Documents for Selective Dissemination of Information, Proceedings of the 26th VLDB Conference, Cairo, Egypt May (2000) 53-64.
2. D. Beech, A. Malhotra and M. Rys, A Formal Data Model and Algebra for XML, Sept. (1999), available at <http://elib.cs.berkeley.edu/seminar/2000/20000207.pdf>.
3. A. Bonifati and S. Ceri, Comparative Analysis of Five XML Query Languages, SIGMOD Record 29 March (2000) 68-79.
4. V. Christophides, S. Cluet and J. Simeon, On Wrapping Query Languages and Efficient XML Integration, ACM SIGMOD Dallas TX USA May (2000) 141-152.
5. A. Deutsch, et al., A Query Language for XML, May 1999, the 8th International WWW conference, Toronto May (1999) 77-91.
6. P. Fankhauser, XQuery Formal Semantics State and Challenges, SIGMOD Record, 30 Sept. (2001) 15-19.
7. D. Florescu, D. Kossmann and I. Manolescu, Integrating Keyword search into XML query processing, the 9th International WWW conference, Amsterdam, May (2000) 119-135.
8. W. Han, D. Buttler, C. Pu Wrapping Web Data into XML, SIGMOD Record 30, Sept. (2000) 33-38.
9. D. Lee and W.W. Chu, Comparative Analysis of Six XML Schema Languages, SIGMOD Record 29, Sept. (2000) 76-87.
10. J. McHugh and J. Widom, Query Optimization for XML, Proceedings of the 25th VLDB Conference, Edinburgh, Scotland, Sept. (1999) 315-326.
11. E. Spertus and L. A. Stein, Squeal: A Structured Query Language for the Web, 2000, the 9th International WWW conference, Amsterdam, May (2000) 95-103.
12. D. Suciu, On Database Theory and XML, SIGMOD Record 30, Sept. (2001) 39-45.
13. I. Tatarinov, et al., Updating XML, Proceedings of the 2001 ACM SIGMOD International Conference of Management of Data, May (2001) 413-424.
14. H. Zhuge, Inheritance Rules for Flexible Model Retrieval, Decision Support Systems 22, (1998) 379-390.

- 15.H.Zhuge, A Problem-Oriented and Rule-Based Component Repository, the Journal of Systems and Software 50 (2000) 201-208.
- 16.H.Zhuge, A Knowledge Grid Model and Platform for Global Knowledge Sharing, Expert Systems with Application 22 (2002).
- 17.H.Zhuge, VEGA-KG: A Way to the Knowledge Web, 11th International World Wide Web Conference, poster proceeding, May, Hawaii, USA, 2002.

Appendix A. KGOL Grammar

The syntax of the KGOL is defined in an extended BNF grammar, where “[]” means by default, G_i represents a Knowledge Grid, and the capital words are retained by the KGOL.

1. Get-statement: GET [ALL| DISTINCT] $R_1 \cdots R_m$ FROM $G_1 \cdots G_m$
 [AT UGL₁...UGL_m][WHERE<Condition-Expression>]
 [SORT BY<Element>][ASC|DESC];
2. Put-statement: PUT $R_1 \cdots R_m$ INTO $G_1 \cdots G_m$ [AT UGL₁...UGL_m]
 [WHERE <Condition-Expression>];
3. Delete-statement: DELETE FROM $G_1 \cdots G_m$ [AT UGL₁...UGL_m]
 [WHERE <Condition-Expression>][PACK];
4. Undelete-statement: UNDELETE FROM $G_1 \cdots G_m$ [AT UGL₁...UGL_m]
 [WHERE <Condition-Expression>];
5. Assign-statement: Assign [type] R1=[type] R2;
6. Create-statement: CREATE G=(Category₁, Category₂...Category_n) [AT UGL]
 CREATE G By Inherit from G_i [AT UGL][With Structure&Content];
7. Browse-statement: BROWSE G_i [AT UGL] [WHERE <Condition-Expression>];
8. Update-statement: UPDATE G_i [AT UGL]
 SET<Element₁>=<Expression₁>[...<Element_m>=<Expression_m>]
 [WHERE <Condition-Expression>];
9. Join-statement: JOIN $G_1 \cdots G_m$ [AT UGL₁...UGL_m] INTO G_i
 [AT UGL][WHERE <Condition-Expression>];
10. Open-statement: OPEN G_i AT UGL TO $G_1 \cdots G_m$ [AT UGL₁...UGL_m]
 [WHERE <Condition-Expression>];
11. Log-statement: LOG $G_1 \cdots G_m$ [AT UGL₁...UGL_m] [ON|OFF]
 [WHERE <Condition-Expression>];